Dynamic Defense In-Depth Model to Enhance SDN Security

By

Mahmoud F. Elhejazi

Supervisor

Dr. Mohamed Musbah

This Thesis was submitted in Partial Fulfillment of the Requirements for Master's Degree of Science in Computer Science

Discussed on 22.10.2022

University of Benghazi

Faculty of Information Technology

2022

University of Benghazi Faculty of Information Technology

Department of Computer Networks

Dynamic Defense In-Depth Model to Enhance SDN Security

By
Mahmoud F. Elhejazi

This Thesis was Successfully Defended and Approved on **22 .10.2022**

Supervisor
Dr. Mohamed Musbah

Signature**: .....**

Dr. Mohamed Younis (Internal **examiner)**

Signature: …….……

Dr. Faraj Sallabi (External **examiner)**

Signature: ……...…….

Dean of Faculty	**Director of Graduate studies and training**
Dr. Abdelsalam Maatuk	**Dr. Othman M.ALbadri**
Signature:……	Signature: …..……

DEDICATION

I would like to express my gratitude to Allah for giving me the ability and strength to reach this experience of my life.

I would also like to thank all who support me, especially my mother, father, and siblings for their support and guidance to complete this work.

Mahmoud F. Elhejazi

ACKNOWLEDGMENT

I would like to take this opportunity to thank all those who contributed to the completion of this work.

I would like to express my heartfelt gratitude to my supervisor Dr. Mohamed Musbah for giving me all the crucial academic and research experience to complete this work professionally. In addition, helping to identify the critical elements of the academic reach and methods to apply to this work.

Mahmoud F. Elhejazi

Table of Contents

COPYRIGHT ……..……II

APPROVAL SHEET ……..……III

DEDICATION …….……...IV

ACKNOWLEDGEMENT ……..……..V

CONTENTS……..…….....VI

CONTENT OF TABLES ……..……...X

CONTENT OF FIGURES ……..……..X

Chapter 1 1

Introduction 1

1.1 INTRODUCTION 1
1.2 MOTIVATION 4
1.3 PROBLEM STATEMENT 4
1.4 RESEARCH AIM AND OBJECTIVES 5
1.5 RESEARCH QUESTIONS 5
1.6 RESEARCH METHOD 5
1.7 SCOPE & LIMITATION 7
1.8 ORGANIZATION OF THE THESIS 7
1.9 CONTRIBUTION 8

Chapter 2 9

Background & Literature Review 9

2.1 INTRODUCTION 9
2.2 SDN ARCHITECTURE TERMS AND CONCEPTS 9
2.3 SDN CURRENT SECURITY STATE 13
2.4 DEFENSE IN-DEPTH PRINCIPLES 15
2.5 LITERATURE REVIEW 16
2.6 ADDRESSING VULNERABILITIES 18
2.7 SUMMARY 19

Chapter 3 20

The Proposed Dynamic Defense-in-Depth (DDiD) 20

3.1 GENERAL DYNAMIC DEFENSE-IN-DEPTH (DDID) MECHANISM 20
3.2 DDID MODEL FOR SDN CONTROLLER LAYER 21
3.3 THREAT VECTORS 21
3.4 COUNTERMEASURES FOR OPENFLOW-BASED SDNS 23
3.5 DETAILED DDID MODEL FOR SDN 24
3.6 EVALUATION MECHANISM 28
3.7 SUMMARY 30

Chapter 4 32

Experimental Work 32

4.1 INTRODUCTION 32
4.2 EXPERIMENTAL PARAMETERS & PREDEFINED CONDITIONS 36
4.3 EXPERIMENTAL IMPLEMENTATION AND RESULTS 37
4.4 DISCUSSION 39
4.5 SUMMARY 41

Chapter 5 42

Conclusion & Future work 42

5.1 CONCLUSION 42
5.2 LIMITATIONS 42
5.3 FUTURE WORK 43

Bibliography 44

Appendix 47

List of figures

Figure 1.1 SDN Architecture (Generic View) 2
Figure 1.2 OpenFlow protocol communication 2
Figure 1.3 DID layers 6

Figure 3.1 Identified threat vectors of SDN architecture 22
Figure 3.2 Proposed DDiD Model Suitable for SDN Architecture in process perspective form. 28
Figure 3.3 Proposed DDiD Model Suitable for SDN Architecture in communication perspective form. 30
Figure 4.1 Experiment Network topology 36

Appx Figure 4.1 Mininet instillation & configuration commands 47
Appx Figure 4.2 Mininet implementation commands 48
Appx Figure 4.3 Mininet start simulation for hosts & switches commands 48
Appx Figure 4.4 OpenDaylight & HPE controllers configuration part 1 49
Appx Figure 4.5 OpenDaylight & HPE controllers configuration part 2 49
Appx Figure 4.6 OpenDaylight & HPE controllers configuration part 3 50
Appx Figure 4.7 OpenDaylight & HPE controllers configuration part 4 50
Appx Figure 4.8 OpenDaylight & HPE controllers connection status 51
Appx Figure 4.9 POX controller setup with Scapy traffic generates commands part 1 51
Appx Figure 4.10 POX controller setup with Scapy traffic generates commands part 2 52
Appx Figure 4.11 Python DoS Attack Script (partial) part 1 52
Appx Figure 4.12 Python DoS Attack Script (partial) part 2 53
Appx Figure 4.13 Entropy tool startup, listening and calculating entropy value for current SDN 54
Appx Figure 4.14 Entropy tool startup, listening, calculating entropy value for DDiD model, and detecting the attacker host. 54

List of tables

Table 3.1 SDN specific and non-specific threats 22
Table 3. 2 Countermeasures for security threats in OpenFlow networks 23
Table 3. 3 Mechanism to threat vectors 29

Table 4. 1 Entropy results for current SDN architecture 38
Table 4. 2 Entropy results from DDiD model 38
Table 4. 3 Standard Deviation from current SDN architecture and DDiD model 39

List of Abbreviations

AAA	Authentication, Authorization, and Accounting
API	Application Programming Interface
API	Application Programming Interface
CPU	Central Processing Unit
DDiD	Dynamic Defense in-Depth
DiD	Defense in-Depth
DOS	Denial of Service
DPI	Deep Packet Inspection
HPE	Hewlett Packard Enterprise
HTTPS	Hypertext Transfer Protocol Secure
ID	Identification
IP	Internet Protocol
MAC	Media Access Control
ML	Machine Learning
NCCIC	National Cybersecurity and Communications Integration Center
ODL	OpenDayLight
POX	Python-based Openflow Controller
REST	Representational State Transfer
SDN	Software-Defined Networking
SSL	Secure Socket Layer
TACACS+	Terminal Access Controller Access Control System Plus
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TPM	Trusted Platform Model
VAN	Virtual Application Networks
VLAN	Virtual Local Area Network

Dynamic Defense In-Depth Model to Enhance SDN Security

By
Mahmoud F. Elhejazi
Supervisor
Dr. Mohamed Musbah

Abstract

The traditional long-established networks and networking techniques are no longer suitable for future ever-expanding networking requirements, specifically, the automation and programmability of network communications. Software-Defined Networking (SDN) is the most agreeable solution for that. SDN intelligence is a logically centralized controller that applies a standard open Application Programming Interface (API) to directly control the packet handling functions of network devices. OpenFlow is currently the main and widely known communication protocol in SDN architecture. As a result of such centralization, the SDN architecture subjected the controller as a single point with more attack surfaces for each layer. This entails the search for more security and protection procedures for the SDN architecture without sacrificing its swift response to changing business requirements. This thesis aims to enhance the protection of the division's concept in SDN architecture, which reduces the creation of more attack surfaces that can be targeted by malicious activities. Thus, the research focuses on the design of a dependable SDN controller model via Defense In-Depth (DID) techniques, including the requirements for a secure, resilient, and robust controller. The Dynamic Defense In-Depth (DDiD) model deployment is proposed for the SDN control layer to enhance overall OpenFlow protocol security. Detailed measurable threats and protection mechanisms, according to the DDiD model, were investigated and implemented using a simulation environment (mininet). Also, the thesis presents a proof of concept evaluation mechanism using entropy for Denial of Service DoS attacks to confirm the applicability of secure structure requirements to the SDN controller layer. The DDiD resulted in a higher standard deviation value between normal traffic and attack traffic than the current SDN architecture, with a diverging value of ±0.02 and utmost ~59.51% difference in a better level of protection .The obtained results confirm the promising potential of achieving the required security goals.

Chapter 1
Introduction

1.1 Introduction

Modern computer networks are structured from a countless number of network devices such as switches, routers, and various types of appliance equipment, which have many complex protocols implemented, for controlling data traffic. The network operators, e.g., network engineers and administrators, are responsible for designing the scheme to deal with a broad range of network events and applications. Though adapting to changing network conditions, they manually convert these high-level schemes into low-level configuration commands. In addition, the network operators have to deal with these complex and tedious tasks with access to very limited tools to achieve accurate adaptation. Networks have become enormously challenging to evolve regarding physical infrastructure, protocols, and performance. The concept of "programmable networks" has been proposed to simplify and address these challenges for better network evolution. In addition, network programmability is part of the broader architecture known as SDN.

SDN is a novel networking architecture that manages and controls packet forwarding, the relaying of packets from one network segment to another by nodes in a computer network called a data plane, within the network to change the limitations of current network infrastructures. In general, the SDN architecture can be broken down into three main layers, application plane, control plane, and data plane. For more clarification, these layers are from bottom to top. The Data plane is the bottom-most layer of SDN architecture. It deals with implementing a data path, which comprises devices, and gets the flow rules and instructions from the upper layer, which will be persisted in the flow table. In some cases, if the received packet does not match any entry in the flow table, the device is responsible for forwarding that packet to the controller for decision-making.

The middle layer, the control plane, is responsible for implementing the control paths on a legacy network. This is the most critical layer of an SDN architecture. It accepts the traffic tasks, traffic engineering, traffic shaping, and network management from application plane servers; and handles it to data plane devices. The highest layer the application layer with the help of a controller, is responsible for the customization of

packet forwarding, policy management, user management, and Quality of Service (QoS). In SDN architecture, all of the network functions and monitoring tools are usually part of the application layer.

There are two interfaces, which are used to help in the communication among the layers, North Bound Interface (NBI) and South Bound Interface (SBI). as shown in Figure 1.1 .

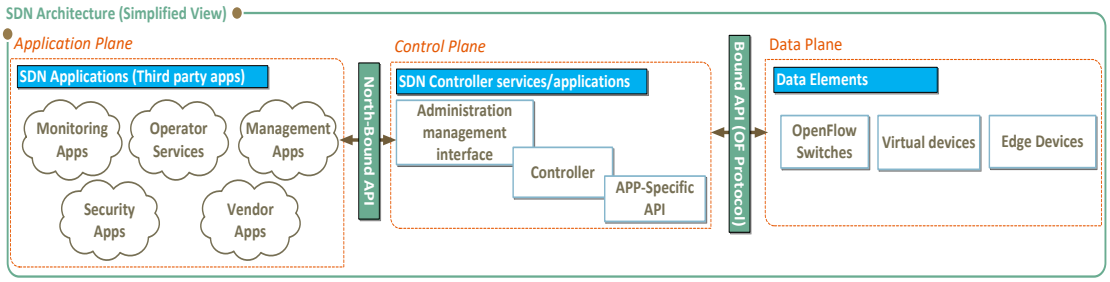


Figure 1.1 SDN Architecture (Generic View)

The concept of SDN is based on the separation of network intelligence, specifically, the control plane is separated from the packet switching process for the data plane into a logically centralized controller. The forwarding decisions are made by the controller, which is responsible for sending the instruction to the packet switches in the form of rules, via standard protocols, such as; OpenFlow protocol.

OpenFlow is a standard communication protocol that enables the control plane to interact with the data plane. Its main purpose is to standardize the communication between switches and the software-based controller SDN. Besides, OpenFlow is still widely used in the implementation of SDN architecture [1], the communication mechanism of OpenFlow protocol is confined to three tables; rule, action, and stats. Figure 1.2 illustrates the communication mechanism of the OpenFlow protocol.

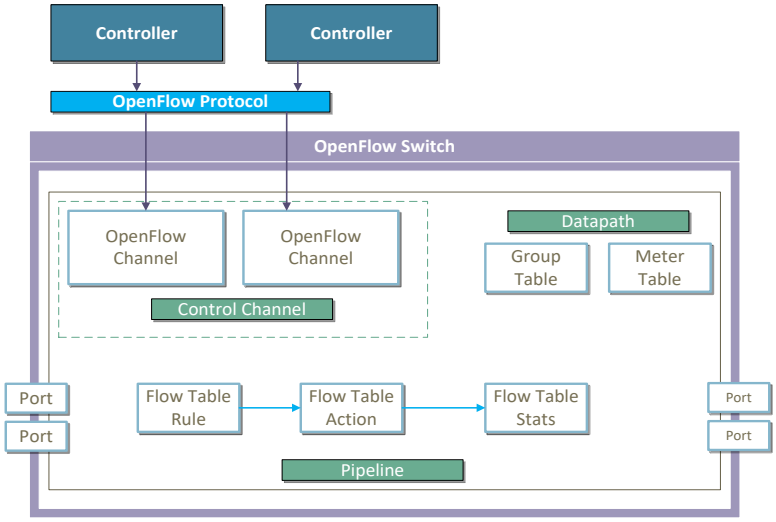


Figure 1.2 OpenFlow protocol communication

Consequently, the purpose of SDN is to enable network operators to answer promptly to the changing business requirements through centralized control support that separates the network's control policies from forwarding tables. Therefore, the separation of the data plane from the control plane in SDN architecture pointed the controller as the most important part of the SDN architecture. As a result, the SDN architecture subjected the controller to a single point of failure [2].

Furthermore, the communication among the three-layer structure and the links in between created more attack surfaces specifically for each layer separately that are not present in traditional network structures, which can be targeted for more malicious activity types. As an example of problem surfaces, the Data plane layer uses the OpenFlow protocol to communicate with low-forcing Transport Layer Security (TLS) v1.0 and most of the communication is using Transmission Control Protocol (TCP) and having TLS as an optional connection. Most of the data plane devices and SDN controllers do not fully support TLS [2],[3],[4],[5].

That being said, this research focuses on finding a better solution by using an approach capable of reducing and mitigating these risks in a divide-and-conquer manner. Inform of multiple layers of protection that are connected altogether for robust and secure SDN design.

One of the most efficient methods, according to Best Practice from National Cybersecurity and Communications Integration Center (NCCIC)[6], is DiD. It is an architecture that uses multiple connected layers for better protection design using TLS v1.3 [3], distribution connection, Access Controls, etc., to the data plane devices. Hypertext Transfer Protocol Secure (HTTPS), Authentication, Authorization, and Accounting (AAA), Deep Packet Inspection (DPI) with fixed-trusted devices ID, etc. to control the plane.

The main aim is to concern the demonstration of DiD mechanism for SDN controllers to build a secure model by clarifying and protecting OpenFlow protocol structure communication functions and preventing the threat vectors which can lead to the exploitation of vulnerabilities of SDN controllers. First, designing a dependable Controller model focuses on the necessity of a secure, flexible, and robust SDN Controller. Secondly, reducing the existing gap between the actual security level of the current SDN Controller design and the potential security solutions for future improvements in the SDN Controller using DDiD. Thirdly, Validating and testing by

simulating the implementation of a secure SDN controller on several systems and devices virtually to ensure its ability to deliver a robust, secure structure.

1.2 Motivation

Several factors have motivated the investigation conducted in this thesis to migrate to the SDN structure successfully. A primary advantage of SDN networking is greater visibility throughout the enterprise networks. The most important obstacle to implementing SDN structure is the operational challenges involved in managing a computer network. There are two essential operational challenges. Time to implement changes: The distributed nature of a network makes it difficult and time-consuming to effect changes in the settings of all network elements. Risk of malfunctioning: These are the security challenges that need addressing to build a secure model that prevents and protects OpenFlow protocol communication functions from threat vectors which can lead to SDN failure.

1.3 Problem Statement

The separation of the data plane from the control plane in SDN architecture pointed to the controller as the most important part of the SDN architecture. As a result, the SDN architecture subjected the controller to a single point of failure [2]. Furthermore, the communication among the three layers of structure and the links in between has created more attack surfaces specifically for each layer separately that were not present in traditional network structures before, which can be targeted for more malicious activity types,especially the bottom layers use the OpenFlow protocol to communicate, which uses low-forcing TLS v1.0 and most of the communication uses TCP and has TLS as an optional connection. Most of the data plane devices and SDN controllers do not fully support TLS. [3],[4],[2],[5].

Therefore, the communication among the three layers is not fully secured and can be easily compromised, by multiple types of attacks and risks that are inclusive for SDN, e.g., falsification of controller rules, data eavesdropping, execute harmful commands, unauthorized access, denial-of-service, exploiting logically centralized controller and deployment of malicious applications.

These attacks and risks are assembled in several threat vectors that have been targeted for mitigation in this research, detailed in Section 2.3.

That being said, the SDN security problem is not a single-tier problem; the SDN security threat involves each layer of the entire system architecture. Therefore, this proposal focuses on finding a better solution by using an approach capable of reducing and mitigating these risks in a divide-and-conquer manner. A model that has multiple layer's protection and is connected for robust secure SDN design.

1.4 Research Aim and Objectives

The aim is to propose a systematic model based on Defence in-depth mechanisms to build a secure model to facilitate the exploitation of vulnerabilities of current SDN architecture. To achieve this aim, this research is focused on the following objectives :

- To conduct an extensive study on the current SDN state in enterprise networks .
- To investigate the mechanisms of the Defense-in-depth concept .
- To propose a dynamic DiD model suitable for the SDN control layer security challenges by utilizing a quantitative method within case study implementation.
- To implement and test the proposed model.
- To compare the results from the proposed model with traditional SDN architecture.

1.5 Research Questions

To achieve these objectives, the research intends to answer the following questions:

- What is the importance of SDN in modern enterprise networks؟
- What are the challenges of SDN controller security؟
- How to mitigate the challenges of SDN controllers for better security ؟
- Why is Defense In-Depth mechanism used as a solution؟
- What is the improvement gained from the proposed DDiD for SDN security؟

1.6 Research Method

The prime methodology that has been used is the quantitative method. Within the case study implementation of the Defense-in-depth mechanism upon SDN architecture, DID is a security principle from NCCIC [6], best practice. Also known as an information assurance strategy that provides multiple, redundant defensive measures in case a security

control fails or a vulnerability is exploited. Based on a structure that is designed to protect the Physical controls: this includes security measures that prevent physical access to IT systems, such as security guards or locked doors, technical controls: security measures that protect network systems or resources using specialized hardware or software, e.g., a firewall appliance.

Finally, Administrative controls: security measures consist of policies or procedures directed at an organization's employees, as shown in Figure 1.3.

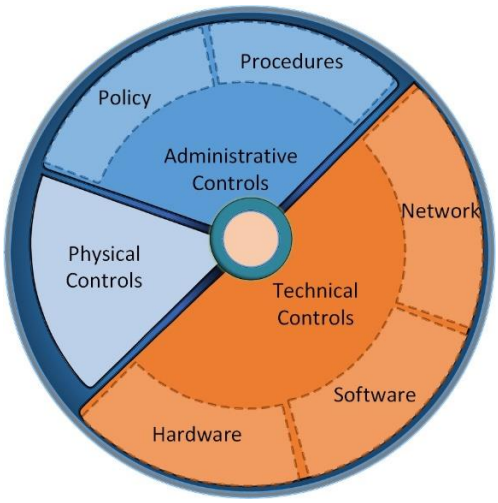


Figure 1.3 DID layers

In the current Controller structure, we are Heading toward methodologies to resolve various threat vectors. In this regard, some of the most efficient mechanisms from several researchers have been presented in this proposal. As a result, different mitigation techniques need to prepare for a suitable design structure. Attempting to lay out a design of a suitable Dynamic defense-in-depth (DDiD) model to address several of these threats. Finally, before applying any design, threat vectors must be identified to evaluate and compare the results obtained from the study.

To summarize the threat vectors [7],[3],[8],[9], that can be mitigated with the use of the Dynamic Defense In-Depth model:

Forged or fabricated traffic flows, targets an exposure in switches, threats on control plane communications, target exposure in controllers, absence of mechanisms to ensure trustworthy between the controller and management applications, targets an exposure in administrative stations, and absence of reliable resources for forensics and restoration. Each of the thread vectors must be mitigated with the proposed DDiD model.

For each validation test case, when comparing the collected data from penetration attack tools with thread vector identifiers to get the positive and the negative results, this presents a strong indication for the research aim and success of reaching a secure SDN controller.

The proposed Dynamic Defense In-Depth model for SND is summarized as follows: states detection connection uses a synchronous distribution for the controllers. Vulnerability avoidance uses independent variety for the controller system. Safety protection uses self-healing for a full system. Relation analysis uses dynamic switch coupling with controllers. Threat avoidance uses reliable relations between controllers & devices. Behavior avoidance uses reliable relations between controllers & apps. Finally, isolation uses virtualized security clusters. Chapter 3 discusses the DDiD mechanism in detail

1.7 Scope & Limitation

This research is mainly focusing on:

- Clarifying the SDN controller concept, the OpenFlow protocol, defense-in-depth architecture, and the implantation of the proposed security enhancement.
- Utilizing DID principles to meet the security requirements of SDN Controller which are; Authentication, Authorization, Facilitation, Isolation, and policies.
- Validating and testing by simulating the implementation of secure SDN on several systems and devices to ensure its ability to deliver robust structure.

However, the requirement of building such a secure structure from selective high-end hardware and software is through a virtual environment to reduce the dependency on specific vendor devices, therefore, the limitations are simulated hardware performance and cost of a virtualization technology license.

1.8 Organization of the Thesis

The remainder of the thesis is organized as follows: Chapter 2 background & related work for related security research papers on SDN security challenges, also the introduction of the Defense-in-depth terminologies. Chapter 3 is the proposed model that is suited for SDN controllers. The findings and focus of the thesis will be in Chapters 3 and 4 to discuss in detail Dynamic Defense-in-depth implementation and results

considering approaches and techniques, and finally Chapter 5 conclusions and future work that concludes this thesis.

1.9 Contribution

Focusing on designing a dependable Controller model including the requirements for a secure, resilient, and robust SDN Controller capable of delivering the expectation of such demands for enterprise's network needs, through deploying DDiD model to deal with the increasing security challenges associated with SDN Controller architecture. As a result of this thesis, the following research paper was published:

1. Mahmoud Elhejazi and Mohamed Musbah. 2021. Dynamic Defense In-Depth Model for SDN Control Layer to Enhance OpenFlow Protocol Security. In The 7th International Conference on Engineering & MIS 2021 (ICEMIS'21). Association for Computing Machinery, New York, NY, USA, Article 46, 1–6. DOI:https://doi.org/10.1145/3492547.3492625

Chapter 2

Background & Literature Review

2.1 Introduction

In the past years, a new architecture paradigm has emerged in the computer network industry called Software Define Network (SDN). In the beginning, it appeared as a potential programming approach that enables dynamic, programmatically efficient network configuration to improve network performance and monitoring. Now SDN is more diverse and abstract for easy to use even for those who are not network engineering. SDN is based on software-based controllers or application programming interfaces (APIs) to communicate with underlying hardware infrastructure and direct traffic on a network. SDN principles in simple terms, is an approach for the provisioning and management of networks.

This chapter will shed some light on multi aspects and terms of SDN architecture and security challenges with more description.

2.2 SDN architecture Terms and Concepts

The terminologies of the research field, such as SDN architecture pillars, layers or planes, connection interfaces, flow protocols, etc., are crucial to the understanding of the topic. This subsection describes the basic terms of SDN.

The term SDN was originally used to describe the ideas and representations that surround the OpenFlow protocol at Stanford University [2]. As originally defined, SDN refers to a network architecture where the forwarding state in the data plane is managed by a remote-control plane decoupled from it. The networking industry has, on many occasions, shifted from this original view of SDN, by referring to anything that involves software as being SDN. Therefore, the attempt in this section is to provide a much less ambiguous definition of SDN. It can be defined as a network architecture with four pillars [2] :

1) The control and data planes are decoupled. Control functionality is removed from network devices that will become simple (packet) forwarding elements.
2) Forwarding decisions are flow-based, instead of destination-based. A flow is broadly defined by a set of packet field values acting as a match (filter) criterion and a set of actions (instructions). In the SDN/OpenFlow context, a flow is a

sequence of packets between a source and a destination. All packets of a flow receive identical service policies at the forwarding devices. The flow abstraction allows the unifying of the behavior of different types of network devices, including routers, switches, firewalls, and middleboxes. Flow programming enables unprecedented flexibility, limited only to the capabilities of the implemented flow tables.

3) Control logic is moved to an external entity, the so-called SDN controller or Network Operating System (NOS). The NOS is a software platform that runs on commodity server technology and provides the essential resources and abstractions to facilitate the programming of forwarding devices based on a logically centralized, abstract network view. Its purpose is therefore similar to that of a traditional operating system.
4) The network is programmable through software applications running on top of the NOS that interact with the underlying data plane devices. This is a fundamental characteristic of SDN, considered its main value proposition.

Note that the logical centralization of the control logic, in particular, offers several additional benefits. [2], First, it is simpler and less error-prone to modify network policies through high-level languages and software components, compared with low-level device-specific configurations. Second, a control program can automatically react to spurious changes in the network state and thus maintain the high-level policies intact. Third, the centralization of the control logic in a controller with a global knowledge of the network state simplifies the development of networking functions, services, and applications.

Following the SDN concept introduced, an SDN can be defined by three fundamental abstractions: (i) forwarding, (ii) distribution, and (iii) specification. Abstractions are essential tools of research in computer science and information technology, being already a ubiquitous feature of many computer architectures and systems. Ideally, the forwarding abstraction should allow any forwarding behavior desired by the network application (the control program) while hiding details of the underlying hardware.

OpenFlow is one realization of such abstraction, which can be seen as the equivalent of a “device driver” in an operating system. The distribution abstraction should shield SDN applications from the vagaries of distributed states by using a common distribution layer, which resides in the NOS. This layer has two essential functions. First,

it is responsible for installing the control commands on the forwarding devices. Second, it collects status information about the forwarding layer (network devices and links), to offer a global network view to network applications. The last abstraction is the specification, which should allow a network application to express the desired network behavior without being responsible for implementing that behavior itself. This can be achieved through virtualization solutions, as well as network programming languages. These approaches map the abstract configurations that the applications express based on a simplified, abstract model of the network, into a physical configuration for the global network view exposed by the SDN controller. This approach has several advantages:

- It becomes easier to program these applications since the abstractions provided by the control platform and/or the network programming languages can be shared.
- All applications can take advantage of the same network information (the global network view) to more consistent and effective policy, decisions while reusing control plane software modules.
- These applications can take actions (reconfigure forwarding devices) from any part of the network. There is therefore no need to devise a precise strategy about the location of the new functionality.
- The integration of different applications becomes more straightforward. For instance, load-balancing and routing applications can be combined sequentially, with load-balancing decisions having precedence over routing policies.

To identify the different elements of an SDN as unequivocally as possible, now present the essential terminology used throughout the SDN.[2].

- Forwarding Devices (FD): Hardware- or software-based data plane devices that perform a set of elementary operations. The forwarding devices have well-defined instruction sets (e.g., flow rules) used to take actions on the incoming packets (e.g., forward to specific ports, drop, forward to the controller, rewrite some header). These instructions are defined by southbound interfaces (e.g., OpenFlow, ForCES (Forwarding Control Elements Separation), Protocol- Oblivious Forwarding (POF)) and are installed in the forwarding devices by the SDN controllers implementing the southbound protocols.

- Data Plane (DP): Forwarding devices are interconnected through wireless radio channels or wired cables. The network infrastructure comprises interconnected forwarding devices, which represent the data plane.
- Southbound Interface (SI): The instruction set of the forwarding devices is defined by the southbound API, which is part of the southbound interface. Furthermore, the SI also defines the communication protocol between forwarding devices and control plane elements. This protocol formalizes the way the control and data plane elements interact.
- Control Plane (CP): Forwarding devices are programmed by control plane elements through well-defined SI instructions. The control plane can therefore be seen as the "network brain". All control logic rests in the applications and controllers, which form the control plane.
- Northbound Interface (NI): The network operating system can offer an API to application developers. This API represents a northbound interface, a common interface for developing applications. Typically, a northbound interface abstracts the low-level instruction sets used by southbound interfaces to program forwarding devices.
- Management Plane (MP): The management plane is the set of applications that leverage the functions offered by the NI to implement network control and operation logic. This includes applications such as routing, firewalls, load balancers, monitoring, and so forth. Essentially, a management application defines the policies, which are ultimately translated to southbound-specific instructions that program the behavior of the forwarding devices. All that elements are shown in Figure 2.1.

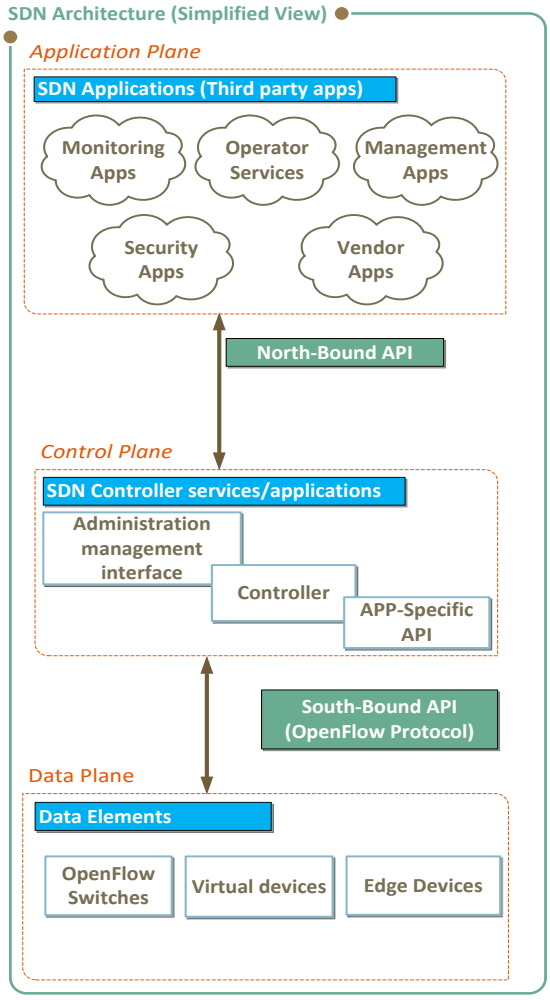


Figure 2.1 SDN architecture elements

2.3 SDN Current Security State

Cyber-attacks against financial institutions, energy facilities, government units, and research institutions that use SDN architecture are becoming one of the top concerns of governments and agencies around the globe. Due to the danger of cyber-attacks and the current landscape of digital threats, security and dependability are top priorities in implementing SDN. While research and experimentation on software-defined networks are being conducted by some commercial players (e.g., Google, Yahoo!, Rackspace, Microsoft), commercial adoption is still in the development stage. Industry experts believe that security and dependability are issues that need addressing with a deep investigation to fully migrate to SDN.[2].

The current state so far is that they are Different threat vectors that have been already identified in SDN architecture, as well as several security issues and weaknesses in

OpenFlow-based networks. The following are at least seven identified threats vector in SDN architecture:

- The first threat vector consists of forged or faked traffic flows in the data plane, which can be used to attack forwarding devices and controllers.
- The second allows an attacker to exploit vulnerabilities of forwarding devices and consequently wreak havoc with the network.
- Threat vectors three, four, and five are the most dangerous ones, which are Exploiting logically centralized controllers, compromised controller and Development, and deployment of malicious applications on controllers since this three can compromise the network operation.
- The sixth threat vector is linked to attacks on and vulnerabilities in administrative stations.
- Last, threat vector number seven represents the lack of trusted resources for forensics and remediation, which can compromise investigations (e.g., forensics analysis).as shown in Figure 2.2

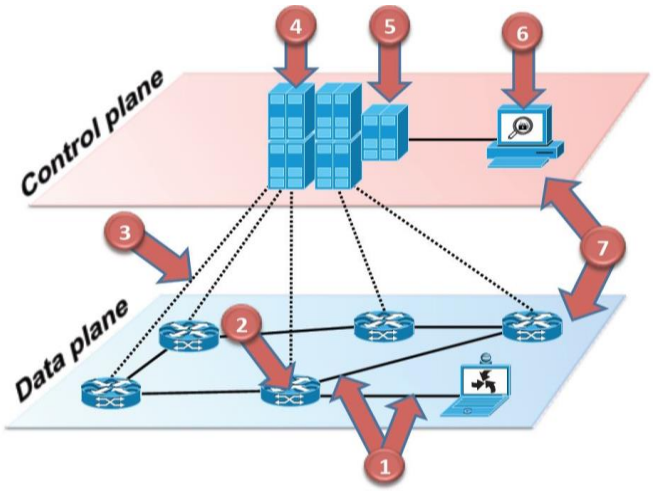


Figure 2.2 Main threat vectors of SDN architecture

Also, there are already several identified security issues in OpenFlow-enabled networks, especially with low-forcing of TLS v1.1. It is possible to identify different attacks on OpenFlow-enabled networks. For instance, information disclosure can be achieved through side-channel attacks like Cache attacks or Power-monitoring attacks targeting the flow rule setup process. When a reactive flow setup is in place, obtaining information about network operation is relatively easy. An attacker that measures the delay experienced by the first packet of a flow and the subsequent can easily infer that

the target network is a reactive SDN; and proceed with a specialized attack [2], This attack known as fingerprinting, may be the first step to launch a DoS attack intended to exhaust the resources of the SDN network. This is only a glimpse of SDN security challenges.

2.4 Defense in-Depth Principles

DID is a technique for information assurance developed by the National Security Agency (NSA) involving several layers of networked electronic and system security defenses. Used as an approach capable of reducing and mitigating security risks in a divide-and-conquer manner, especially for any system, security was not have been taken into account from the planning stage. Also identified as the most efficient mechanism according to the Best Practice form [6], by implementing Different types of network protection software (barriers) are implemented to combine various network security techniques on a single network, including firewalls and Intrusion Prevention Systems (IPS) / Intrusion Detection Systems (IDS) systems. To ensure device and network stability against many threats variables, that affect device protection, including physical security, policy, and procedure. The functions of the barriers in the DiD strategy are detection, deterrence, delay, and response. These functions provide a range of types of barriers to maximize the probability of prevention of unauthorized access and detection potential. These functions can be mapped onto the desired system structure [10]. Constructing a multilayer security DID mechanism must be carefully designed for each specific system separately, from three main layers as shown in Figure 2.3 [11]:

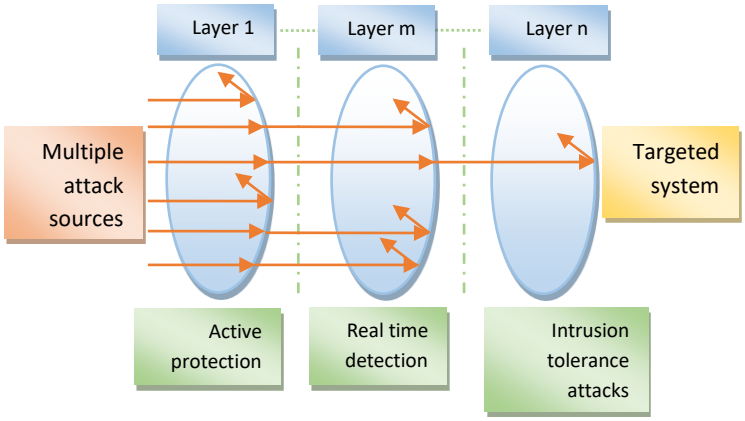


Figure 2.3 Defense-in-depth layers

- Active Protection: Network security products usually impossible to find all the network vulnerabilities and external attacks. Even if the product is designed with comprehensive security defense functions, with the passage of time and the development of protection technology, there is always a chance of reducing this vulnerability with a protective active layer. Examples of such technologies include replication, synchronization, distribution, and backup.
- Real-time detection: Even if the layer of Active Protection defense does not exclude the possibility of being successfully overcome the attacks, a variety of technical means of real-time intrusion detection and prevention is utilized to deal with those attacks, which have not been successfully repulsed. Examples of such technologies include IDS monitoring, and attempts threshold.
- Intrusion tolerance attacks: Intrusion tolerance technology can integrate the immune theory, threshold cryptography, data recovery, and self-healing; It can adopt trusted computing, trusted network, fault-tolerant protocol, data redundancy, and recovery strategy, providing continuous network service and achieve final safety operation of the system.

2.5 Literature Review

The ability of the network operation and security policies to continually adapt to changes in business network services will determine, by how well SDN is implemented and managed. Through a customizable central control, SDN enables network operators to quickly respond to shifting business requirements.

Therefore, research efforts have been devoted to determining the proposed solutions by researchers for such an issue and then developing approaches and models to enhance SDN security early implementation to increase efficiency.

The authors in [3]، stated that most modern SDN-controller, support TLS across D-CPI (Data-Controller Interface). Secure communication with Secure Socket Layer SSL/TLS v1.0 defines the authentication of the communicating parties by using PKI (X.509 certificate) with subsequent data encryption between the parties across the interface of communication to prevent any mitigate tampering with message exchanges. Several controllers such as; SE-Floodlight (Security Enhanced extension, OpenDaylight, and Ryu controller support SSL/TLS v1.1 optionally, although other controllers such as ROSEMARY [12], and ONOS do not support SSL/TLS at all. The author proposed the

following design as secure recommendations features for SDN, Design with Software Security Principles, Secure Default Controller Settings, and Application Future-Proofing. Currently, not a single SDN controller includes each of the identified features for a secure, robust, and resilient SDN controller. However, that design lacks on secure controller design mechanism or framework such as; multiple controllers to application instances (Resilience), representational state transfer (REST) API token-based authentication provided by the user authentication. API key token-based authentication, and Resource Monitoring.

In [4], the author stated that the controller should be provided with additional protection against TCP congestion, IP spoofing, and DoS especially when the SSL/TLS-based communication is unable to protect the SDN Controller from IP-based attacks on the control panel. In such cases, an equivalent protocol such as TLS should be used for protecting the communication between the control layer and the application layer to prevent defects, failure, and threats [4]. However, there must be a key or certificate materials that an appropriately managed with Host Identity Protocol (HIP) and (AAA) to ensure that the security of Public Key Infrastructure (PKI) is underpinned .

According to [13], the main SDN security issue is OpenFlow protocol does not enforce the implementation of the TLS, but defines it only as optional, therefore the author proposed that there is a need for a solution between the control plane layer and data plane layer that include, disabling older versions of TLS protocol, totally uninstalling such version from the controller and use the proposed security extension for TLS v1.2 in form of timestamp between each request to response, which drops connection in case of exceeding the defined time frame per request. Unfortunately, these countermeasures target specific attacks, like Man-in-the-Middle attacks, in which there are more attack points in SDN architecture. Therefore, the recommended data encryption protocol currently is v1.3. It is also the most used protocol not only within OpenFlow but also throughout the Internet.

Nonetheless, according to [5], centralized control and insufficient mechanism of security protection make SDN controllers an external target of malicious attacks. In addition, there are not enough security and encryption measures in the communication process between the control layer and the data layer. Flow rules are easy to suffer malicious tampering during the process of publishing. In general, SDN lacks sufficient multi-level protection mechanisms. Hence, the authors focused their research on the following aspects, Controller attack detection and precaution, Controller scalability and

cross-domain communication, and Application authentication issues. Their conclusion is the SDN security problem is not a single-tier problem; the SDN security threat involves each layer of the entire system architecture. However, an idealized global security solution needs to cover hardware, operating systems, software, and other aspects not just the previously mentioned aspects.

Nonetheless, according to [14], the author conducts a comprehensive survey on the core functionality of SDN from the perspective of secure communication infrastructure at different scales. A specific focus is given to addressing the challenges of securing SDN infrastructure and categorizing the appropriate solutions for specific threats at each layer of the SDN communication. Lastly, security implications and future open research challenges are presented to help gain further insights into the domain of SDN security. However, In this paper, it lacks an appropriate and suitable framework model for a clear implementation of a secure SDN structure for each plan.

Finally, it is important to mention studies in [1],[7],[8],[9],[2], that are essential to this thesis as a major part of building the necessary foundation for implementing the SDN component. that triggers the motivation to mitigate the SDN security challenges building resilient control plane. However, these studies succeeded in describing the vulnerabilities, threats, and risks in the SDN architecture. Also, they lack a systematic framework in form of mechanisms for each plane layer to implement a suitable solution.

2.6 Addressing Vulnerabilities

It is clear from previous papers, that there is a necessity to provide a model for more security improvement on existing SDN controllers and future ones, to become robust, secure, and intelligent. From the DID principles The targeted risks and challenges mitigation are assembled into a seven-threat vector previously mentioned in section 2.3. In this regard, the following are three countermeasures that require improvement for a secure SDN controller:

- **Design based on robust security standards:** SDN controller needs rework, designed from the initiation as a secure software design entity including privilege limitation, sensitive data encryption, and secure defaults in a DDiD mechanism manner. In addition, a controller’s security should be tested using static and dynamic analytic tools.

- **Secure default configuration settings:** All SDN Controller default implementations must apply appropriate safe measurement for operations, configuration backups, and communication processes from the setup stage to ensure that controllers are secure during the entire lifecycle of the system.
- **Application independence:** The application layer must be designed outside the controller's aspects to enhance transferability across the controller interfaces in both high-level and low-level protection.

2.7 Summary

This chapter has given a brief overview of the important concepts of SDN architecture, planes and interconnection interfaces. The concerns and current state of SDN security challenges. In addition, basic fundamental terms and terminologies of DiD. The chapter also introduced the principle of DiD mechanisms, which is a technique for information assurance involving several layers of networked electronic and system security defenses. An insight into DiD's three main layers of barriers, Active Protection: a set of the mechanism that includes replication, synchronization, distribution, backup, and Real-time detection: which includes IDS monitoring and attempts threshold. And Intrusion tolerance attacks include adaptive trusted connection, data recovery, and self-healing .

Also, provided in this chapter is a survey of multi-research papers about the current state of SDN security and how many approaches were taken to address these security challenges.

This chapter also covers the various concern-addressing techniques in terms of communication with SSL/TLS by use of PKI (X.509 certificate), SDN Controller IP-based attacks protection, multi-level protection mechanism, and different timestamp between each request to response per connection. Furthermore, the main threat vectors and challenges will be addressed in the next chapters, and the proposed solution in a systematic framework.

The next chapter provides a deep study of SDN security challenges according to multiple research papers, addressing vulnerabilities, and a brief review will be discussed on each research aspect to mitigate the vectors of SDN security.

Chapter 3
The Proposed Dynamic Defense-in-Depth (DDiD)

3.1 General Dynamic Defense-in-Depth (DDiD) Mechanism

Since the DiD definition is based on protection layering to ensure security, solutions such as firewalls, IDSs and IPSs are also being implemented as part of DiD barries, especally this solutions are combined with modern devices to include the network defense system with various solutions as a new capabilities.

These capabilities must also be implemented to avoid continuous monitoring and probing of such static defenses and to identify internal threats, without causing unnecessary complexity inside the SDN controller system [15].

Therefore, the dynamic defense allows the current-based defenses, such as firewalls, IDSs, IPSs, and malware analysis systems, to become harder to identify, check and bypass. Currently, these based security defenses are easy to penetrate, because they are static and easy to target. When an attacker discovers network protection that defends the intended network or device, the attacker is granted unrestricted time and attempts to check the system's security, eventually finding a vulnerability, by placing a dynamic defense in front of these static, feature-based defenses, the existing defenses become more difficult to find and test and, therefore, provide greater coverage. As the attack surface continuously changes, an attacker is required to spend vast quantities of time and energy to recover the target. Even if the target can be identified by an attacker, they have a finite period of time to check the feature-based defenses behind the dynamic protection before being forced to reacquire the target again.

To advance the current DiD model, a Dynamic DiD model should be focused on the capabilities of symmetry and Proactive defenses. Current network defenses are designed around the features of specific network defense tools, such as identifying malware, blocking packets, or analyzing network events. These defenses are effective against specific attacks, but cannot holistically defend networks. Implementing symmetric defenses, allows for protection against insider threats, both through data exfiltration and network attacks originating within the host network. By deploying proactive defenses, functional network defenses are further enabled by limiting the scope

of possible attacks and the amount of time for attackers to penetrate before reacquiring the target,[15], can summarize the dynamic design to contain two main layers :

- **Symmetric Defenses:** By deploying the same defenses on the internal network as on the external edge, the network can secure itself and other networks. By analyzing internal traffic with symmetric defenses, a Dynamic DiD model can detect and stop data exfiltration. Also, by detecting and stopping attacks at the originating network, symmetric defenses contribute to global network security.
- **Proactive Defenses:** Proactive defenses are enabled through sensing, detecting, orienting, and engaging adversaries in order to assure security, all by dynamic computing performance and dynamic network adaptation, dynamic defenses secure the network by forcing attackers to continually reacquire targets.

3.2 DDiD Model for SDN Controller Layer

Before applying the proposed design of a suitable DDiD model to enhance SDN's current state security, first must identify the threat vectors of SDN architecture.

3.3 Threat vectors

Different threat vectors in SDN architecture as well as several security issues and vulnerabilities in OpenFlow-based networks have already been identified. Although certain threat vectors are similar to current networks, some are more SDN-specific, such as; attacks on the connectivity of the control plane and data plane Table 3.1 [2]. It is necessary to mention that certain threat vectors are independent of technology or protocol specification (e.g., OpenFlow, POF, ForCES), since they present challenges to SDN's functional and architectural layers respectively. [7],[8],[9],[16].

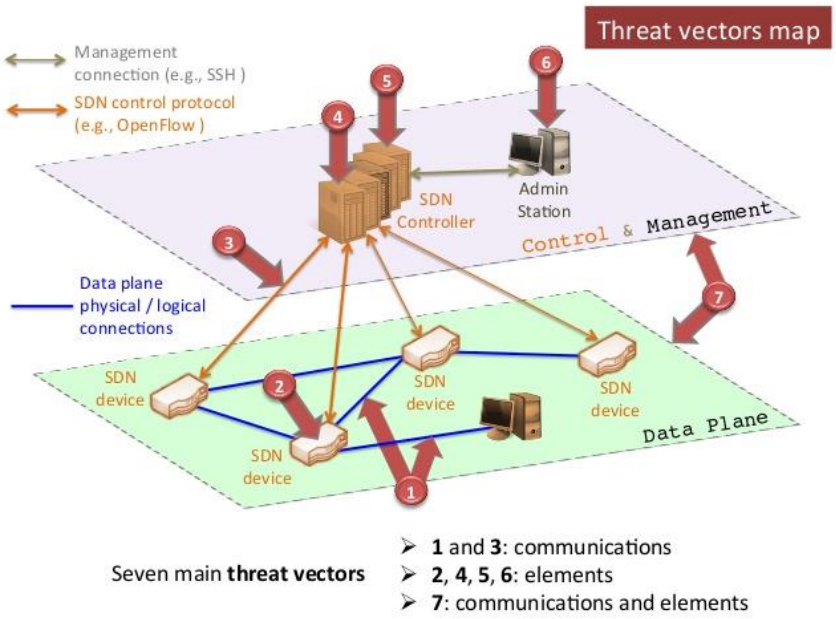


Figure 3.1 Identified threat vectors of SDN architecture

The following table summarizes the threat vectors related to SDN architecture

Table 3.1 SDN specific and non-specific threats

Threat vectors	**Specific to SDN**	**Consequences in SDN**
Vector 1	no	Open door for Distributed Denial-of-Service (DDoS) attacks.
Vector 2	no	Potential attack inflation.
Vector 3	yes	Exploiting logically centralized controller.
Vector 4	yes	Compromised controller may compromise the entire network.
Vector 5	yes	Development and deployment of malicious applications on the controller.
Vector 6	no	Potential attack inflation.
Vector 7	no	Negative impact on fast recovery and fault diagnosis.

Moreover, Figure 3.1 and Table 3.1 summarize the identified threats vector in terms of whether independent or specific to the SDN architecture as mentioned in section 2.3. The lack of isolation, protection, access control, and stronger security recommendations are some of the reasons for these vulnerabilities [2],[8],[9],[16].

Other technical and operational protection issues, in Open-Flow networks, include the absence of clear security guidelines for developers, and the absence of TLS and access control support for most switch and controller implementations. The assumption is that TCP is enough as links are "physically secure," in fact several switches that have listener mode allowed by default (allowing the establishment of malicious TCP connections is highly possible) [2],[7].

3.4 Countermeasures for Openflow-based SDNs

Several countermeasures can be put in place to mitigate the security threats in SDNs. Table 3.2 [2], summarizes several countermeasures that can be applied to different elements of an SDN/OpenFlow-enabled network.

Table 3. 2 Countermeasures for security threats in OpenFlow networks

Measure	**description**
Access control	Provide strong authentication and authorization mechanisms on devices.
Attack detection	Implement techniques for detecting different types of attacks.
Event filtering	Allow (or block) certain types of events to become handled by special devices.
Firewall and IPS	Tools for filtering traffic can help to prevent different types of attacks.
Flow aggregation	Rules to match multiple flows to prevent information disclosure and DoS attacks.
Forensics support	Allow reliable storage of network activity traces to find the root causes of different problems.

Intrusion tolerance	Enable control platforms to maintain correct operation despite intrusions.
Packet dropping	Allow devices to drop packets based on security policy rules or current system load.
Rate limiting	Support rate limit control to avoid DoS attacks on the control plane.
Shorter timeouts	Useful to reduce the impact of an attack that diverts traffic.

Common methods such as access control, mechanisms for attack prevention, event filtering, firewalls, and devices for intrusion detection can be used to minimize the effect of threats or to prevent them. It may be applied in different applications, such as controllers, forwarding systems, middleboxes, etc. Middleboxes, separate intermediary security devices, can be a good option for enforcing security policies in an enterprise. They are more robust and special-purpose devices. Such a strategy also reduces the potential overhead caused by implementing these countermeasures directly on controller or forwarding devices. However, middleboxes can add extra complexity to network management [2],[7].

3.5 Detailed DDiD Model for SDN

The seven vectors show that the potential threat for SDNs is diverse when compared to traditional networks. Therefore, different mitigation techniques must be decided in a suitable design structure. This section lays out the design of a suitable DDiD model to address these threats. The following are the proposed DDiD model mechanisms:

M1. **Synchronous distribution (States Detection):** is the most significant innovation technique for improving the dependability of SDN systems and enhancing network security. Explicitly, a synchronous distribution controller with load balancing is useful for achieving a secure and dependable structure, via a clone duplicate of the controller with at least three instances and a mixed replica approach by replicating the application with multiple controllers to ensure that fault tolerance of both software and hardware,

which happened from accidents or malicious behavior. This mechanism is conducted by connecting three controllers with a revision number to check configurations and rules changes among controllers. In case of failure or a new connection of devices (switch or host). This will mitigate the effect of attack types like DDoS, way to identify the characteristics of such an attack is by using the method of entropy tests[9] .

M2. **Independent variety (vulnerability identification):** is another important mechanism for increasing the robustness and security of SDN, it is necessary to replicate with varied controllers for the avoidance of common mode faults of the same management application, to limit the intersecting vulnerabilities and software bugs. Thus diversifying constraints the combined impact of attacks on common vulnerabilities. In SDN environment, the same management application can run on a different controller. This can be approached by defining common abstracted APIs between applications and multi-controller. This mechanism is conducted by operating the three controllers with their different vendor systems cohesively, within the same administration station. This will mitigate the effect of exploits for each system. So, it will not affect the other two systems, since they are not sharing the same vendor structure.

M3. **Self-healing (safety protection):** reactive and proactive recoveries can restore the system to a healthy state and keep the network virtually functional by replacing the compromised components in the event of persistent threats circumstances. However, for effective self-healing, it is necessary to replace the compromised components with new and diverse versions [3]. Most importantly, diversity must be applied too in the recovery process to strengthen the defense of the system against risks, which target specific vulnerabilities in the SDN structure. This mechanism is conducted by a pre-stored backup, which triggers the threshold when an unauthorized change occurs. This will also mitigate multi-attack types of exposure and exploitation on the whole system, especially after the attack damaged the system, way to identify the characteristics of such an attack is by using the method of safe backup with an Inspection and detection entity.

M4. **Dynamic switch coupling (Relation analysis):** there are situations in which a switch is correlated with only one SDN Controller. In this case, the controller of the switch will not be fault tolerant. Therefore, to avoid such failures, it is necessary that each switch dynamically and securely associates with multiple controllers and that can be

achieved by using a threshold pre-shared cryptography approach for detecting malicious controllers that could prevent attacks such as man-in-the-middle for instance.

M5. **Reliable relation for controller & devices (Threat identification):** setting up a reliable relationship between controller and devices is a critical requirement for increasing the trustworthiness of the overall control plane. Network devices should associate with the controller dynamically without causing unreliable relationships. A typical approach is to trust all controllers and network devices until the trustworthiness of the controller is strongly questionable [3]. Additionally, the controller should be set to report malicious or misbehaving devices, according to failure or deviating detection algorithm. Also, the malicious controller should be automatically isolated when its trustworthiness falls below an unacceptable threshold. This also will mitigate attack types like Packet Sniffing. This mechanism is conducted by upgrading the cryptographic protocol to SSL/TLS v1.3 [9]. and applying the trusted platform module TPM Hardware Protection [17].

M6. **Reliable relation for controller & apps (Behavior identification):** in this situation, a dynamic reliable model should be utilized for a Controller and application software components, which are presenting a changing behavior as a result of attacks or bugs. That is Measured by the "trustor" factor that observes the behavior of the specific quality attributes such as reliability, availability, confidentially, maintainability, safety, and integrity. That " trustor" factor is used by REST API token-based authentication provided by the user authentication type. Therefore, the model can be applied to detect the relationship between the controller and software application and identify malicious behaviors [18]. This will also mitigate attack types like spoofing or brutal force, way to identify the characteristic of such an attack is by using the method of authentication, system patching, and permissions inspection [9].

M7. **Security clusters (isolation):** many kinds of applications use isolated this technique to secure the network from attackers. In SDN GUI, user-level applications must deny access to kernel-level systems using well-defined policies [19]. In this way, the impact of most attacks will not penetrate past the GUI. Therefore, the isolation protects the security of the SDN hardware and drivers. Also, in SDN Controller, a security domain is achieved using mechanisms such as virtualization of sandboxing [3]. With this structure, an active

isolation mode established using a well-defined interface and accessibility allows minimal communication and operation between the isolated virtual domains.

The proposed model just discussed form the core mechanisms of what has been considered in the implementation of a DDiD for SDN. Nevertheless, such designs may benefit from the use of traditional techniques, such as firewalls or IDS/IPS systems, and additional protection tools to specify and compose packet-forwarding policies and to check the connection between plans in real-time. As depicted in Figure 3.2.

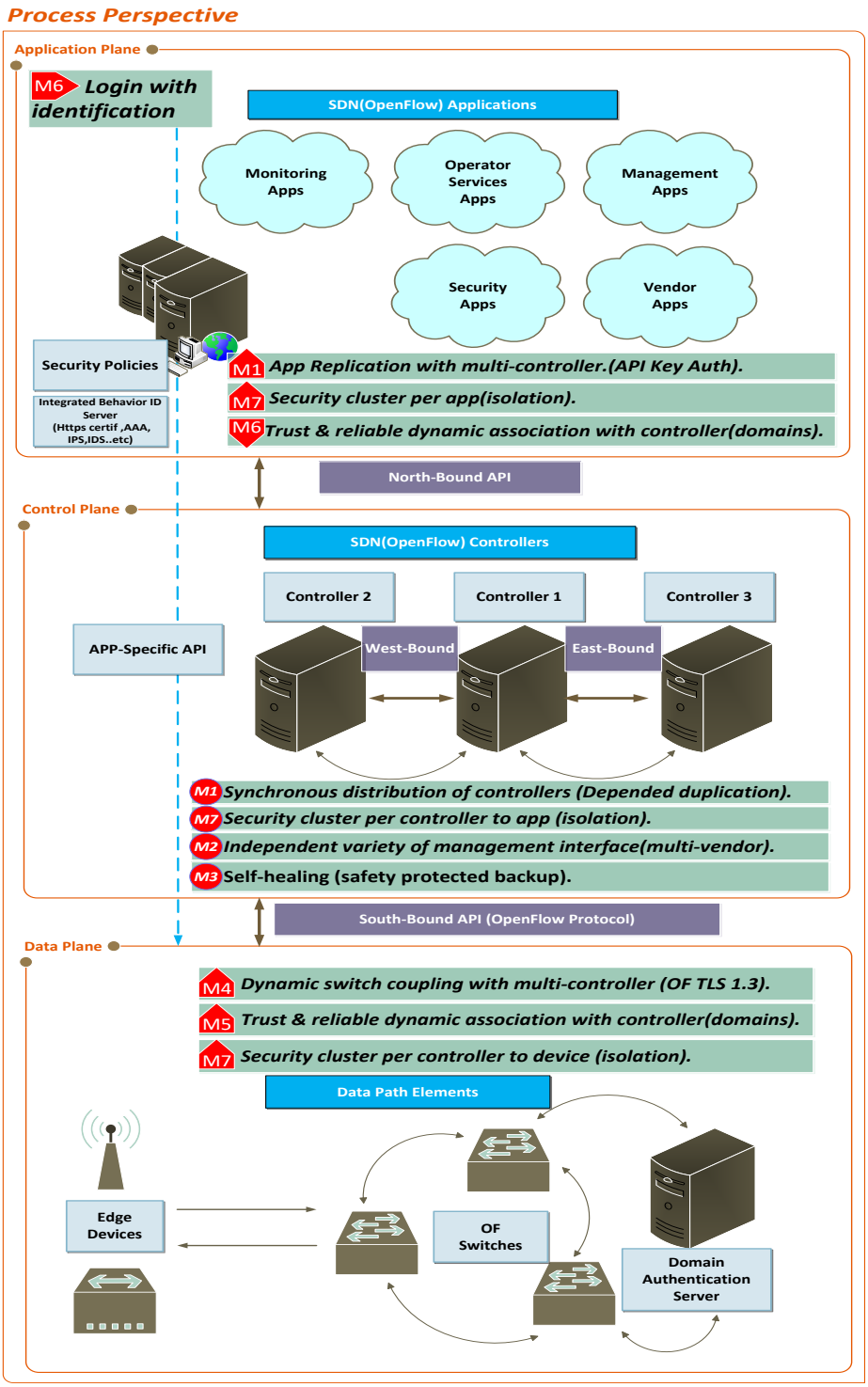


Figure 3.2 Proposed DDiD Model Suitable for SDN Architecture in process perspective form.

3.6 Evaluation Mechanism

A description of the evaluation mechanism is shown in this section for each of the seven vectors. Table 3.3 summarizes the threat vectors and the proposed mechanism accordingly.

Table 3. 3 Mechanism to threat vectors

Threat vectors	**mechanism**
vector 1: *fabrication*	• M1, M5
vector 2: *exposure in switches*	• M3, M5
vector 3: *communications threats*	• M2, M4, M5
vector 4: *exposure in the controller*	• M1, M2, M3, M4, M6, M7
vector 5: *trustworthy*	• M1, M6, M7
vector 6: *exposure in admin stations*	• M2, M3
vector 7: *lack of restoration*	• M1, M3

For each validation test case, when comparing the collected data from penetration attack tools with Table 3.3 of threat vector identifier via a mechanism to obtain accurate results, this presents a strong indication for the research aim and success of reaching a secure model for SDN controller. As depicted in Figure 3.3

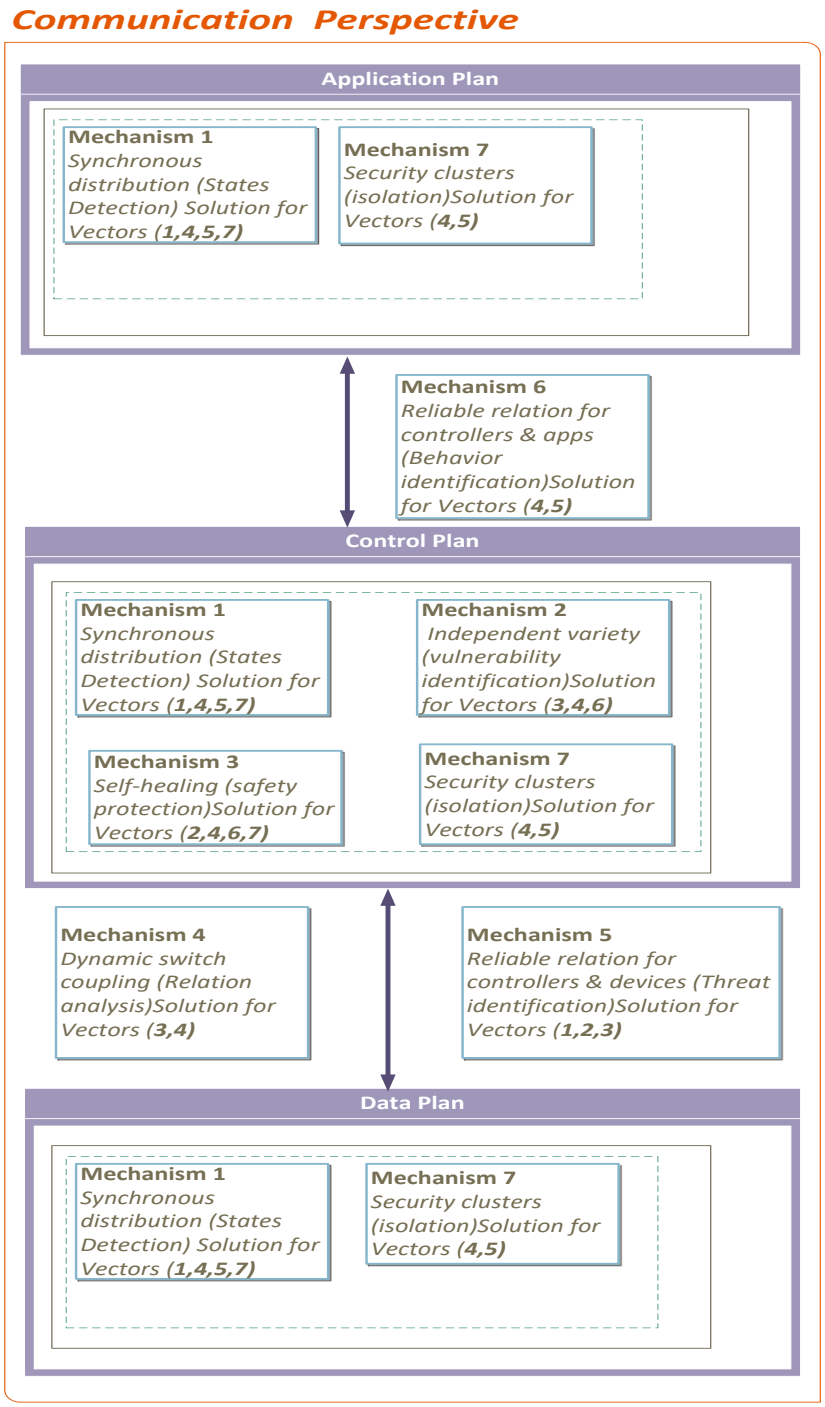


Figure 3.3 Proposed DDiD Model Suitable for SDN Architecture in communication perspective form.

3.7 Summary

In this chapter, we have described a framework to enhance SDN security, by first identifying threat vectors, and the countermeasures for each one of the vectors and then proposed the DDiD model in two perspective processes and processes which are: M1.Synchronous distribution for controllers (States Detection), M2. Independent variety

for controllers (vulnerability identification), M3. Self-healing for a full system (safety protection), M4. Dynamic switch coupling with controllers (Relation analysis), M5. Reliable relation between controllers & devices (Threat identification), M6. Reliable relation between controllers & apps (Behavior identification), M7. Security clusters (isolation). This proposed model maps up every threat with mechanisms of solution. Also, clarify the way how to evaluate and verify the DDiD model.

In the next chapter, the proposed framework is implemented in two aspects one fore current SDN architecture and the other is the proposed DDiD model to evaluate the effectiveness of the security enhancements.

Chapter 4
Experimental Work

4.1 Introduction

In this section, the objectives are to clarify and validate the implementation of the DDiD model to deal with all the SDN security challenges discussed in chapter 3. Therefore, the focus is to apply the designed model, and the penetration test is only as proof of concept, not testing for all types of cyber attacks, in an emulated environment using mininet, which is a network emulator that creates a network of virtual hosts, switches, controllers, and links. Running through standard Linux and its switches support OpenFlow protocol for highly flexible custom SDN. Also, Mininet allows working with various topologies structures, also supporting different controllers like OpenDaylight (ODL) Hewlett Packard Enterprise and Virtual Application Networks (HPE-VAN),etc. [20].

Using also ODL controller, which is a Java, based open-source developed and managed by the Linux Foundation, gives flexibility to a developer to plug-in new applications using northbound APIs, it also supports OpenFlow and other standard protocols from IETF for southbound communication.

In this implementation, the ODL controller has been considered for many reasons:

- ODL tends to standardize the APIs to achieve industry recognition, which turns to prioritize security requirements .
- ODL has a dedicated DoS attack detection and prevention module packaged with recent builds.
- The OpenDaylight community has emphasized more on the security aspects in their recent release. Spoofing of identity is not viable in ODL because the Authentication, Authorization, and Accounting (AAA) service is embedded in the controller and ensures that only authenticated users get access to the resources of the network .
- Also Spoofing or impersonating an identity of a switch could not be achieved in ODL because the Secure Network Bootstrapping Infrastructure (SNBI) module handles the switch authentication process by checking the certificate of an individual switch, which is handled in ODL by using a combination of Media Access Control (MAC), Virtual Local Area Network (VLAN) Identificaion (ID),

Internet Protocol (IP) and Location address as an index in the Device Manager service.

- Both DoS and elevation of privileges threats are addressed in ODL by the Defense4All and AAA modules. The Defense4All module can detect and mitigate different types of network attacks including DoS to its NBI, SBI, processes, and data storage .
- ODL processes can only be accessed by highly privileged (root) users .
- The data flow in the southbound interface for the communication between switches and the controller is secured with TLS protocol, similar protection is achieved in the northbound interfaces for the communication between applications and REST APIs with HTTPS protocol (which uses TLS 1.2 protocol). This secures the channel against threats like tampering and information disclosure.[12].

In addition, using HPE–VAN (Hewlett Packard Enterprise Virtual Application Networks) controller provides a unified control point in an SDN/OpenFlow-enabled network, simplifying management, provisioning, and orchestration. This enables the delivery of a new generation of application-based network services. The HPE VAN SDN Controller has been considered for many reasons:

- Uses OpenStack Keystone to provide token-based authentication to provide user authentication.
- Completely isolates the security mechanism from the underlying REST API .
- Exposes a REST API to allow any authentication server that implements this REST API to host elsewhere (outside the SDN controller).[20].

For penetration test tools that have been used are python scripts via Scapy, which has been used in this experiment to generate packets. Two types of traffic were generated: normal and DOS (Protocol attacks type) attack traffic which has a higher rate than normal traffic. applied by Kali Linux OS which is a Debian-derived Linux distribution designed for digital forensics and penetration testing, maintained and funded by Offensive Security Services [21].

In this implementation, the network topology was built with 3 hosts, 4 switches, and 3 controllers (ODL/HPE-VAN/POX) as depicted in Figure 4.1.a. ODL GUI, is used for the graphical representation of this topology, POX controller is used for running some python attacks and test scripts.

All tools and software used in the experiment for implementing emulated SDN architecture with the Dynamic Defense In-Depth model are described in detail as follows:

- Mininet-2.2.2-170321-ubuntu-14.04.4-server-amd64 [Open vSwitch emulator with POX 1.4 “dart” controller] .
- Hpe-van-sdn-ctlr-2.7.18.0503-ova [SDN controller #1] .
- Ubuntu-18.04.2-live-server-amd64 with OpenDayLight karaf-0.8.4_Fluorine_August_2018 [SDN controller #2].
- DOS python 3.7. 4 script (Scapy 2.4.5 tool library) on Kali Linux OS.
- Middleboxes and mechanisms (AAA RADIUS, IDS, IPS, firewall, PSK, API key Auth, SSL/TLS 1.3, HTTPS cert, L2 filter access and dot1x (802.1x)).
- Hardware[ASUS TUF FX705GE laptop specs: Intel Core i7-8750H Processor (9M Cache, up to 4.10 GHz) / 16GB of RAM type DDR4 with 2666MHz / 1TB HDD + 128GB SSD / NVIDIA GeForce GTX 1050Ti , with 4GB GDDR5 VRAM / 17.3-in (16:9) FHD (1920x1080) 144Hz / Windows 10 pro/ VM WorkStation v12],

For the time and limitation of the Virtual Machine (VM) resources lab and the scope of this research the only type of cyber-attack that needs to be used for testing will be DOS attacks. Cause it is the most common and less skill needed from the attacker. Also, it is the first step to any cyber-attack, to get info about the targeted system before the real attack.

Moreover, see appendix Figures 4.1 to 4.14, which demonstrate how the experiment assembled for the configuration, installation, and setup of network layouts for each component:

1. **mininet:** in this stage of the experiment as shown in (appx Figure from 4.1 to 4.3) is to prepare and install setup commands for the simulated virtual switches and hosts (end-user devices, also the connection parameters of IPs and ports for each controller, and finally the network topology. The Purpose here is to create the data plane infrastructure nods, for the next step; the connection with the control plane.

2. **hpe-van SDN & OpenDayLight SDN:** next as shown in (appx Figures from 4.4 to 4.8) the installation and the startup services commands for the two controllers (HPE-VAN and OpenDayLight) that are both connected to mininet switches in the same network topology emulation, the Purpose here is first to connect the control plane

controllers to the data plane infrastructure nods. Finally, to manage and administrate the controller's setting, rules, and configuration through the GUI control panel.

3. **POX SDN and Scapy traffic generator:** flow up the steps as shown in (appx Figure from 4.9 to 4.10) the installation and startup services commands for the POX controller. Is also connected to mininet switches in the same network topology emulation with other controllers. the purpose here of POX controller is also used for running Scapy python script that generates normal traffic, abnormal attack traffic, and test entropy script.

4. **Python DoS Attack Script:** next as shown in (appx Figure 4.11 and 4.12) the python DOS attack script execution; that has been used inside one of the hosts to begin the attack on the topology controllers and also script will gather the effect result of the attack on the network reachability.

5. **Entropy tool startup and calculating:** Entropy is a detection method used to detect the DDoS attack. It is mainly used to calculate the distribution randomness of some attributes in the network packets' headers. the value is between 0 to 1 or 0 to 2 depending on the number of classes in the dataset used in the experiment, but it means the same thing, a very high level of disorder the closer to 0 it became, which also means in case of DDoS is an attack traffic not normal traffic. flow up as shown in (appx Figure 4.13 and 4.14) the entropy startup service commands and the calculation before and after the DOS attack on the network in both designs the current SDN architecture and the proposed DDiD model. The Normal Traffic is 1.46 for the current SDN architecture, and for the DDiD model, the result was for Normal Traffic 1.54. and for the attack entropy results are explained and shown in detail in the next section the implementation and results section.

6. **Network topology:** was built with 64 hosts, 9 switches, and 3 controllers (ODL/HPE-VAN/POX) as shown in (Figure 4.1).

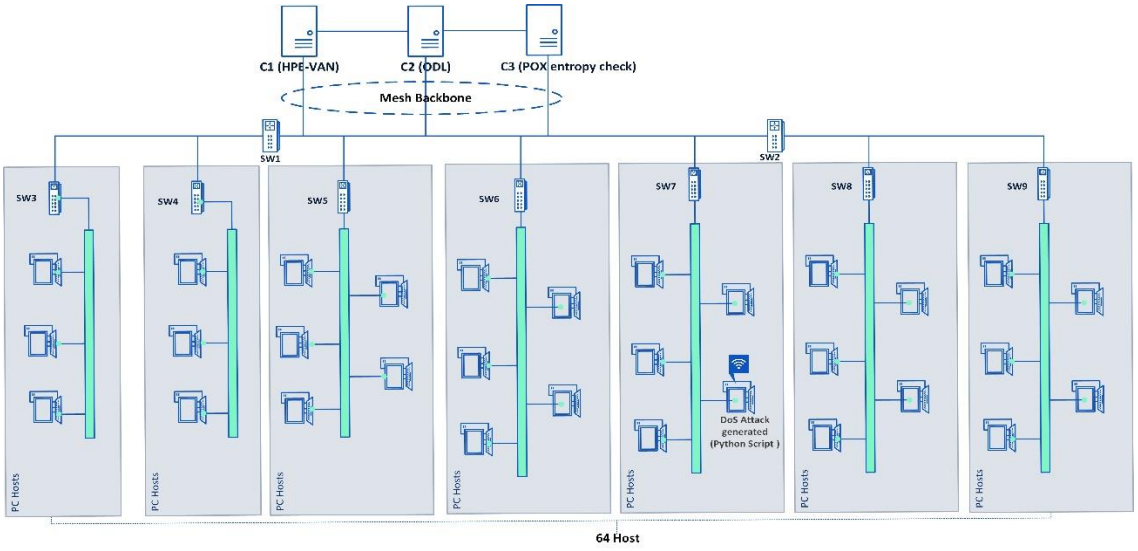


Figure 4.4 Experiment Network topology

4.2 Predefined Conditions & Parameters

Regarding the experiment's predefined conditions All the simulated virtual switches, controllers, and hosts (end-user devices, also the connection parameters of IPs and ports for each controller), must be implemented using the OpenFlow 1.3 protocol stander connection, and hybrid connection type on Linux OS to work properly with fewer glitches.

Regarding, the experiment Parameters utilized to implement the previous mention seven mitigation mechanisms are as follows:

- Applying a Synchronous distribution mechanism (States Detection); by connecting three controllers with a revision number to check configurations and rules changes among controllers. In case of failure or a new connection of devices (switch or host).
- Applying an Independent variety mechanism (vulnerability identification); by operating the three controllers with their different vendor systems cohesively, within the same administration station.
- Applying a Self-healing mechanism (safety protection); by creating a pre-stored backup, which triggers the threshold when an unauthorized change occurs. To start the process of stopping the system and starting a safe mode of recovery, and at the same time, the new requests will be forwarded to other controllers.
- Applying Dynamic switch coupling mechanism (Relation analysis); by using a pre-shared cryptography approach with an access list, to block any unauthorized request for coupling, from switches or controllers.

- Applying Reliable relations for controller & devices mechanism (Threat identification); by upgrading the cryptographic protocol to SSL/TLS v1.3 and applying the trusted platform module TPM Hardware Protection.
- Applying Reliable relations for controller & apps mechanism (Behavior identification); by creating REST API token-based authentication provided by the user authentication type, with TACACS+ (Terminal Access Controller Access Control System Plus) as a AAA server.
- Applying Security clusters mechanism (isolation); by using virtualization as a sandbox isolation approach between controllers with their application plane and data plane, for a cluster of hosts, as separate domains.

4.3 Experimental Implementation and Results

In consequence, for experiment needs, it must first identify the Wight values and Features for DOS attack before and after Appling DDiD model, and the right label to use it as measurement .

- **Features (x) with Weight:** Packet size, Packet arrival interval, Packet number, Packet protocol ID & port, Packet priority, Number of requests, Response time, CPU usage .
- **Labels (y):** consist of Sensitive traffic (QOS prioritized traffic), and Best-effort traffic (Normal /abnormal behavior traffic), which is categorized as Undesired traffic, and hence can be further categorized as Direct attack (DDOS attack). Indirect attack (Man-in-the-Middle attack). Exploit attack (vulnerability attack).

In the proposed model of DDiD, the randomness of the incoming packets must be measured for the right indication of the success of the model. One of the communally used measures of the randomness of DOS attacks is entropy-based or Machine Learning (ML) identification (takes time to learn the features before start identifying them). Entropy measures the probability of an event happening concerning the total number of events using the previous mentions Features in the realm of cybersecurity. [22].

To compute the entropy, it uses the below equations and where x_i is the pool of Features used in proportion to the probability Pi of each connection request total n .

$log_2\{(x1),(x2),(x3),...\}$

$$p_i = \left(\frac{x_i}{n}\right)$$

Entropy $= \sum_1^n p_i(x)\, log_2\left(\frac{1}{p_i(x)}\right)$

When each feature appears only once per request, the entropy will be at it is the maximum value, which means normal traffic. If an attack is occurred by sending a large number of packets directed towards an SDN controller the entropy will be at it is less value (unequal distributed entropy will show) which means attack traffic. Table 4.1 demonstrates the entropy result for the current SDN architecture

Table 4. 1 Entropy results for current SDN architecture

Traffic type	**Average entropy**
Normal Traffic	1.46
Attack Traffic of 10 attempts	1.33

From the two entropy values of normal traffic 1.46 and attack traffic 1.33, the Standard Deviation becomes possible to calculate, for final comparison. Therefore, the Standard Deviation value is σ= 0.03. the following table 4.2 demonstrates the Entropy result for the proposed model

Table 4. 2 Entropy results from DDiD model

Traffic type	**Average entropy**
Normal Traffic	1.50
Attack Traffic of 10 attempts	1.26

Notice the average entropy value is 1.50 that a higher value than normal traffic of current SDN architecture, cause of multi-entity added and structure change, which adds more resource usage, with more demand on performance, which acceptable tradeoff compering to more risk mitigation. and for the attack traffic, the value is 1.26 that a lesser value of entropy than the current SDN architecture, which gives an easier indication for attack detection. Consequently, from the two values of normal and attack traffic, the Standard Deviation value is σ= 0.06.

Table 4.3 demonstrates the comparison between the Standard Deviation from the current SDN architecture and DDiD model, more calculation details in appx 4.15 and appx 4.16

Table 4. 3 Standard Deviation from current SDN architecture and DDiD model

	Standard Deviation
Current SDN architecture	σ: 0.03
DDiD model	σ: 0.06
Diverge	± 0.02
Percentage Difference	~59.51% difference

Hence, from the result value of Standard Deviation in both cases, the DDiD got a higher Deviation between normal traffic and attack traffic than the current SDN architecture. With the diverging value of ±0.02 and utmost ~59.51% difference in the level of protection. To put it in perspective, a high standard deviation value in each separated case means that the traffic is outspread far from the average value of normal traffic and that indicates abnormal traffic was detected. In our case that was the event of a DDOS attack (to explain the DDOS attack impact).

In general, the attack produced massive and continuous data packets, so the standard deviation of flow packets will be higher than the normal traffic flow. Which why as previously mentioned, the standard deviation is commonly used in conjunction with entropy to detect DDOS attacks. The DDiD model with the help of entropy and higher standard deviation value managed to detect and drop the packets from the device that was the source of DDOS attack by noticing the massive number of continuous requests per period and mitigating the attack as shown Appx Figure 4.14. Furthermore, the values of the results may vary. This is because the experiment has been conducted in an emulated environment from top to bottom perspective for the software and the hardware.

4.4 Discussion

This paper aimed to enhance the protection of the divided concept in SDN architecture, which reduces the creation of more attack surfaces that can be targeted by

malicious activities. Also, describing the evaluation mechanism, which confirms if the SDN controller layer met the requirement of a Secure structure. Consequently, this research was carried out on the design of a dependable controller model with the requirements for a secure, resilient, and robust SDN controller, by reducing the existing gap between the actual security level of the current SDN Controller design and the potential security solutions, through deploying a DDiD mechanism in Openflow protocol for SDN Controller .

In general, DDiD has proven a significant aspect in enhancing the security of SDN architecture. The key factor lies in the fact that DID mechanism is more flexible, agile, reliable, and robust to mitigate SDN security challenges. In addition, a secured SDN is an essential requirement to get implemented in an operational network for control and administration purposes. Furthermore, that kind of architecture will play a further role in the ongoing growth of internet services demands.

DDiD mechanism suitable for SDN is still in the enhancement phase and can go forward with more future work, like more efficient distributed layers of protection for each instance of the SDN component without affecting much in the performance and hardware resources. This paper has highlighted and outlined the Great potential of SDN from one side and proposes a solution to the dark side of SDN which is the security challenges of using DDiD. Therefore, from This Experiment the mainly learned points are:

1. Clarifies the SDN controller's concept, the OpenFlow protocol, and network infrastructure emulation to implant the SDN architecture.
2. Meeting the main requirements of SDN architecture which is; the application layer, control layer (control plane), and infrastructure layer (data plane).
3. Validating and testing by simulating the implementation of SDN on several controllers and device emulators to ensure its ability to deliver robust results.
4. Applying DDiD Model for SDN Control Layer to Enhance OpenFlow Protocol Security and then testing it by Launching a DOS attack led to noticeable results that ensure the enhancement of SDN security.

4.5 Summary

This chapter, meeting up the objectives of clarifying and validating the implementation of DDiD model to deal with the SDN security vectors challenges in an emulated environment. As well in this chapter, the effectiveness of the proposed framework has been evaluated in two experimental studies, each one has ten different variables of DoS attack attempts; each attempt has its weight values and Features for both structures, the current SDN architecture and after Appling the DDiD model to validate the correctness of the framework mechanisms, which are used to form the DDiD model.

Furthermore, the results followed during the experiments were carefully measured. By maintaining the integrity of each entropy phase value that has been calculated with as many samples as possible, and keep checked by stander deviation.

The next chapter discusses the conclusion in detail and concludes the thesis.

Chapter 5
Conclusion & Future work

5.1 Conclusion

SDN allows network operators to quickly respond to changing business requirements by determining traffic from the centralized controller without interacting with the physical endpoint devices. Therefore, more security and network infrastructure protections are needed.

An extensive investigation of the mechanism of DDiD model was conducted as a proposed security framework for the OpenFlow protocol for SDN Controller. In general, DDiD has proven a significant aspect in enhancing the security of SDN architecture. The key factors lie in the fact that DID mechanism is more flexible, agile, reliable, and robust to mitigate SDN security challenges.

DiD employ different types of network protection software (barriers) that combine various network security techniques on a single network. The DDiD mechanisms are synchronous distribution, independent variety, self-healing, dynamic switch coupling, trust relation for controller and devices, trust relation for controller & apps, and security clusters against the vectors: fabrication, exposure in switches, communications threats, exposure in the controller, trustworthy, exposure in admin stations and lack of restoration.

The testing method used in this research is entropy-based detection, to measure the randomness of DOS attacks. From the result value of standard deviation in both entropy-based cases, the DDiD resulted in a higher standard deviation value between normal traffic and attack traffic than the current SDN architecture. With a diverging value of ±0.02 and utmost ~59.51% difference in the level of protection .

5.2 Limitations

The requirement of building such a secure structure from selective high-end hardware and software is through a virtual environment to reduce the dependency on a specific vendor device, therefore, the scalability of such an experiment that has a large data between traffic and rules is limited to simulated hardware performance, controller

open-source software glitches patching, resource utilization, application packaging deployment compatibility, and cost of a virtualization technology license.

5.3 Future Work

Even though the proposed model showed better security mitigation, more work on enhancing such a model is accomplished. For instance, more efficient distributed layers of protection for each component of the SDN. a zero-trust model should be applied to verify explicitly for each connection with data plane devices or application plane, least privileged access for each administrator user, and finally assume breach before real compromise happened, all that without affecting much in the performance and hardware resources.

Bibliography

[1] A. Lara, A. Kolasani, and B. Ramamurthy, "Network Innovation using OpenFlow: A Survey," *IEEE*, pp. 1–20, 2013.

[2] D. Kreutz *et al.*, "Software-Defined Networking : A Comprehensive Survey," *Manuscr. Accept. Publ. Proc. IEEE. Novemb. 10, 2014.*, vol. 103, no. 1, pp. 1–63, 2014.

[3] S. Scott-Hayward, "Design and deployment of secure, robust, and resilient SDN controllers," *2015 IEEE*, pp. 1–5, 2015.

[4] M. Liyanage, M. Ylianttila, and A. Gurtov, "Securing the control channel of software-defined mobile networks," *Proceeding IEEE Int. Symp. a World Wireless, Mob. Multimed. Networks 2014, WoWMoM 2014*, 2014.

[5] Y. Liu, B. Zhao, P. Zhao, P. Fan, and H. Liu, "A survey: Typical security issues of software-defined networking," *China Commun.*, vol. 16, no. 7, pp. 13–31, 2019.

[6] Homeland Security, "Recommended Practice: Improving Industrial Control Systems Cybersecurity with Defense-In-Depth Strategies," *Ics-Cert*, no. September, pp. 1–48, 2016.

[7] D. Kreutz, F. M. V. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," *HotSDN 2013 - Proc. 2013 ACM SIGCOMM Work. Hot Top. Softw. Defin. Netw.*, pp. 55–60, 2013.

[8] P. Krishnan and J. S. Najeem, "A review of security threats and mitigation solutions for SDN stack," *Int. J. Pure Appl. Math.*, vol. 115, no. 8 Special, pp. 93–99, 2017.

[9] A. Pradhan and R. Mathew, "Solutions to Vulnerabilities and Threats in Software Defined Networking (SDN)," *Procedia Comput. Sci.*, vol. 171, no. 2019, pp. 2581–2589, 2020.

[10] C. L. Smith, "Understanding concepts in the defence in depth strategy," *IEEE Annu. Int. Carnahan Conf. Secur. Technol. Proc.*, pp. 8–16, 2003.

[11] S. Liu, P. Zhang, and H. Sun, "Research on defense in-depth model of information network confrontation," *Proc. - 4th Int. Conf. Comput. Inf. Sci. ICCIS 2012*, pp. 267–270, 2012.

[12] R. K. Arbettu, R. Khondoker, K. Bayarou, and F. Weber, “Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers,” *2016 17th Int. Telecommun. Netw. Strateg. Plan. Symp. Networks 2016 - Conf. Proc.*, pp. 37–44, 2016.

[13] B. Agborubere and E. Sanchez-Velazquez, “OpenFlow communications and TLS security in software-defined networks,” *Proc. - 2017 IEEE Int. Conf. Internet Things, IEEE Green Comput. Commun. IEEE Cyber, Phys. Soc. Comput. IEEE Smart Data, iThings-GreenCom-CPSCom-SmartData 2017*, vol. 2018-Janua, no. February 2018, pp. 560–566, 2018.

[14] M. Rahouti, K. Xiong, Y. Xin, S. K. Jagatheesaperumal, M. Ayyash, and M. Shaheed, “SDN Security Review: Threat Taxonomy, Implications, and Open Challenges,” *IEEE Access*, vol. 10, pp. 45820–45854, 2022.

[15] S. Groat, J. Tront, and R. Marchany, “Advancing the defense in depth model,” *Proc. - 2012 7th Int. Conf. Syst. Syst. Eng. SoSE 2012*, pp. 285–290, 2012.

[16] R. Pradeepa and M. Pushpalatha, “Exploring attack vectors and security challenges in SDN,” *Int. J. Innov. Technol. Explor. Eng.*, vol. 8, no. 11, pp. 2945–2949, 2019.

[17] TCG, “Trusted Platform Module Library Part 3: Commands,” 2014.

[18] Z. Yan and C. Prehofer, “Autonomic trust management for a component-based software system,” *IEEE Trans. Dependable Secur. Comput.*, vol. 8, no. 6, pp. 810–823, 2011.

[19] K. K. Karmakar, V. Varadharajan, U. Tupakula, and M. Hitchens, “Towards a Dynamic Policy Enhanced Integrated Security Architecture for SDN Infrastructure,” *Proc. IEEE/IFIP Netw. Oper. Manag. Symp. 2020 Manag. Age Softwarization Artif. Intell. NOMS 2020*, 2020.

[20] H. Packard Enterprise, “HPE VAN SDN Controller 2.7 Administrator Guide,” no. March 2016, 2016, [Online]. Available: http://h20564.www2.hpe.com/hpsc/doc/public/display?docId=c05028095.

[21] M. A. Raphaël Hertzog, Jim O’Gorman, *Kali Linux Revealed Mastering the Penetration Testing Distribution*. 2017.

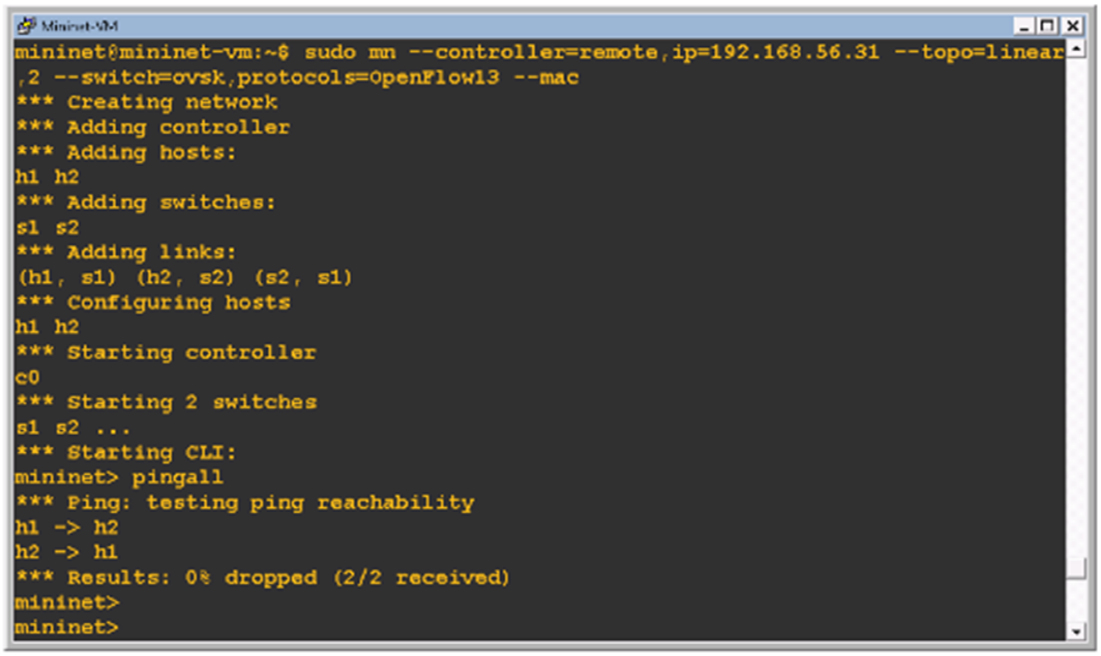
[22] R. M. A. Ujjan, Z. Pervez, K. Dahal, W. A. Khan, A. M. Khattak, and B. Hayat, “Entropy based features distribution for anti-ddos model in SDN,” *Sustain.*, vol. 13, no. 3, pp. 1–27, 2021.

Appendix

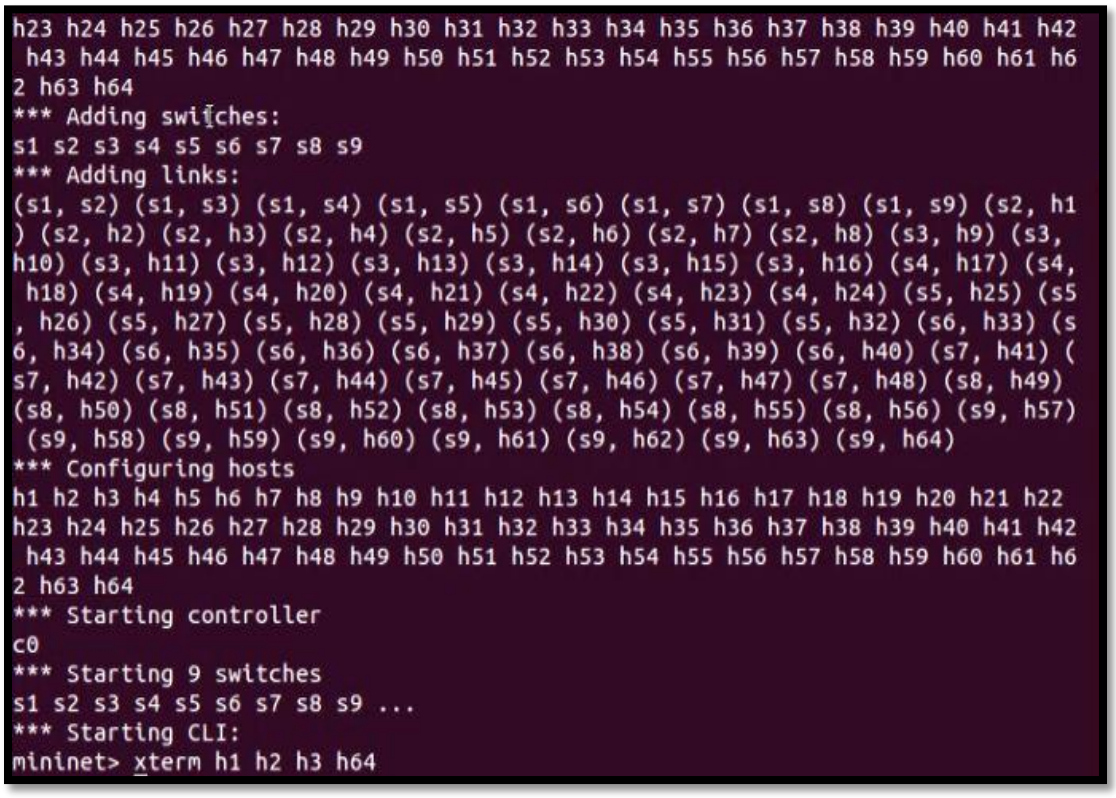
This section contains the experiment figures of configuration, calculation formulas, and setups for installing the labs.

```
1.  mininet@mininet-vm:~$sudo netplan apply
2.  mininet@mininet-vm:~$sudo mn -c //clean any Temporary files
3.  mininet@mininet-vm:~$sudo mn
4.  or
5.  mininet@mininet-vm:~$sudo mn --controller=remote,ip=192.168.1.50 //define the controller ip
6.  mininet@mininet-vm:~$sudo mn --controller=remote,ip=192.168.1.50 --switch=ovsk,protocols=
OpenFlow13 //decide openflow version
7.  mininet@mininet-vm:~$sudo mn --controller=remote,ip=192.168.1.50 --switch=ovsk,protocols=
OpenFlow13 --topo=single,5 //one switch & 5 hosts
8.  mininet@mininet-vm:~$sudo mn --controller=remote,ip=192.168.1.50 --switch=ovsk,protocols=
OpenFlow13 --topo=linear,3 //3 switches & 1 host each
9.  mininet@mininet-vm:~$sudo mn --controller=remote,ip=192.168.1.50 --switch=ovsk,protocols=
OpenFlow13 --topo=tree,fanout=3,depth=3 //depth=level of tree     main switch[L1] & 3 switch
submain[L2] & 9 switches submain[L3]] & 3 hosts each =27 device
10. mininet@mininet-vm:~$sudo mn --controller=remote,ip=192.168.1.50 --ipbase=20.0.0.0/8
//change the topology subnet
11. mininet@mininet-vm:~$sudo mn --mac --controller=remote,ip=192.168.1.50 --ipbase=20.0.0.0/8
//change the topology subnet //and organize the mac address
12. mininet@mininet-vm:~$sudo mn --mac --controller=remote,ip=192.168.1.50 --link=tc,bw=10,delay
=10ms //determining the bandwidth and the dealy
13. mininet@mininet-vm:~$ls
14. mininet@mininet-vm:~$cd mininet/custom
15. mininet@mininet-vm:~/mininet/custom$sudo nano topo-2sw-2host.py //to change the default
mininet topology python file
16. //mytopo=class name inside .py file
17. mininet@mininet-vm:~$sudo mn --controller=remote,ip=192.168.1.50 --switch=ovsk,protocols=
OpenFlow13 --custom=topo-2sw-2host.py --topo=mytopo
18. mininet>pingall
19. mininet>nodes    //show nods names
20. mininet>network or net  //show nods network
21. mininet>dump     //show ip's
22. mininet>h1 ping h2
23. mininet>h1 ping -c 5 h2
24. mininet>h1 arp
25. mininet>h1 route
26. mininet>link s1 h1 down
27. mininet>link s1 h1 up
28. mininet>iperf //tool for network performance measurement
29. mininet>sudo ovs-vsctl show //to show connection status between switches and controller
30. mininet>sudo ovs-ofctl dump-flows s1 //to show flows table of open-switch
```

Appx Figure 4.1 Mininet instillation & configuration commands



Appx Figure 4.2 Mininet implementation commands



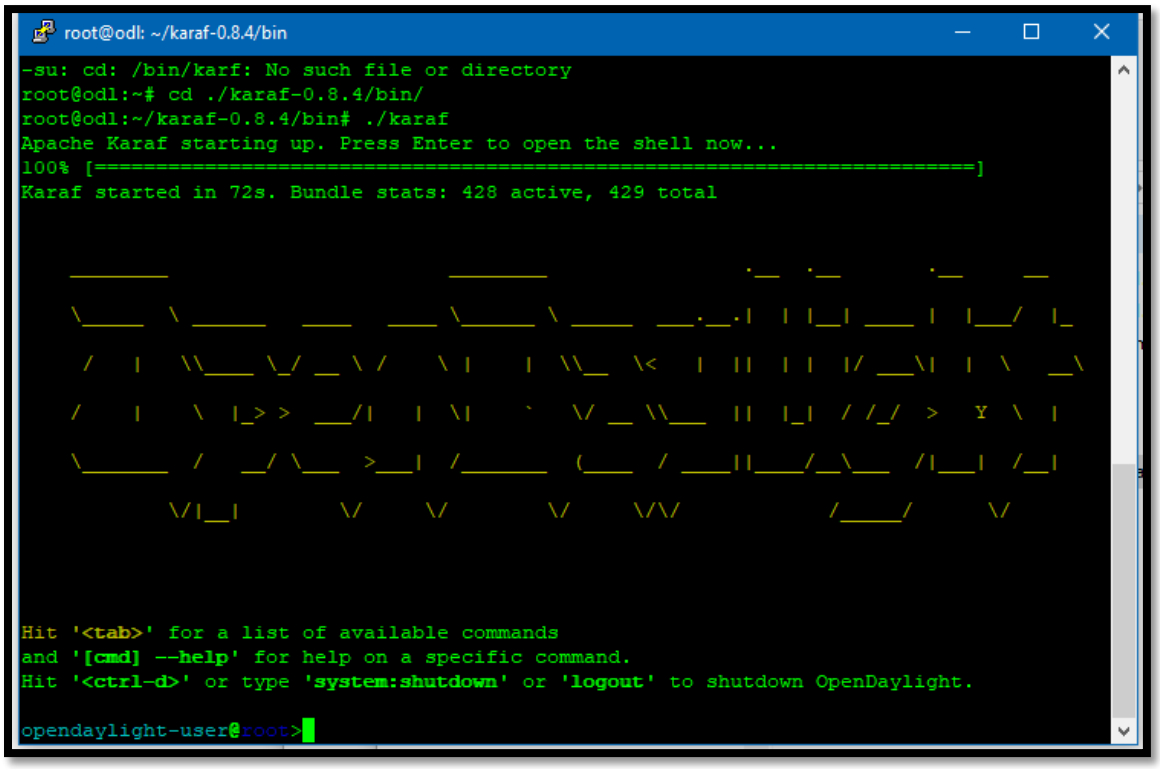
Appx Figure 4.3 Mininet start simulation for hosts & switches commands

```
1.  https:\\192.168.1.50:8443

2.  sudo service sdnc status //to start https serveries
3.  sudo service sdnc stope //to stop https serveries
//////////////////////////////////////////////////////
4.  apt-get install maven git openjdk-8-jre openjdk-8-jdk unzip
5.  wget https:
//nexus.opendaylight.org/content/repositories/opendaylight.release/org/opendaylight/integration/k
araf/0.8.4/karaf-0.8.4.zip
6.  ls
7.  unzip karaf-0.8.4.zip
8.  cd karaf-0.8.4
9.  ls
10. cd bin
11. export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
12. //permanently save
13. nano ~/.bashrc
14. //export JAVA_HOME="/usr/lib/jvm/java-8-oracle"
15. export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
16. . ~/.bashrc

17. ./karaf //to start opendaylight
```

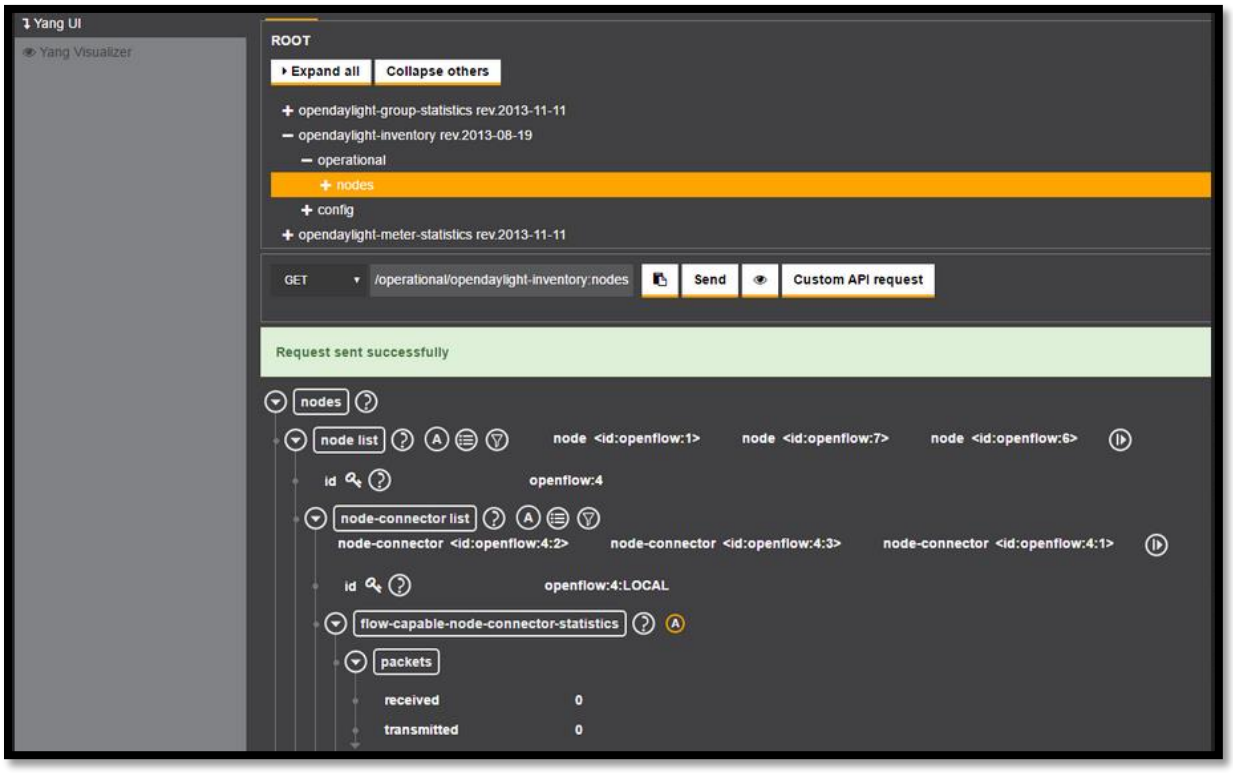
Appx Figure 4.4 OpenDaylight & HPE controllers configuration part 1

```
22. //to startup the web GUI
23. feature:install odl-dlux-core
24. feature:install odl-dluxapps-topology
25. feature:install odl-dluxapps-applications
26. feature:install odl-dluxapps-nodes
27. feature:install odl-restconf
28. feature:install odl-l2switch-switch
29. feature:install odl-mdsal-apidocs
30. logout
31. //extra
32. feature:install odl-restconf-all
33. feature:install odl-openflowplugin-all
34. feature:install odl-l2switch-all
35. feature:install odl-snmp-plugin
36. feature:install odl-restconf

37. //webpage
38. http://192.168.1.51:8181/index.html
```

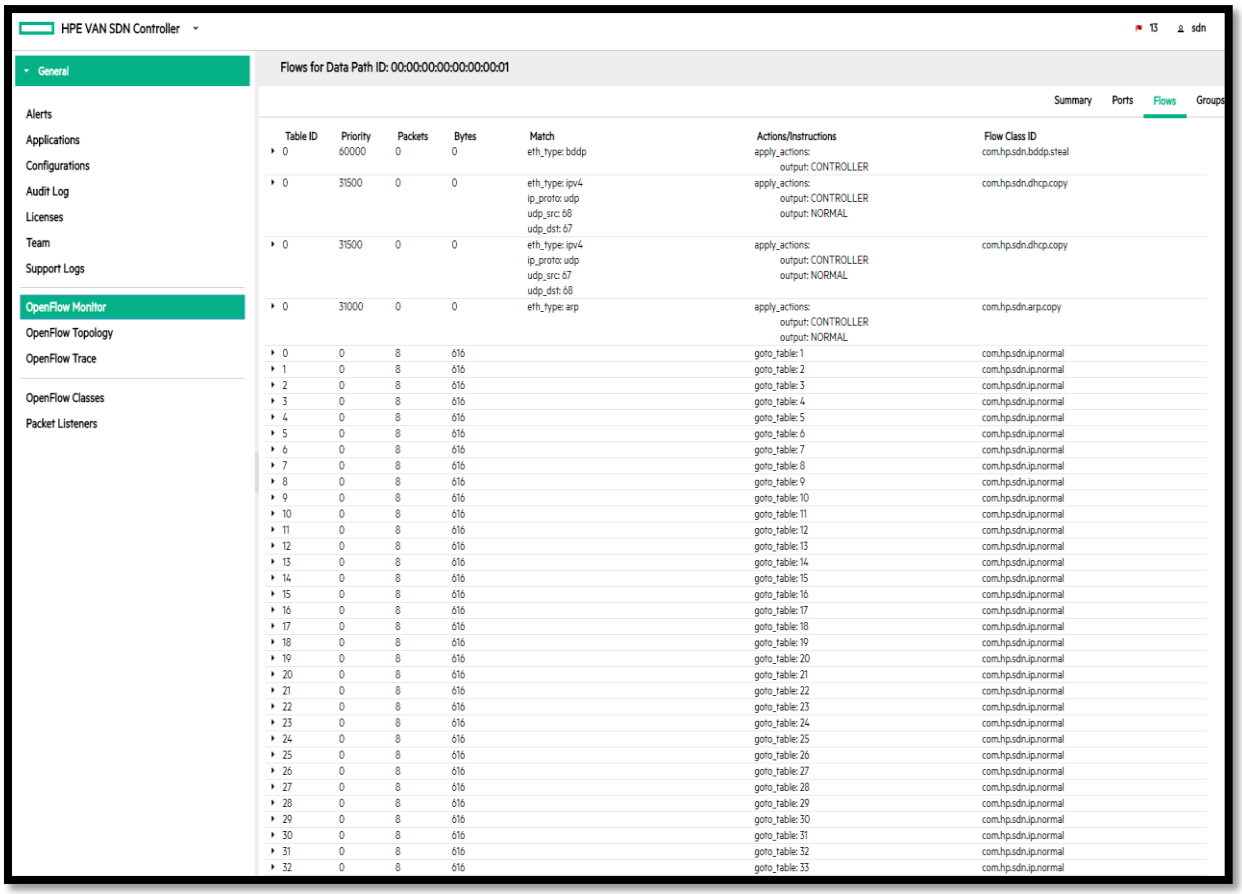
Appx Figure 4.5 OpenDaylight & HPE controllers configuration part 2



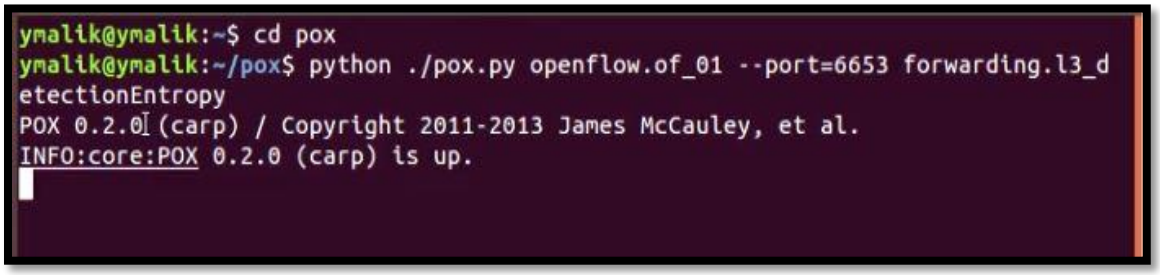
Appx Figure 4.6 OpenDaylight & HPE controllers configuration part 3



Appx Figure 4.7 OpenDaylight & HPE controllers configuration part 4



Appx Figure 4.8 OpenDaylight & HPE controllers connection status



Appx Figure 4.9 POX controller setup with Scapy traffic generates commands part 1

```
root@ymalik:~/mininet/custom# python traffic.py -s 2 -e 65
<Ether  type=0x800 |<IP  frag=0 proto=udp src=160.90.110.162 dst=10.0.0.15 |<UDP
  sport=2 dport=80 |>>>

.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=58.176.80.146 dst=10.0.0.10 |<UDP
 sport=2 dport=80 |>>>

.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=166.72.220.61 dst=10.0.0.48 |<UDP
 sport=2 dport=80 |>>>

.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=63.192.135.72 dst=10.0.0.10 |<UDP
 sport=2 dport=80 |>>>

.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=72.166.103.115 dst=10.0.0.34 |<UDP
  sport=2 dport=80 |>>>

.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=147.175.242.241 dst=10.0.0.32 |<UD
P  sport=2 dport=80 |>>>
```

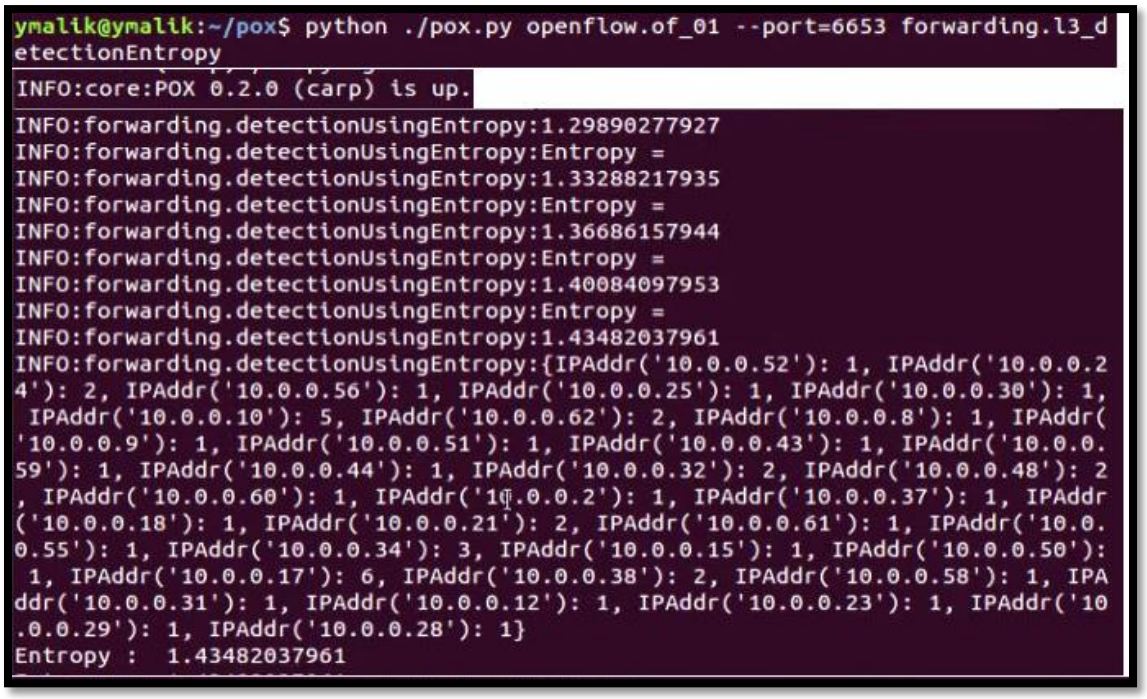
Appx Figure 4.10 POX controller setup with Scapy traffic generates commands part 2

```
# Basic user interface header
print(r"""
 ____   ___   ____          _  _                 _
|  _ \ / _ \ / ___|    /\  | || |         | |
| | | | | | | |___     /  \ | || |_ _ __  __| | __
| | | | | | |\___ \   / /\ \|__   _/ _` |/ _| |/ /
| |_| | |_| |___) | / ____ \  | || (_| | (_|   <
|____/ \___/|____/ /_/    \_\_|\__\__,_|\___|_|\_\ """)
print("\n*****************************************************************")
print("\n****** Work by of M.F.H 2021     ******************************")
print("\n*****************************************************************")


# If the user doesn't run the program with super user privileges, don't allow them to
continue.
if not 'SUDO_UID' in os.environ.keys():
print("Try running this program with sudo.")
exit()

# Remove .csv files before running the script.
for file_name in os.listdir():
# We should only have one csv file as we delete them from the folder
#  every time we run the program.
if ".csv" in file_name:
print("There shouldn't be any .csv files in your directory. We found .csv files in your
directory and will move them to the backup directory.")
# We get the current working directory.
directory = os.getcwd()
try:
# We make a new directory called /backup
os.mkdir(directory + "/backup/")
```

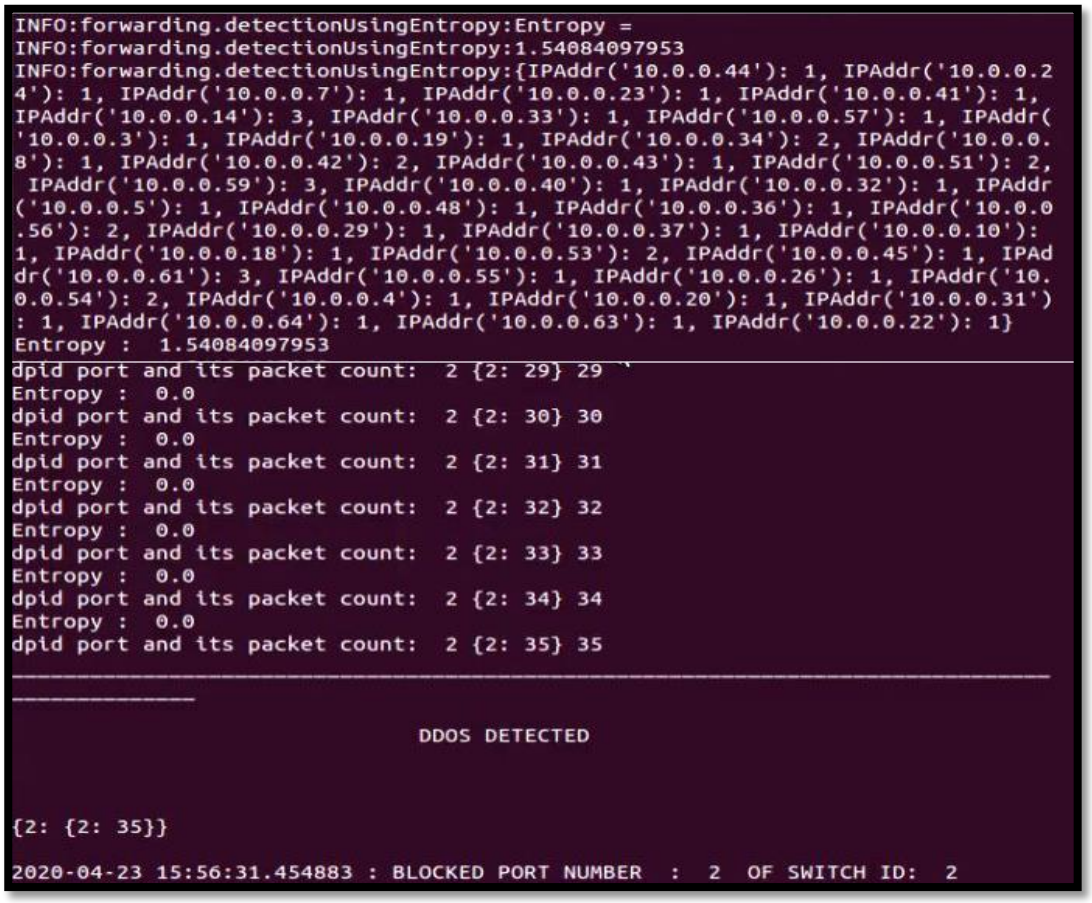
Appx Figure 4.11 Python DoS Attack Script (partial) part 1

```
c.  fieldnames = ['BSSID', 'First_time_seen', 'Last_time_seen', 'channel', 'Speed', 'Privacy',
'Cipher', 'Authentication', 'Power', 'beacons', 'IV', 'LAN_IP', 'ID_length', 'ESSID', 'Key']
d.  if ".csv" in file_name:
e.  with open(file_name) as csv_h:
i.  # This will run multiple times and we need to reset the cursor to the beginning of the file.
ii. csv_h.seek(0)
iii.    # We use the DictReader method and tell it to take the csv_h contents and then apply
the dictionary with the fieldnames we specified above.
iv. # This creates a list of dictionaries with the keys as specified in the fieldnames.
v.  csv_reader = csv.DictReader(csv_h, fieldnames=fieldnames)
vi. for row in csv_reader:
vii.    # We want to exclude the row with BSSID.
viii.   if row["BSSID"] == "BSSID":
ix. pass
x.  # We are not interested in the client data.
xi. elif row["BSSID"] == "Station MAC":
xii.    break
xiii.   # Every field where an ESSID is specified will be added to the list.
xiv.    elif check_for_essid(row["ESSID"], active_wireless_networks):
xv. active_wireless_networks.append(row)

82. print("Scanning. Press Ctrl+C when you want to select which wireless network you want to
attack.\n")
83. print("No |\tBSSID              |\tChannel|\tESSID                         |")
84. print("___|\t___________________|\t_______|\t______________________________|")
85. for index, item in enumerate(active_wireless_networks):
86. # We're using the print statement with an f-string.
87. # F-strings are a more intuitive way to include variables when printing strings,
88. # rather than ugly concatenations.
89. print(f"{index}\t{item['BSSID']}\t{item['channel'].strip()}\t\t{item['ESSID']}")
90. # We make the script sleep for 1 second before loading the updated list.
91. time.sleep(1)

92. except KeyboardInterrupt:
93. print("\nReady to make choice.")

94. # Ensure that the input choice is valid.
95. while True:
96. # If you don't make a choice from the options available in the list,
97. # you will be asked to please try again.
98. choice = input("Please select a choice from above: ")
99. try:
100.    if active_wireless_networks[int(choice)]:
101.    break
102.    except:
103.    print("Please try again.")

104.    # To make it easier to work with and read the code, we assign the results to variables.
105.    hackbssid = active_wireless_networks[int(choice)]["BSSID"]
106.    hackchannel = active_wireless_networks[int(choice)]["channel"].strip()

107.    # Change to the channel we want to perform the DOS attack on.
108.    # Monitoring takes place on a different channel and we need to set it to that channel.
109.    subprocess.run(["airmon-ng", "start", hacknic + "mon", hackchannel])
```

Appx Figure 4.12 Python DoS Attack Script (partial) part 2



Appx Figure 4.13 Entropy tool startup, listening and calculating entropy value for current SDN



Appx Figure 4.14 Entropy tool startup, listening, calculating entropy value for DDiD model, and detecting the attacker host.

Appx 4.15 Standard Deviation value calculation from entropy in case of current SDN architecture:

σ: 0.03506211066094

Count, N: 10 number of attemted
Sum, Σx: 13.95054
Mean, μ: 1.395054
Variance, σ^2: 0.001229351604

Steps

$$\sigma^2 = \frac{\Sigma(x_i - \mu)^2}{N}$$

$$= \frac{(1.36330 - 1.395054)^2 + ... + (1.42532 - 1.395054)^2}{10}$$

$$= \frac{0.01229351604}{10}$$

$$= 0.001229351604$$

$$= \sqrt{0.001229351604}$$

$$\boldsymbol{\sigma = 0.03506211066094}$$

Margin of Error (Confidence Interval)

The sampling means most likely follow a normal distribution. In this case, the standard error of the mean (SEM) can be calculated using the following equation:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{N}} = 0.011087612926144$$

Based on the SEM, The lesser the margin of error, the larger confidence one should have that a poll result would reflect the result of the experiment.

Appx 4.16 Standard Deviation value calculation from entropy in case of SDN with DDiD model:

σ: 0.064767968665074

Count, N: 10 number of attemted
Sum, Σx: 13.40615
Mean, μ: 1.340615
Variance, σ^2: 0.004194889765

Steps

$$\sigma^2 = \frac{\Sigma(x_i - \mu)^2}{N}$$

$$= \frac{(1.36870 - 1.340615)^2 + ... + (1.28719 - 1.340615)^2}{10}$$

$$= \frac{0.04194889765}{10}$$

$$= 0.004194889765$$

$$= \sqrt{0.004194889765}$$

$$\boldsymbol{\sigma = 0.064767968665074}$$

Margin of Error (Confidence Interval)

The sampling means most likely follow a normal distribution. In this case, the standard error of the mean (SEM) can be calculated using the following equation:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{N}} = 0.020481430040405$$

Based on the SEM, The lesser the margin of error, the larger confidence one should have that a poll result would reflect the result of the experiment.

استخدام نموذج الدفاع العميق الديناميكي لتحسين أمن شبكات SDN

إعداد

محمود فرج الحجازي

المشرف

د. محمد مصباح

الخلاصة

لم تعد الشبكات التقليدية وتقنياتها القائمة منذ فترة طويلة مناسبة لمتطلبات الشبكات الآخذة في التوسع في المستقبل، وتحديداً أتمته اتصالات الشبكة وإمكانية برمجتها. الشبكات المعرفة بالبرمجيات (SDN) هي الحل الأكثر قبولًا لذلك. يستخدم SDN بشكل منطقي وحدة تحكم مركزية تطبق واجهة برمجة تطبيقات مفتوحة قياسية (API) للتحكم المباشر في وظائف معالجة الحزم لأجهزة الشبكة. يعد OpenFlow حاليًا بروتوكول الاتصال الرئيسي والمعروف على نطاق واسع في هيكلية SDN. نتيجة لمثل هذه المركزية، أصبحت هيكلية SDN لوحدة التحكم كنقطة فشل واحدة مع المزيد من أسطح القابلة للهجوم لكل طبقة. يستلزم ذلك البحث عن مزيد من إجراءات الأمان والحماية لمعمارية SDN دون التضحية باستجابتها السريعة لمتطلبات العمل المتغيرة. تهدف هذه الورقة إلى تعزيز حماية مفهوم الأقسام في بنية SDN، مما يقلل من إنشاء المزيد من أسطح الهجوم التي يمكن أن تستهدفها الأنشطة الضارة. وبالتالي، يركز البحث على تصميم نموذج منصة تحكم SDN يمكن الاعتماد عليه عبر تقنيات Defense In-Depth (DID)، بما في ذلك متطلبات وحدة تحكم آمنة ومرنة وقوية. يُقترح نشر نموذج Dynamic Defense In-Depth (DDiD) لطبقة تحكم SDN لتعزيز أمان بروتوكول OpenFlow العام. التفصيل في التهديدات القابلة للقياس وآليات الحماية، وفقًا لنموذج DDiD، حيث يتم التحقيق فيها وتنفيذها باستخدام بيئة المحاكاة (mininet). تقترح الورقة أيضًا آلية تقييم (واختبارات) لتأكيد قابلية تطبيق متطلبات الهيكل الآمن على طبقة تحكم SDN. بقيمة متباينة تبلغ ± 0.02 وأقصى فرق ~ 59.51٪ في مستوى حماية أفضل. تؤكد النتائج التي تم الحصول عليها على انه هناك إمكانيات واعدة في النموذج المقترح لتحقيق الأهداف الأمنية المطلوبة.

استخدام نموذج الدفاع العميق الديناميكي لتحسين أمن شبكات SDN

قدمت من قبل:

محمود فرج الحجازي

تحت إشراف:

د. محمد مصباح

قدمت هذه الرسالة استكمالا لمتطلبات الحصول على درجة الماجستير في

علوم الحاسوب

بتاريخ 2022.10.22

جامعة بنغازي

كلية تقنية المعلومات

2022