View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by King Fahd University of Petroleum and Minerals

SOFTWARE DEFINED NETWORKING BASED MOVING TARGET DEFENCE MECHANISM

BY

Ahmed Salman Hasan Al Naser

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER NETWORKS

April, 2019

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Ahmed Salman Al Naser** under the direction of his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfilment of the requirements for the degree of **Master of Science in Computer Networks.**

[illegible]

Prof. Tarek R. Sheltami
(Advisor)

[illegible]

Dr. Farag A. Azzedin
(Member)

[illegible]

Dr. Anas A. Al-Roubaiey
(Member)

[illegible]

Dr. Ahmad Almulhem
Department Chairman

[illegible]

Prof. Salam A. Zummo
Dean of Graduate Studies

عمادة الدراسات العليا
DEANSHIP OF GRADUATE STUDIES

24/11/19

Date

© Ahmed Salman Al Naser

2019

TO MY FAMILY

ACKNOWLEDGMENTS

To begin with, I am most grateful to Allah for the good health and wellbeing that were allowed me to complete this thesis. I would also like to express my sincere appreciations to my advisor Professor Tarek Rahil Sheltami for his continuous support and guidance during my master degree studies and supporting me with my research. Also, I would like to acknowledge him for his insightful comments, patience, and motivation. Furthermore, I would like to extend my thanks to the rest of my thesis committee: Dr. Farag Azzedin and Dr. Anas Al-Roubaiey for their comments, suggestions, and encouragement, but also for the different questions which incentivized me to widen my research from different perspectives.

Table of Contents

List of Tables

List of Figures

LIST OF ABBREVIATIONS

AG	Attack Graphs
API	Application Programming Interface
APT	Advanced Persistent Threats
CLI	Command Line Interface
CPU	Central Processing Unit
DHCP	Dynamic Host Control Protocol
DNS	Domain Name System
DDoS	Distributed Denial of Service
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol VII
LAN	Local Area Network
MAC	Medium Access Control
MTD	Moving Target Defence
NAT	Network Address Translation
OFP	OpenFlow Protocol
ONF	Open Networking Foundation
ONOS	Open Network Operating System
OSI	Open Systems Interconnection
PC	Personal Computer
QoS	Quality of Service
SDK	Software Development Kit
SDN	Software Defined Networking
SMT	Satisfiability Modulo Theory
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTL	Time To Live
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
VM	Virtual Machine
WAN	Wide Area Network

Abstract

Full Name	:	[Ahmed Salman Hasan Al Naser]
Thesis Title	:	[SOFTWARE DEFINED NETWORKING BASED MOVING TARGET DEFENCE MECHANISM]
Major Field	:	[Computer Networks]
Date of Degree	:	[May 2019]

Computer networks management and configuration is challenging due to their complexity and dynamic nature. Configurations are often done in a decentralized fashion because the network devices are separated, and each controlling device has its own resources and logic. Software Defined Networking (SDN) is one of the new paradigms proposed to solve the management complexity issues with large networks. It offers a centralized point of control through separating the network planes. At the heart of SDN lies the controller which is an entity responsible for managing and configuring the various parts of the network from switches, routers and access policies.

There have been many solutions to protect an SDN based network and boost its security. However, few works have addressed protecting the SDN controller itself. In this work, we use Moving Target Defence (MTD) which is a technique where the environment is constantly changing to deceive potential attackers and protect the SDN controller from Distributed Denial of Service (DDoS) attacks. We survey the literature for techniques that employ SDN-based MTD as security measure. Then, propose an approach that makes use of a distributed SDN controller environment and apply MTD on it. Finally, the mechanism is implemented and tested.

ملخص الرسالة

الاسم الكامل: أحمد سلمان حسن آل ناصر

عنوان الرسالة: حماية الشبكات المعرفة بالبرمجة باستخدام الدفاع المتحرك

التخصص: شبكات الحاسب الآلي

تاريخ الدرجة العلمية: مايو، 2019

إدارة الشبكات واعدادها تشكل تحديا نظرا لطبيعتها المتغيرة والمعقدة. الاعدادات تدخل على الشبك بشكل غير مركزي لكون الأجهزة منفصلة عن بعضها البعض، كل جهاز يتحكم بموارده ومنطقه. الشبكات المعرفة بالبرمجة هي احدى الطرق الجديدة التي طرحت لتحل مشاكل إدارة الشبكات الكبيرة. توفر هذه الطريقة وسيلة للتحكم بالشبكة بطريقة مركزية بواسطة فصل منطق التحكم من الشبكة. في قلب الشبكات المعرفة بالبرمجة يقع المتحكم ووظيفته إدارة وإعداد أنحاء الشبكة المتعددة من مبدلات الشبكة وموجهاتها وسياسيات التمكن.

يوجد العديد من الحلول لحماية الشبكات المعرفة بالبرمجة. لكن القليل من هذه الحلول يركز على حماية المتحكم. في هذا العمل، نقوم باستخدام أحد التقنيات الجديدة المسمية بالدفاع المتحرك. هذه التقنية تهدف إلى تغير بيئة الشبكة بشكل متواصل لخداع المهاجمين وحماية الشبكة من هجمات المنع من الخدمة. نقوم بالبحث عن الحلول الموجودة لحماية الشبكات المعرفة بالبرمجة التي تستخدم هذه التقنية الجديدة. بعد ذلك، نطرح طريقة لحماية المتحكم باستخدام أسلوب غير مركزي للتحكم بالشبكة. في النهاية، نقوم بتصميم الحل المقترح واختباره بعدة طرق لإيجاد مدى فعاليته.

Chapter 1

Introduction

Computer networks management and configuration is challenging due to their complexity and dynamic nature. A typical network consists of a large number of nodes such as switches, routers, and many types of middleboxes [1]. Events are continuously occurring in such networks in a simultaneous manner. Networks are maintained by network operators that are responsible for configuring the network as well as enforcing high level policies while responding to events that could occur. In order to enforce these policies, low-level configuration in network devices must be carried by the operator.

Configurations are often done in a decentralized fashion because the network devices are separated, and each controlling device has its own resources and logic. These devices must be accessed individually and most of the time that is done through a command line interface (CLI). Moreover, the configuration is done on a network snapshot most of the time. This means that network changes that happen continuously must be accommodated manually by the network operator [1].

Network operators need some tools and methods that can support their work when managing and configuring network. This has proven to be difficult since most networks are composed of many propriety-based devices that are interconnected together. This led to the development of solutions to focus on temporary solutions such as tools that analyse and fix low-level configurations. At the same time, more functionality is required by operators as well as an increase in the complexity levels of these network policies [1].

Software-defined networking (SDN) is an approach that is used in computer networking to programmatically manage, control, and configure networks efficiently in order to improve network performance and monitoring [2]. The main proposal of SDN is the decoupling of the control plane (routing packets), the data plane (packets forwarding), and the application plane. This architecture allows for a centralized approach to management which is easier to maintain and troubleshoot.

SDN offers many advantages over traditional networks. Software is easier to change compared to traditional firmware and can be done through commands in the network devices. Another advantage is the centralization of the management process. Network operators can configure the network through a single point, which is the controller. This is much more convenient and easier than changing distributed devices individually in a traditional network.

Despite its many advantages, SDN also faces several challenges. One being the separation between the control plane and data plane while still offering acceptable levels of service. The architecture must be able to support reliability, scalability, Quality of Service (QoS), and service management [3]. Another challenge that needs to be considered is providing security for SDN. With the separation of planes, the control plane access needs to be authenticated and authorized. Finally, to allow SDN to be adopted by the industry, it must be able to integrate with current networks.

1.1 Problem Statement

Network applications are widely spread and used in both the industry and research. These systems are usually exposed to a wide range of cyber-attacks. With the addition of SDN, the network layers are separated allowing for more ways to exploit the network. One such method is to flood the SDN controller with requests to bring it down. Since the controller is at heart of the SDN platform, if it is brought down the network will cease to function. No new flows will be installed on the switches making new connections impossible. This act cripples the network and therefore stops any applications from contacting each other through the network.

There has been major research in securing the network infrastructure which consists of switches, routers and end devices. Since this area has been covered the security of an SDN-based network has been boosted considerably. However, there is a gap when it comes to securing this platform. The SDN controller needs to be protected against Distributed Denial of Service (DDoS) attacks so that the network can function. The motivation for this work was to fill in the gap of securing the SDN controller from being flooded and brought down. In this paper, MTD is used to secure the SDN controller by introducing a distributed controller environment. The target of this paper is to defend the SDN controller from being brought down by a malicious attacker.

1.2 Objectives

The main target of this research is to propose an approach that addresses the problem of securing the SDN controller from DDoS attacks. For the approach to be beneficial, it must achieve the following objectives:

- Saving Network Bandwidth

  The network should not suffer an overhead that impacts the applications performance by a big margin.

- Self-Configuration

  The proposed solution should adapt the network automatically without manual intervention from the network operator.

- Security

  The network must be secure against DDoS attacks.

- Scalability

  The network should be scalable without violating previous objectives or impacting network performance greatly.

1.3 Methodology

To secure the SDN network, a distributed controller solution is employed. Using a distribute controller provides a robust backend for the network. On top of it, the MTD mechanism is used to swap the controlling entity of each network device periodically. The details of the mechanism required deep knowledge of several topics. In the following chapters, the background required to understand the solution are discussed in length. Below is a simple figure showing the basic idea of the implemented mechanism. As shown, the mechanism swaps the active controllers over time.

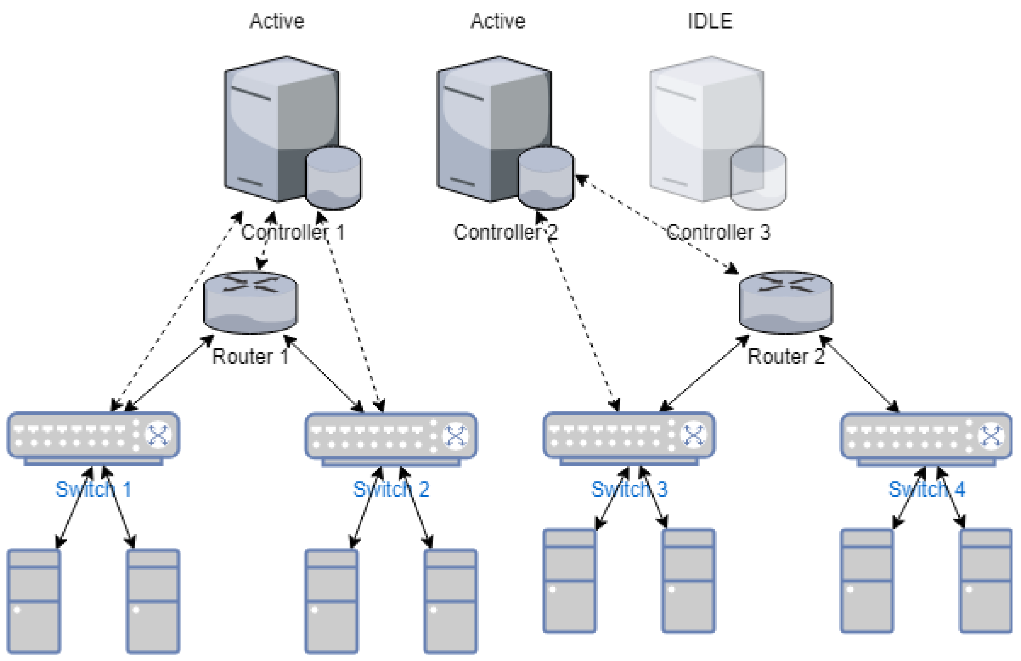


Figure 1 Network Before MTD Cycle

During the first cycle, Controller 1 and 2 are active and will handle the management of all the network devices. Meanwhile, Controller 3 will stay in an idle state, waiting for either the next cycle or when a failure in one of the other controllers happens. In the second figure shown, the second controller goes idle and the third one becomes active. Once the controller becomes active, it will take over the resources that lost their original controller. This process happens in handover fashion. Meaning that before the second controller goes fully idle, it first hands over ownership of the assets it is managing.

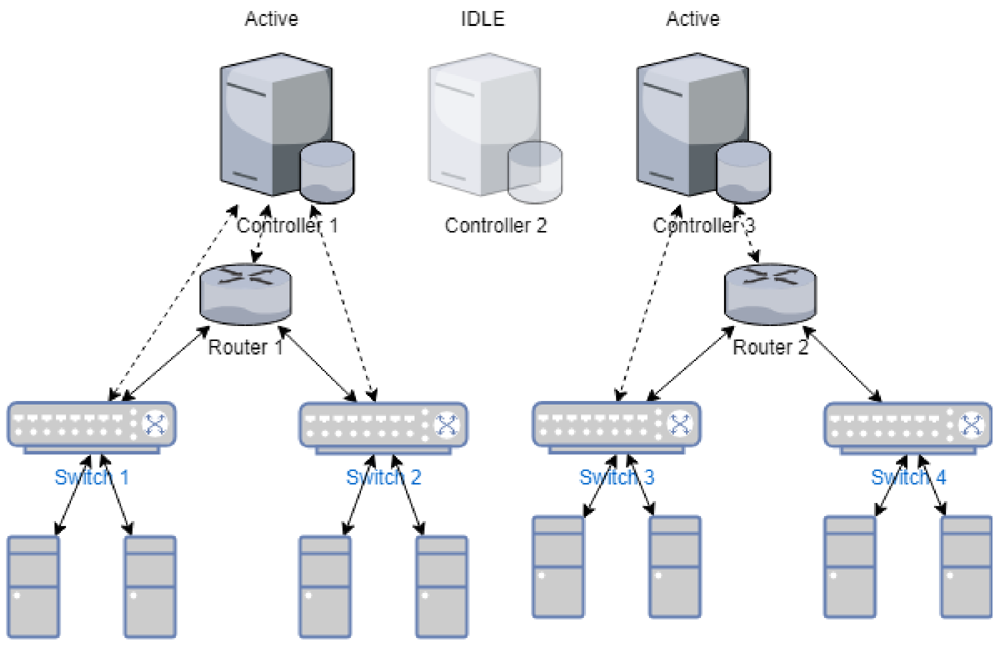


Figure 2 Network After MTD Cycle

1.4 Thesis Structure

This thesis will be structured as follows:

Chapter 1: Introduction

Introduction of the thesis topic and outlining the objectives of the research. Furthermore, an overview of the methodology is shown.

Chapter 2: Software Defined Networking (SDN)

The concept of SDN is discussed in detail including its history, a comparison with traditional networking paradigms, and its architecture.

Chapter 3: OpenFlow Protocol

The OpenFlow protocol is discoursed in this chapter. The need for it is outlined and then its architecture is also shown.

Chapter 4: Moving Target Defence (MTD)

The MTD concept is explained here. The method difference from classical security measures is discussed as well.

Chapter 5: Related Work

The work that is related to this thesis is outlined here. The strengths and weakness are shown for each one. After that, the gap in research is discussed and the need for a new solution is explained.

Chapter 6: Proposed Solution

In this section, the proposed solution is explained in details. The threat model is disclosed and the countermeasures are shown.

Chapter 7: Results Discussion

Here, the results of the implementation are shown and then discussed. Figures are shown to support the research results.

Chapter 8: Conclusion and Future Work

Finally, the conclusion of this work is shown here. Furthermore, possible future work is discussed here.

Chapter 2

Software Defined Networking (SDN)

2.1 Introduction

SDN is defined as the separation of the control logic and data forwarding in a network. The control logic is outsourced from the network devices to a singular entity named the Controller. The Controller manages the network and can be programmed through the usage of Application Programming Interfaces (APIs). In this section, the architecture of SDN is described in detail and its benefits to the network are outlined.

2.2 SDN History

One of the earliest efforts to decouple the control logic and forwarding functions was conducted by the Internet Engineering Task Force (IETF). The team proposed and published a standard interface named the Forwarding and Control Element Separation (ForCES) [4]. Furthermore, IETF pursed the same idea in the form of the Linux Netlink as an IP Service Protocol.

There were two main reasons that these efforts were not adopted by the community. The first reason is that the Internet community at the time feared that the separation of the planes might introduce a risk. The second reason was that the vendors thought that introducing a new standard interface will light up a competition due to the addition of the APIs between the data and control planes.

Later on, an open source implementation led to the creation of one of the major contributors to SDN today, which is OpenFlow. After the introduction of OpenFlow, SDN gained traction in the research field spawning many projects and supporting a wide variety of research topics. In addition to academic uses, OpenFlow was adopted by multiple deployments [5].

2.3 Traditional Networking Paradigm

In traditional networks the control and data planes are coupled together. A switch forwards data based on flow tables within the switch itself. To setup these devices that the network is composed of, each one of them needs to be accessed and configured on its own. This makes it difficult to establish the network and creates a burden on the network operators.

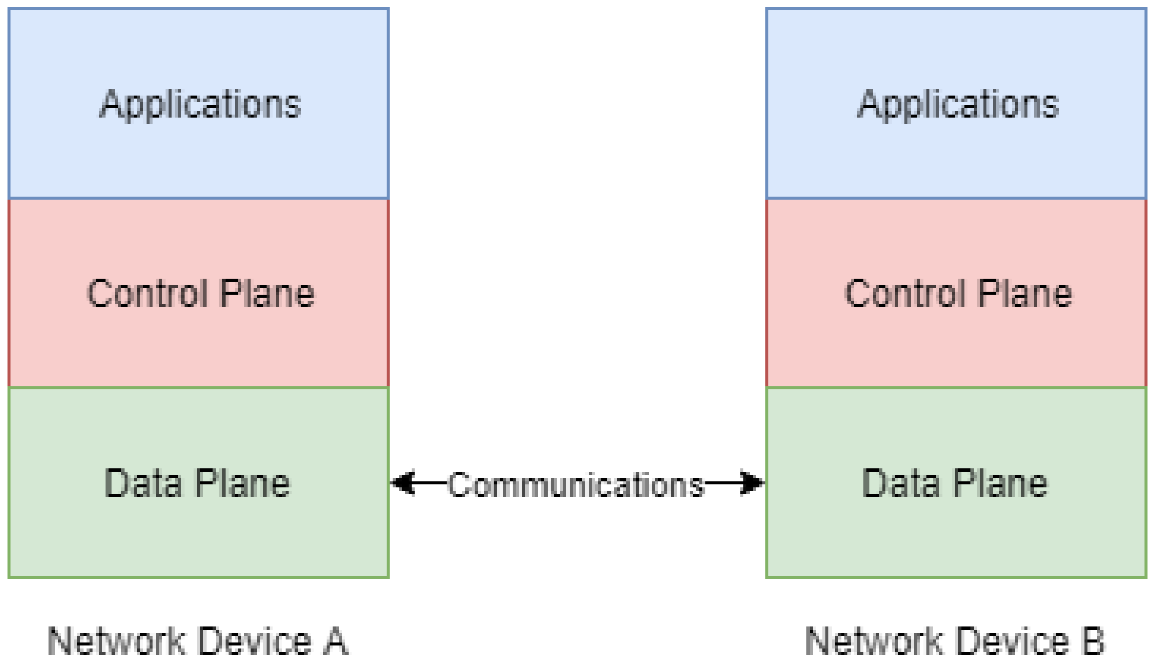


Figure 3 Traditional Network Paradigm

2.3.1 Traditional Network Layers

Traditional networks are based on the Open Systems Interconnection (OSI) Reference Model. This model was published in 1984 by ISO and the International Telecommunication Union. The goal of the model was the standardization of the communications protocols without the need to consider the network technologies or internal structure. The model defines seven abstraction layers that partition the system [6]. In this paradigm, each layer will provide services to the layer below it while receiving services from the layer above it.

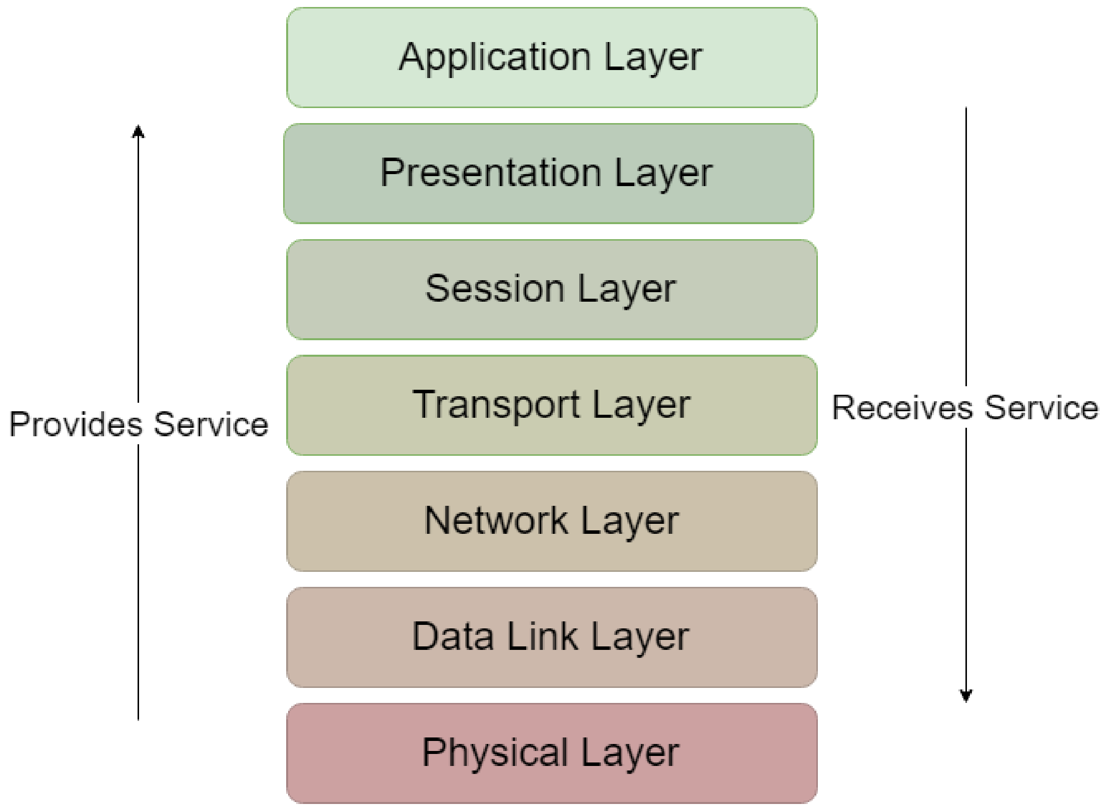


Figure 4 OSI Model

2.3.1.1 Layer 1: Physical Layer

The lowest layer at which raw unstructured data is transmitted by using a transmission medium. The main purpose of this layer is to convert digital bits into signals. In this layer, physical data rates, voltage changes, and physical connections are defined. A network topology can describe the components that constitute this layer.

2.3.1.2 Layer 2: Data Link Layer

This layer provides a node-to-node connection. The link here is direct between two nodes. The main tasks of this layer include detecting and correcting errors that occurred during the Physical Layer operation. Furthermore, it provides flow control between the two connected devices. This layer is divided by IEEE 802 into two sublayers.

- **Medium Access Control (MAC) Layer:** which handles permissions to send data and how a medium is accessed by the devices in the network.
- **Logical Link Control (LLC) Layer:** which handles error detection and link synchronization in addition to identifying the network layer protocols.

For IEEE 802 networks such as 802.3 Ethernet, 802.11 Wi-Fi, and 802.15.4 ZigBee, the MAC and LLC layers operate at the data link layer.

2.3.1.3 Layer 3: Network Layer

Nodes communicate across networks by employing the services of the Network Layer. This layer allows the transferring of variable length data sequences (packets) across different networks through functional and procedural means available for nodes. By definition, many nodes connect to a network, each connected node is permitted to transfer messages to other nodes. Furthermore, each node will have an address. Messages are sent through the network by providing the data and the address of the target node. This means that the network will handle the delivery of the message even if it has to pass through some intermediate nodes. The network will also be responsible of handling large messages by splitting them to fragments as necessary. These fragments are then sent independent of each other and are reassembled at another node. The network has an option to report any delivery errors, but it is not a requirement of this layer. Reliability of messages delivery is not guaranteed in this layer, it is left to the protocol to decide whether to provide reliable message delivery or not. Some examples of the protocols implemented at this layer include Enhanced Interior Gateway Routing Protocol (EIGRP), Internet Control Message Protocol (ICMP), and Internet Group Management Protocol (IGMP) [7].

2.3.1.4 Layer 4: Transport Layer

End to end transfer between hosts is the responsibility of the transport layer. It provides the means to transfer variable-length data sequences in terms of functional and procedural methods, while assuring a level of quality of service. The transport layer functions include flow control of a given link, segmentation, and error control [7]. There can be different types of protocols in this layer, some protocols provide the following services:

- **Maintain a state:** meaning it keeps information about the session.
- **Connection-oriented**: meaning it establishes a connection between the hosts before transmitting any data.

This allows the transport layer to can keep track of the segments and provides the means to avail segments retransmission in case of failures in the delivery. Contrast to that, when no errors occur, the transport layer can acknowledge the successful data transmission and continue sending the next available data. The transport layer also handles messages segmentations on the data received from the application layer. The process of dividing messages that are long and therefore will not normally be transferable into smaller messages is called Segmentation. Examples of protocols that are commonly referred to be operate at this layer but were not developed as part of the OSI model are:

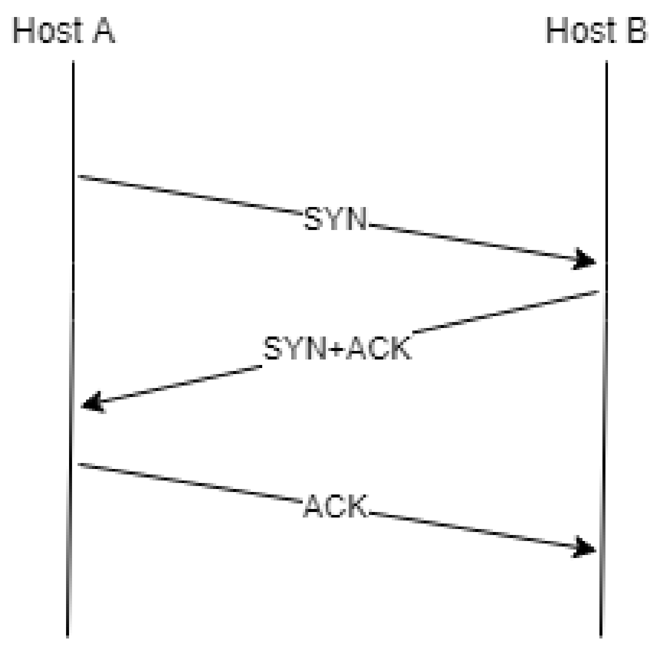


Figure 5 TCP connection establishment

- **Transmission Control Protocol (TCP):** a connection-oriented protocol that provides reliable ordered delivery and error checking.
- **User Datagram Protocol (UDP):** a connectionless protocol that does not provide reliability nor error checking and sends messages in no particular order.

2.3.1.5 Layer 5: Session Layer

Connections between computers are controlled in this layer. At the application level, connections are established then managed between the local and remote hosts. It also terminates the connection. Procedures that allow controlling the session by checkpointing, suspending, restarting, and terminating it are provided with the option of full-duplex, half-duplex, or simplex operation. This layer is present in the OSI model but absent in the Internet Protocol Suite. Its functions, which include gracefully closing a session, are handled in the Transmission Control Protocol in the implementation of the Internet Protocol Suite. Applications that make use of remote procedure calls often implement this layer [7].

2.3.1.6 Layer 6: Presentation Layer

Application layer entities make use of the presentation layer to establish a context between each other, in which a mapping between them might be provided that allows using different syntax and semantics [7]. Presentation protocol data units could be encapsulated inside the session protocol data units If a mapping is available and then they can be passed down the protocol stack.

By providing a translation between application and network formats, this layer can provide independence from data representation. The application takes the transformed data from the presentation layer in a form it accepts. Data formatting is handled by this layer when sending across the network and it is sometimes referred to as the syntax layer. Compression can be availed in this layer.

2.3.1.7 Layer 7: Application Layer

The end user interacts with the application layer making it the OSI layer closest to him/her, this creates a common point of contact between the user and the OSI model as both of them access the application layer. A communication component is needed to be implemented in the software when interacting with this layer. As far as the OSI model scope goes, such application programs don't belong in it. The main functions of this layer include identifying communication partners, determining resources availability, and synchronizing the communication between the partners.

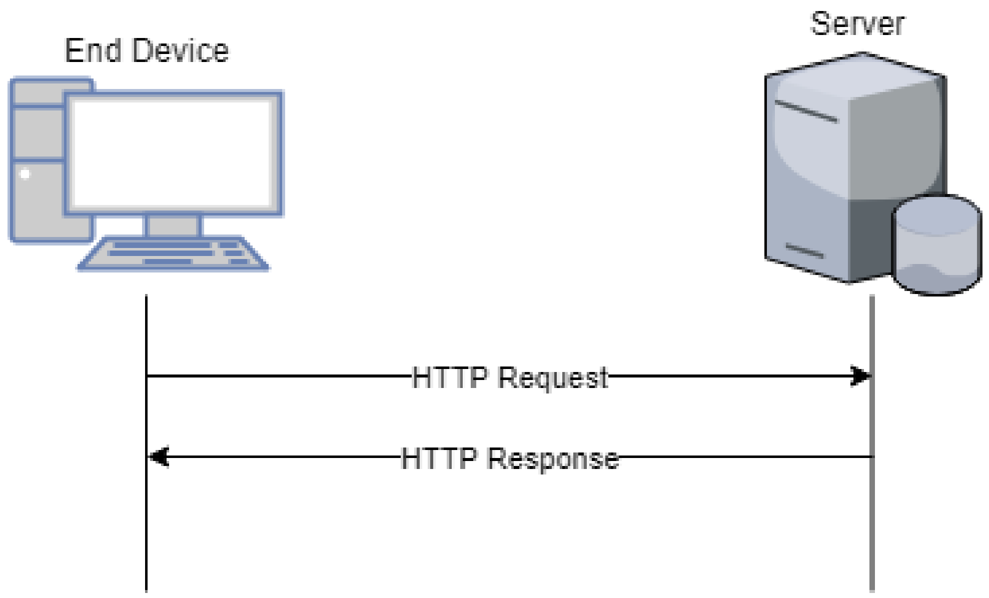


Figure 6 Application Layer Protocol Example - HTTP

To identify a communication partner, the application layer determines the identity and availability of the communication partner for an application with data to transmit. Applications may employ more than one application entity in this layer. Take for example website used to order products. Such site might make use of two application-entities: one using HTTP when communicating with users, while the second records orders and sends them to a remote database protocol. Both these protocols are not particularly concerned with orders. The application implements the for the orders itself. The availability of the resources cannot be determined by the application. Some examples of the protocols that fall under this layer are HTTP, FTP, SMTP, and DNS.

2.3.2 Network Devices

2.3.2.1 Router

In computer networks, forwarding data packets is carried by a device called the router. Over the Internet, routers have the duty of directing the traffic to reach the destination. Data packets could contain things like a web page or an email and is sent through the Internet. Routers forward packets on their interfaces from one to another throughout the network until the packets arrive at the destination network device.

2.3.2.2 Switch

Computer networks employ a device call a network switch (which is also often referred to as switching hub, bridging hub, officially MAC bridge) to connect devices on a computer network in order to carry a multitude of functions including using packet switching to receive, process, and forward data to the destination device. Operating at the data link layer (layer 2) of the OSI model, a network switch is a multiport network bridge that processes and forwards data which is enabled by employing hardware addresses. There exist also some switches that can operate at the network layer (layer 3) which allows them to incorporate routing functionality in addition to their basic functions.

2.4 The Need for SDN

The number of devices that are connected to the internet in the recent years has skyrocketed. Many types of devices are now connected such as mobile devices, smart home devices, and normal desktop devices. The traditional networking architecture is not able to support the increasing needs of the users. Some of the trends that lead the need to introduce a new paradigm capable of supporting a dynamic network include [8]:

- **Traffic Patterns that change:** classically, devices were communicating as client-server to retrieve the data to the user. However, in the recent trends, devices access multiple databases and servers while users are pushing for data from multiple types of devices which might operate in a moving environment.
- **Information Technology Customization:** users are accessing the network from many devices such as smart phones, tablets and notebooks. IT needs to accommodate the

needs of these users while still maintaining a high standard of protecting corporates data and upholding compliance.

- **Cloud Services:** another technology that is becoming more popular is the employment of cloud computing. Enterprises need to access its applications on demand while still maintaining security requirements. The environment must also be easily adaptable to the ever-changing demands of business.

2.5 SDN Architecture

The primary concept of SDN is the decoupling of the intelligence from the forwarding devices and moving it to a centralized controller [9]. As illustrated by Figure 1, the structure of SDN can be divided into application layer, control layer, and infrastructure layer.

2.5.1 Infrastructure Layer (Data Layer)

Consists of the network devices such as routers, switches, and access points. This layer is responsible for forwarding packets according to the rules/policies assigned. This layer forms the infrastructure of the network to work on by connecting the various devices together. Since the control layer is extracted from this model, the devices in the infrastructure are accessed through an abstraction.

2.5.2 Control Layer

The mediator between the infrastructure and application layers. It contains the controller which controls the SDN functions. The controller installs forwarding rules on the switches and routers through programming [9]. Communications between the controller and the infrastructure are enabled by the southbound interface such as OpenFlow. Using the southbound interface, this layer can access the abstractions of the infrastructure layer in order to control the network devices. On the other hand, the northbound interface will be serving as an access point for the applications. This can be done by using APIs to access the devices. As an example, an application could gather statistics about the network devices by using the functionalities of this interface.

2.5.3 Application Layer

The highest layer in the architecture and is responsible for business and security applications. Many applications are contained in this layer such as mobility management, intrusion detection systems, network virtualization. This layer communicates with the controller through the northbound interface.

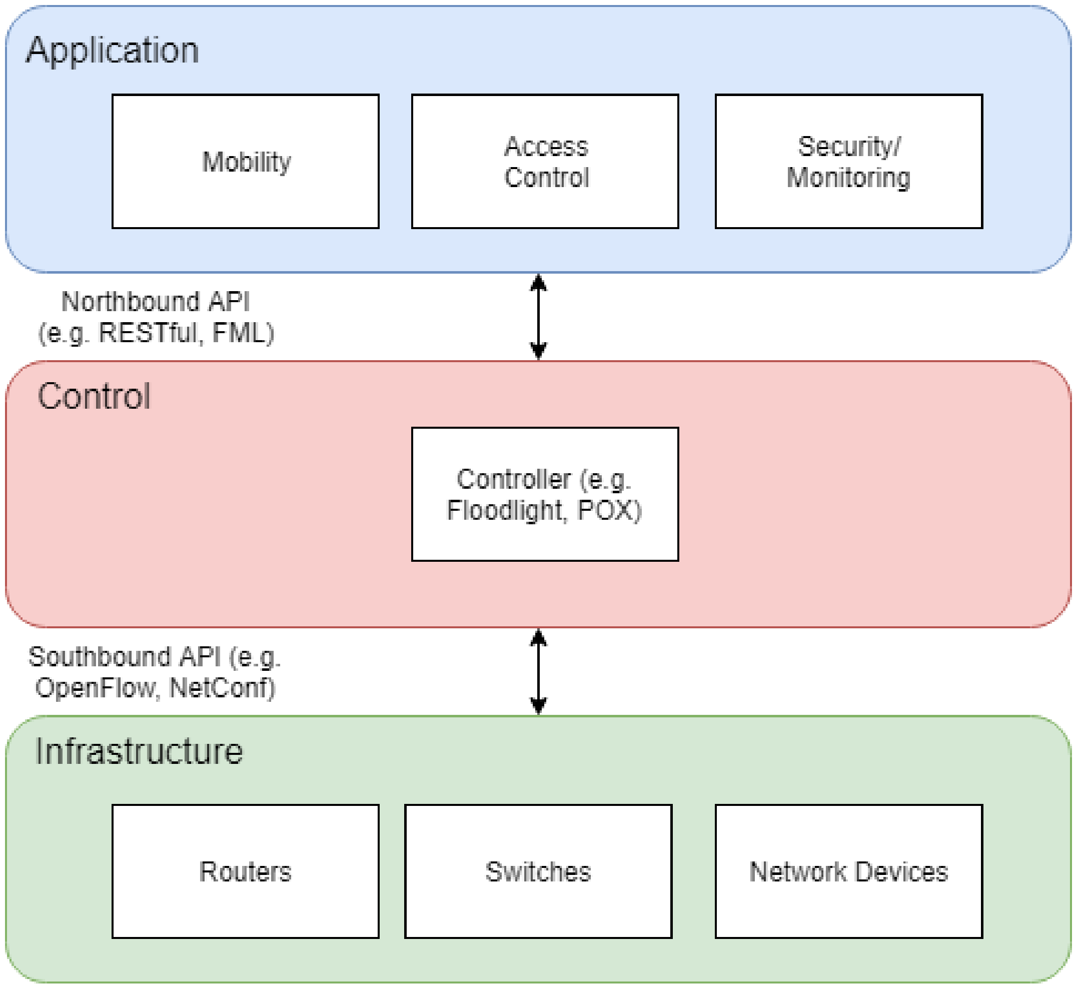

Application
Mobility
Access Control
Security/ Monitoring
Northbound API (e.g. RESTful, FML)
Control
Controller (e.g. Floodlight, POX)
Southbound API (e.g. OpenFlow, NetConf)
Infrastructure
Routers
Switches
Network Devices

Figure 7 SDN Architecture showing the three layers: infrastructure, control, and application.

2.6 SDN Capabilities

SDN is enabled mainly through the usage of a central controller that manages the network elements such as routers, switches, and firewalls through a program. Through this feature, SDN can adapt to dynamic network changes by adjusting the flows to enhance network performance. Furthermore, the abstraction through layers, where different layers can be interfaced by APIs, alleviates the burden on programmers [10]. As consequence of the abstraction, the application layer does not interact with the infrastructure components directly. There are many tools that support this abstraction like Frentic [10] and pyretic [11]. Physical infrastructures can be shared by multiple users through virtualization [12]. Network virtualization software is made available by many companies such as VMWare, Microsoft, Hyper-V, Citrix, Xen server, and RHEL [12].

2.7 SDN Controllers

There are many SDN controllers in the field and several ways to categorize them. One possible categorization is whether a controller is centralized or distributed [13]. In a centralized controller, all forwarding devices are managed by a single entity. This creates a single point of failure and harms scalability. In addition, one controller could fail to handle the management of a large network. Some examples of centralized controllers are NOX-MT [14], Beacon [15], and Floodlight [16].

Alternatively, distributed controllers can adapt to any environment requirements through scaling up [13]. Distributed controllers can be made of either a centralized cluster of nodes or a distributed set of network components. Some examples of distributed controllers are Onix [17], HyperFlow [18], and ONOS [19]. Distributed controllers mostly operate on a weak consistency semantics, this implies that data updates are eventually carried on all controller nodes. Opposite to that, having strong consistency guarantees that the most updated property will be read by all controllers after a write operation. Very few controllers provide strong consistency with Onix and ONOS as examples.

Name	**Architecture**	**Northbound API**	**Consistency**	**License**	**Prog. Language**
Beacon [15]	Centralized multi-threaded	Ad-hoc API	No	GPLv2	Java
Floodlight [16]	Centralized multi-threaded	RESTful API	No	Apache	Java
Kandoo [20]	Hierarchically Distributed	_	No	_	C, C++, Python
HyperFlow [18]	Distributed	_	Weak	_	C++
Onix [17]	Distributed	NVP NBABI	Weak, Strong	Commercial	Python, C
NOX-MT [14]	Centralized multi-threaded	Ad-hoc API	No	GPLv3	C++
OpenDayLight [21]	Distributed	REST, RESTCONF	Weak	EPL v1.0	Java
ONOS [19]	Distributed	RESTful API	Weak, Strong	_	Java
POX [22]	Centralized	Ad-hoc API	No	GPLv3	Python
Ryu NOS [23]	Centralized multi-threaded	Ad-hoc API	No	Apache 2.0	Python
SNAC [24]	Centralized	Ad-hoc API	No	GPL	C++
Trema [25]	Centralized multi-threaded	Ad-hoc API	No	GPLv2	C, Ruby

Table 1 SDN Controllers Classification [13]

Chapter 3

OpenFlow Protocol

3.1 OpenFlow Overview

One of the main protocols enabling SDN is OpenFlow [26]. Through OpenFlow, switches and routers flow tables can be programmed as necessary. Traffic can be partitioned by network administrators to allow research and production to coexist in a network. This allows researchers to conduct experiments and try new security mechanisms on the same network without affecting production.

3.2 OpenFlow History

The OpenFlow standard is under the management of the Open Networking Foundation (ONF) [27]. ONF is the main promoter of the SDN paradigm and it is led by users. OpenFlow has been defined by ONF as the first communication interface that has been standardized to orchestrate the connection between the SDN architecture layers [26]. Network devices can be manipulated by OpenFlow through direct access. Such devices include switches and routers which can be either physical or virtual. OpenFlow serves as way to take back network control from proprietary network switches into an open source one that can be contributed to by the community.

3.3 The Need for OpenFlow

OpenFlow was introduced due to the lack of a solution that could process a huge number of packets needed for a college research project. Commercial products were too expensive and the open source implementations at the time needed for a programmable network were inefficient. A more flexible approach was required that needed to satisfy several assumptions including [26]:

- Provides high performance while minimizing the cost.

- Capable of enabling a wide variety of projects and research subjects.
- Could isolate production traffic from development traffic.
- Compliant with the needs of a closed platform.

3.4 OpenFlow Architecture

In traditional networks, control of the packets flow is handled by the network devices (e.g. switches, routers, etc). These devices are built by commercial companies using their propriety code. Each vendor has its own firmware that needs to be interfaced with in a different way than the other. This makes it difficult for researchers to design and implement their projects using the current available resources. An open source alternative is needed so that the researchers can have a transparent platform to work with.

The idea of OpenFlow is quite simple, flow tables that are already in use by Ethernet switches and routers are reused by the standard to build firewalls, implement NAT, provide QoS, and gather various statistics about the network.

OpenFlow allows to separate the development flows from the productions ones by providing an open protocol to program routers and switches flow tables. Researchers have control on how to handle their flows, starting from how to process them and including the path they traverse. This allows the researchers to experiment with new protocols, security mechanisms, and even implementing IP replacements [26].

OpenFlow switches data path are built with a flow table and for each flow entry, an action is associated. It is possible to extend the list of actions performed by an OpenFlow switch. To provide a high level of performance while incurring a low cost, the data path needs to offer some flexibility. This dictates that a range of actions that are somewhat limited but still prove to be valuable.

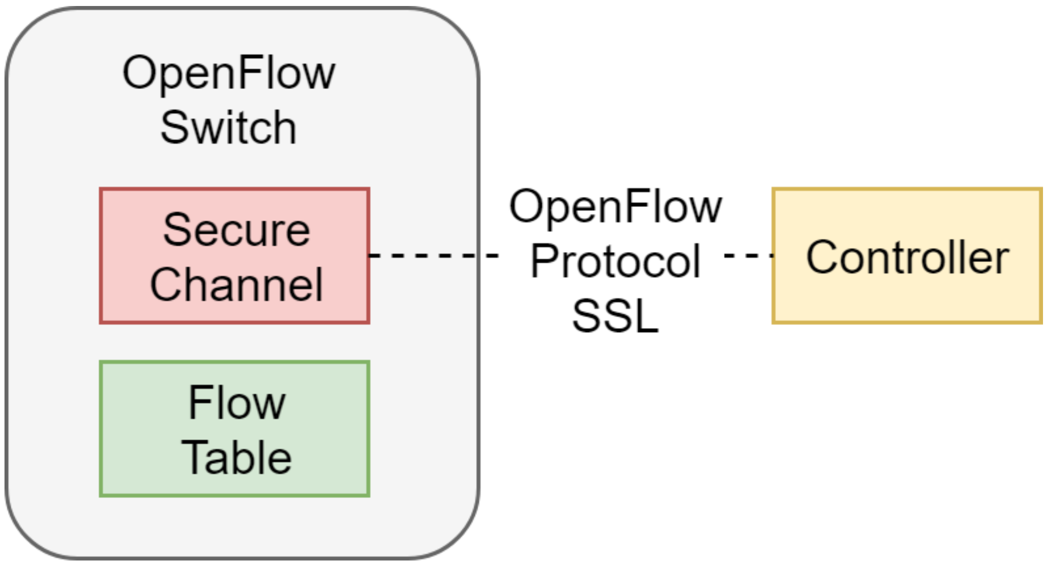


Figure 8 OpenFlow switch architecture.

OpenFlow switches are made of three different essential components. First, (1) flow tables that has an association between flow entries and actions. They also require a (2) secure channel that connects the SDN controller to the switch allowing commands and packets to be exchanged. Finally, the (3) OpenFlow protocol, which is open standard that describes communications between the controller and the switches. Figure 2 shows the described OpenFlow switch architecture.

Switches can be classified into two categories 1) dedicated OpenFlow switches that do not implement any of the layer 2 and layer 3 functionalities and 2) OpenFlow-enabled switches that have been preloaded with the OpenFlow protocol and contain several interfaces.

3.4.1 Dedicated OpenFlow Switches

A dedicated OpenFlow switch acts a dumb forwarding only device that is used a s data path by the controller. From this switch's point of view, flows are constrained by the implementation of the Flow Table. As an example, flows could be just a TCP connection between two hosts. The flow could also be all the VLAN tag sharing packets. For each flow entry, an action is assigned to it [26].

As mentioned before, the actions list can be expanded, however the basic ones that each OpenFlow switch comes with are [26]:

1. Forward the packet to a target port. This is the most basic function of a switch. Without it, routing the packets through the network would be impossible.

2. Encapsulate the flow and send it to a controller. In this action, the packet is encapsulated and sent through the secure channel to the controller. This occurs most of the time when a new flow arrives to the switch and it does not correspond to any flow rules it has. In such cases the controller's aid is needed to determine if a new flow entry needs to be installed to accommodate this packet.

3. Drop the packet flow. This is mostly used to secure the network against attacks. The most basic attack that could harm the network if this action was omitted is the DDoS attack. It would flood the switch with a huge amount of new flows and bring it down.

3.4.2 OpenFlow-enabled Switches

These switches will have the enhanced features of the OpenFlow protocol installed on them. They will also have a secure channel that the dedicated switches can use to communicate with the controller as well as Flow Tables [26]. Most of the time, TCAM will be reused by the switches for Flow Tables to use.

Switch Type	**Dedicated OpenFlow Switch**	**OpenFlow-enabled Swtich**
Forwarding	Yes	Yes
Use Secure Channel	Yes	Yes
Install Flows	No	Yes

Table 2 OpenFlow Switches Types

3.5 OpenFlow Controllers

In the OpenFlow environment, a controller is responsible for adding and removing flows to the switches. A controller can be as simple as an application that runs inside a normal PC. In this case it might be used to interconnect multiple other PCs together for the sake of the experiment. In general, OpenFlow can be thought of as a generalized version of the VLANs concept [26].

At a higher level, a controller can also be much more sophisticated. It could dynamically add and remove flows from the switches. This can occur in real time as the experiment is taking place. Furthermore, a controller can separate multiple experiments flows allowing researchers to reuse it multiple times.

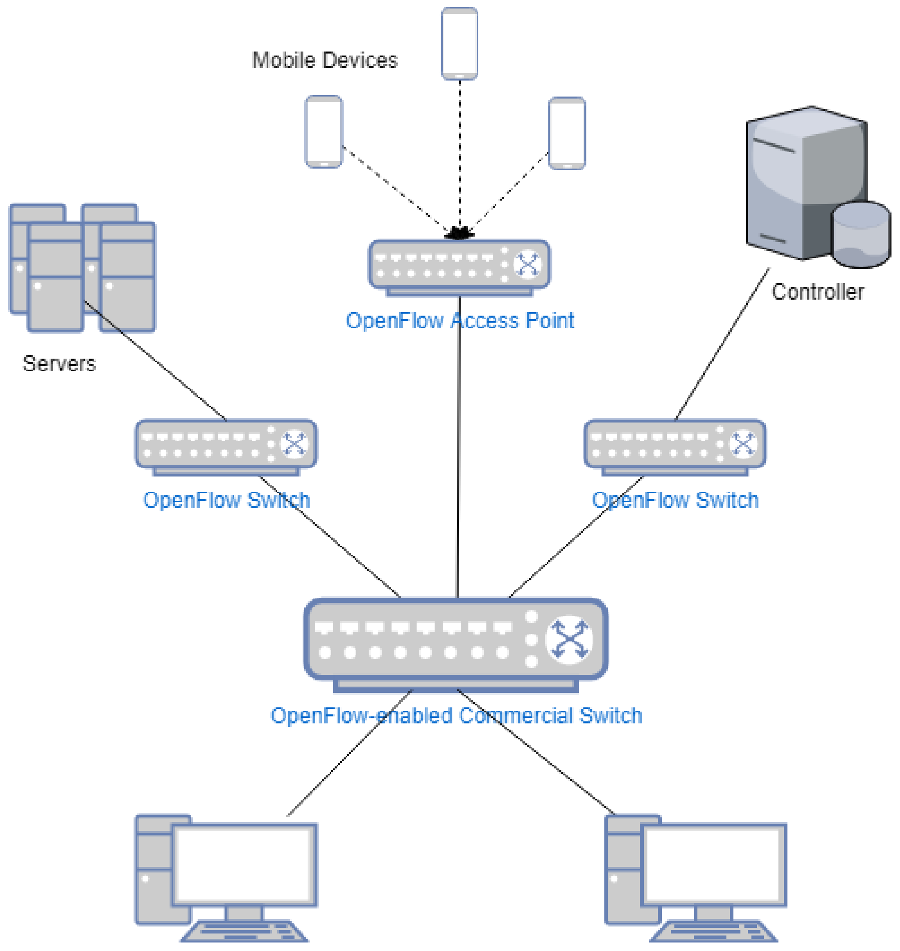


Figure 9 OpenFlow Network Example [26]

3.6 OpenFlow Channel

A communication channel is used between the controller and the network switches specifically for OpenFlow messages exchange. The OpenFlow protocol initiates this channel between the controller and each switch. It is possible to establish more than one connection originating from one switch to a single controller or even to multiple controller. The connections can be classified to either a master connection or a slave connection. Some of the connections can be established as equal as well [26].

Chapter 4

Moving Target Defence (MTD)

4.1 Traditional Security and Attacks Methodology

In classical security mechanisms, the assumption is a static network setup with minimal changes in topology. To attack such networks, the malicious entity first surveys it in order to find out potential vulnerabilities in the network. After identifying such vulnerabilities, the attacker attempts to exploit them and either bring down the network, or attempt to acquire unauthorized access on the assets of the network. Furthermore, an attacker could control some of the assets and make them act as zombies for future activities.

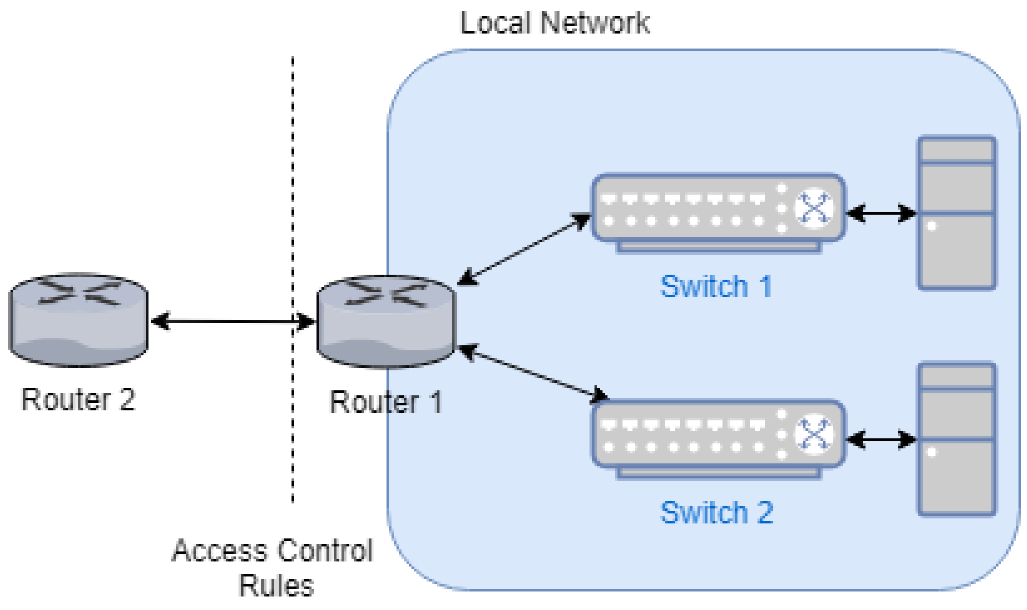


Figure 10 Traditional Network Defence Using Access Control

From a defender point of view, protecting the system can be done by controlling access on the network assets [28]. This could be either a physical isolation or through network access policies. For example, an access policy could be placed on a certain asset to protect it from unauthorized access. This guards the data and blocks attackers from gaining it. This does not come at no cost however, the more rules are added to the network, the harder it becomes for regular users to access the assets. More importantly, this introduces an overhead on the network that delays requests.

Another possible method of guarding the network is by remodelling its structure [28]. This includes patching out system vulnerabilities or upgrading the system resources. Since at the core of an attacker strategy, the existence of a vulnerability is a requirement for an attack to succeed. This approach invalidates attacks that rely on vulnerabilities to succeed. However, it is inefficient to patch out all vulnerabilities in the system, especially if it was a large one consisting of many subcomponents that are developed independently. Some of these might even be acquired from third parties making it harder to identify the vulnerabilities.

It can be concluded from the above that such traditional techniques might be ineffective against a continuous surveying attack alongside with a long-term analysis of the network. Furthermore, finding and amending all the vulnerabilities in a system is difficult task to accomplish cognitively. To summarize, the main challenges [28] are:

- The predefined network structure gives the attackers an easier prey due to it being susceptible to long-term analysis. Malicious entities can collect information about the network for as long as needed before attempting to attack it. From that point on, the attacker would have the advantage which might include zero-day vulnerabilities. While on the other hand, the defender has a hard time exploring all possible attack vectors and patching out all vulnerabilities.
- The static aspect of the network allows the attackers to slip in unnoticed malicious plug-ins by carefully placing them after an analysis is conducted. Given enough time, an attacker could open a backdoor for future attacks as well. Meanwhile, the defender might

have trouble detecting intrusions in real time. Not to mention that there is a time frame when a vulnerability is discovered until it is patched that could be exploited by the attacker.

- The connected nature of the network provides a perfect target for attackers. Once an attacker gains access through one point of the network, attacking the other assets becomes relatively easier. In contrast to that, a defender needs to maintain a security level on all the access points and all components of the network. Providing a comprehensive defence on the network is necessary although tough to achieve.

4.2 Moving Target Defence Definition

The concept of MTD has been introduced in 2009 at a security summit. The definition kept being clarified over the years. Recently, MTD is defined as a technique that formulates, creates, analyses, and deploy a system in a continuously changing manner while maintaining complexity to deceive attackers and increase the time required to study the system in order to increase the system resiliency [28].

Under traditional networks defence mechanisms, the attack surface that an attacker observes expands time goes on. This is due to the static nature of the network. Moreover, the certainty of the configuration keeps increasing since it does not change over time. To counter this, MTD's constant changing of the network layout creates a bigger attack surface that keeps mutating over time. This reduces the effectiveness of the analysis on the network and increases the cost of the attack.

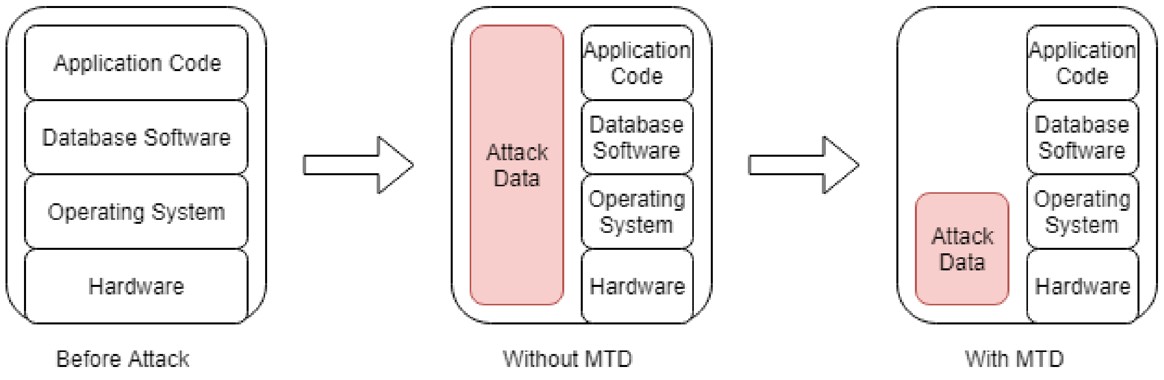


Figure 11 Attack Surface with and without MTD [28]

4.3 Moving Target Defence Overview

At the core of MTD is the adaptation process [29]. Initially, a system is deployed in a normal state that it can operate on. After it starts running, an MTD begins adapting the system's configurations. The adaptation can be triggered through an intrusion detection system (IDS) alert or by evaluating the environment data. The process of adapting the system occurrence can be fixed to certain timings or can be randomized. Furthermore, it can also be triggered through an external source. Any new changes in the configurations must be valid, meaning that these changes must satisfy the system's constraints. If the adaptation process is found to be valid, it is implemented on the target system.

The main goal of an MTD system is to remove the attackers' advantage of time [29]. The traditional approach to securing a system is by reducing the attack surface [30]. This means that the system is protected by removing unnecessary software, closing unused ports, or updating the software to its most recent iteration. However, this leads to several problems including insufficient authentication mechanisms, complex firewall rules, and places a substantial overhead on maintaining access control and credentials. These issues usually lead to an extended period of the configurations remaining static.

With MTD, the attack surface is hardened through adaptive approaches. These approaches attempt to make changes to the attack surface at runtime, this alleviates the burden on the administrator by automating the responses to the system. However, approaches that fall under this category require considerable effort to develop and deploy, not to mention the large number of intrusions/malwares signatures that are maintained by the MTD system. Moreover, such systems can be weak to zero-day exploits.

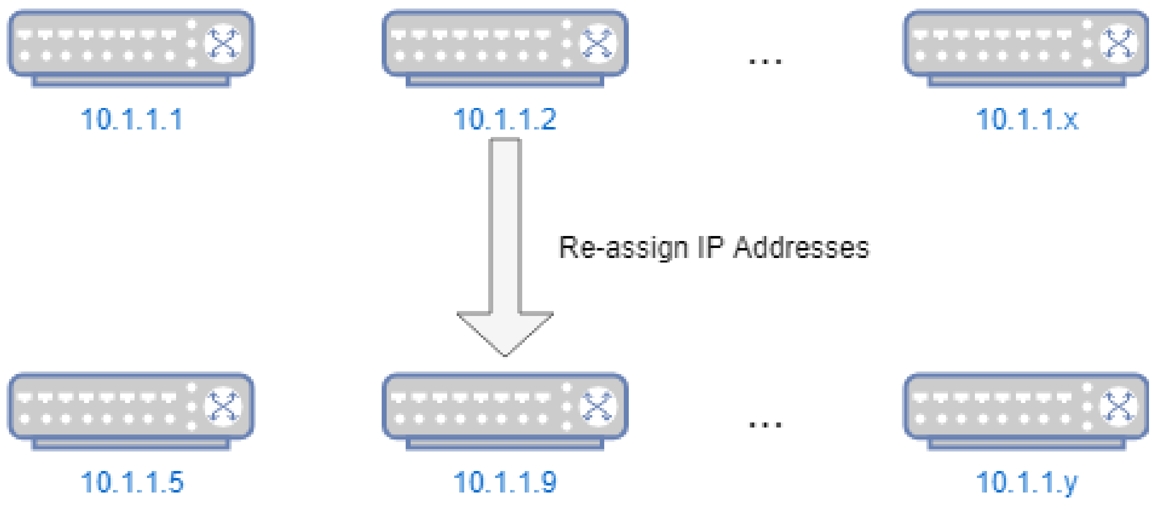


Figure 12 MTD with IP Addresses

A concept introduced by [29] is the exploration surface. It refers to the exploration or reconnaissance part of cyber-attacks that occurs before launching an attack on the system. The purpose of this phase is to understand the system better by scanning the open ports, finding the online machines, or understanding the network topology. The authors of [29] explain how to differentiate between the attack surface and exploration surface through the example of a C class IPv4 address. In this example, the exploration surface consists of the subnet of IPv4 addresses that follow the C class subnet, such as {192.168.0.1, 192.168.0.2, . . . 192.168.0.254}. Where the attack surface can be simply the online computers in this same range. If there is only one, then the size of the attack surface is 1.

4.4 Moving Target Defence and DDoS

MTD proves to be a good technique to mitigate DDoS attacks. In traditional setups, the attacker could flood a controller by sending too many requests to install new flows. This would overflow the controller with requests and will prevent it from serving the actual packets coming into the network. Since the controller would be busy processing the new flow requests, any new flows would suffer a long delay before they get processed, if they do at all.

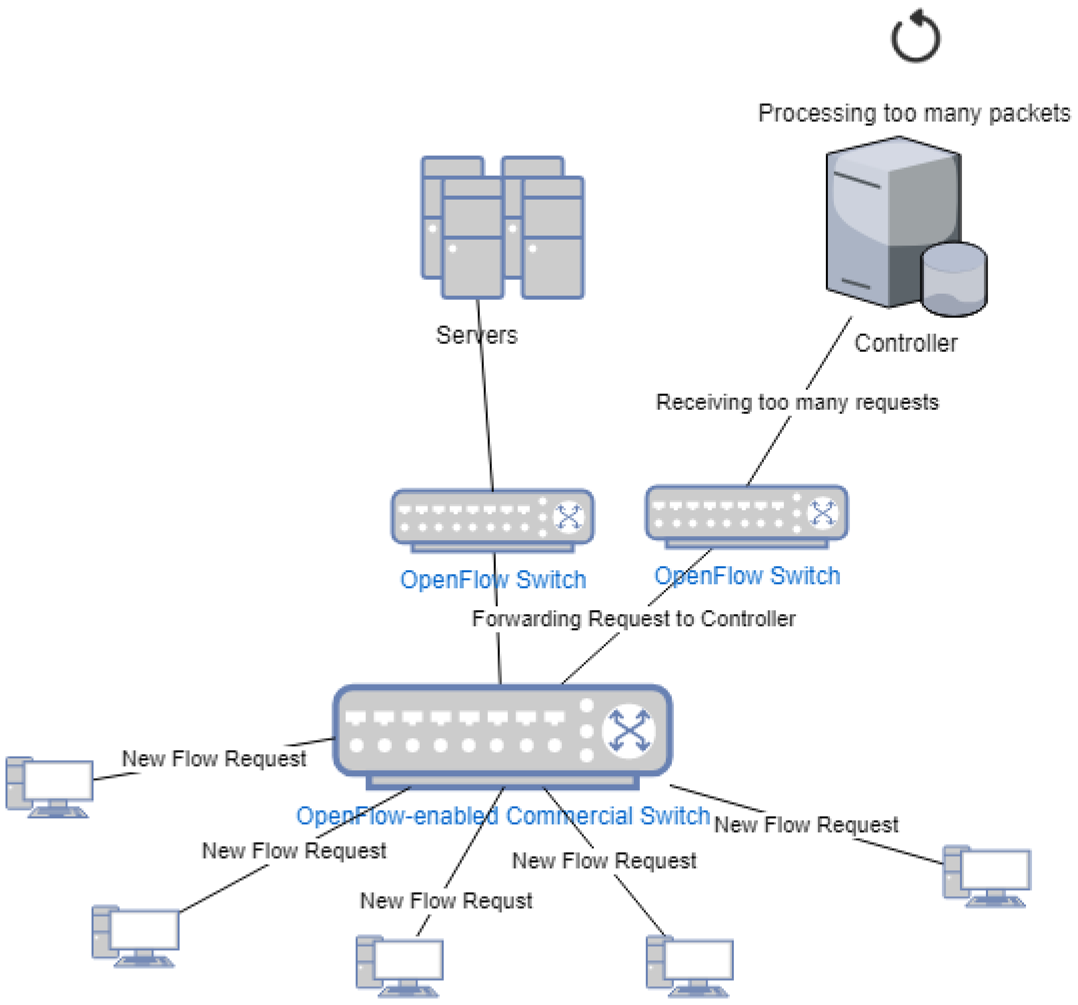


Figure 13 DDoS Attack on SDN Network

Chapter 5

Related Work

There have been many works proposing solutions to secure networks by employing SDN or MTD. Few works attempt to combine the two concepts together to create an adaptive secure system on top of the flexible SDN architecture. In this section, the recent efforts to secure systems using either SDN, MTD, or a combination of the two are outlined and discussed.

MASON, a framework that can be used for cloud vulnerabilities and intrusion events assessment is developed by the authors of [31]. The framework employs port hopping as a countermeasure after it identifies high-security risk in a network service. However, in this approach, QoS is not considered when taking MTD decisions.

A defence by pretence mechanism is proposed by the authors of [32] that focuses on defending against flood attacks. The solution provided by the paper targets cloud environments. The attack the authors were interested in countering is DDoS in the context of SDN. This approach however does not address targeted attacks like Advanced Persistent Threats (APT).

An information flow control security architecture is presented in [33]. The introduced solution combines the placement of the SDN controller and host-based information flow tracking. The authors implement a prototype through a host agent. However, if the host is compromised, the host agent can be disabled which opens the controller to a DDoS attack which will exhaust its resources.

A moving target defence architecture is proposed by [34]. It makes use of random address hopping and random finger printing. The authors also purpose a model that allows network configurations to be mutated validly. Furthermore, the authors of [35] propose a three-layer model for comparing MTD approaches and evaluating their performance. A case study is used as a means to show how the model works and highlight the viability of the model.

IP addresses mutation is used by the authors of [36] as an MTD mechanism within OpenFlow. The approach mutates IP addresses with high unpredictability but still minimizes the overhead on normal operations. This method however focuses only on countering the exploration phase of cyber-attacks. The approach is not studied in its effectiveness to defend against DDoS and application layer attacks.

The authors of [37] study SDN-based MTD techniques and discuss the advantages and disadvantages of such systems. They explain how to evaluate systems of this nature and discuss the challenges and the methods used to overcome these challenges. They conduct several experiments and conclude that SDN-based MTD increases the overhead for the attacker in terms of time and traffic load. However, the focus of the paper is on introducing the concept and the experiments are carried out mainly to study the overhead introduced on the exploration surface. A more advanced system needs to be implemented that can handle more attack cases.

In [38], the authors propose a system to defend against blind DDoS attacks by employing an SDN-based MTD approach. Blind DDoS is one of the threats targeting SDN and is introduced by the authors. It is analysed and an attack defence approach is proposed. The proposed solution works over a multi-controller environment making it scalable. However, the approach makes use of random packet transmission delay which affects normal traffic. Furthermore, synchronizing multi controllers route tables is not considered in the approach, each controller regenerates its own routing tables.

A scalable SDN-based MTD solution to protect cloud networks is proposed by [39]. The approach makes use of Attack Graphs (AG) to assess attack scenarios then select the required countermeasures to reconfigure the network. Furthermore, the paper proposes a conflict detection framework that ensures that a consistent policy state is maintained in a distributed SDN-based cloud environment. This approach focuses on securing cloud environment and might not be suitable for other settings. It also uses a predefined database of signatures making it susceptible to zero-day attacks.

CHAOS, an SDN-based MTD system is proposed by the authors of [40]. In this approach, the hierarchy of all hosts in a network are obfuscated through many mechanisms including IP obfuscation, port obfuscation, and fingerprint obfuscation. They conduct experiments and the results show that information disclosure is reduced while still allowing normal flow of traffic. The paper however focuses mainly on minimizing information disclosure without attempting to counter DDoS attacks.

A technique that is a scalable form of MTD using SDN is proposed in [41]. The approach enables defenders to differentiate between trustworthy and untrustworthy clients by employing pre-shared keys, applying cryptographic MACs, or embedding passwords to provide access control.

In the context of Internet Service Providers, the authors of [42] investigate the effectiveness of MTD techniques using SDN. The authors make use of ONOS [19] and focus on it. The research indicated that the chance of a DDoS attack succeeding decreases as more network traffic is processes by a high number of collaborative partners. The results of the paper also show that wide scale attack on the network can be mitigated through MTD.

In [43], the authors propose a method to secure against unauthorized access by employing an IP address hopping MTD within an SDN environment. The proposed technique is implemented at the hop level as well as at the data plane level which reduces the overhead cause by it. The authors also built an IP address synchronization algorithm that avoids causing overhead on the network by using a one-way hash and piggybacking on the regular communications of the network.

A scheme that randomizes the namespace U-TRI is introduced in [44]. The purpose of this scheme is to hide packets identifiers. The proposed scheme allows for attack surface movement by using a hierarchical virtual namespace. Furthermore, by randomizing the namespace, it boosts the capability to withstand attacks.

A heterogenous redundant security scheduling mechanism is shown in [45]. The authors use a multi-controller setup and the non-dominated sorting algorithm, MDSA, to secure the network while attempting to maintain load balancing. This strategy also relies on the dynamic nature of MTD to adapt the network. In this study, 5 different operating systems are chosen for the sake of simplicity. The authors take into consideration a single controller structure, a static redundancy structure, random scheduling reliability priority scheduling, and MDSA when comparing.

In [46], the authors make use of the SDN network to protect it from DDoS attacks. A formal solution is shown based on Satisfiability Modulo Theory (SMT). The purpose of the solution presented in the paper is to reduce the sharing of critical links across the SDN planes, data and control. However, since the mechanism is making use of the network assets themselves, it generates an overhead on the network.

An MTD based solution to monitor cloud networks and software vulnerabilities targeting attacks and identify them is presented by the authors of [47]. The authors make use of Markov Game to find out the attacker's optimal policy then puts in place countermeasures to defend the network. However, to make the solution work, a linear program must be solved which can become computationally inefficient. To balance this out, the authors assume that only action pairs are used by the defender who has observability of the attacker's action.

Most of the works discussed above attempt at securing the network infrastructure. Meaning they focus on securing the network switches, routers and links. The works discussed make use of MTD in an SDN environment which is a great way to employ it. However, the works do not attempt to secure the SDN controller which is a vital component in an SDN environment. If a DDoS attack were to be launched on the controller, the whole network will be incapacitated.

In this work, the focus will be to secure the SDN controller from DDoS attacks. The concept of MTD will be applied on an SDN controller to protect it from being taken down. Since the network infrastructure security has been discussed by many works as shown above, in this work it will not be tackled. It will be assumed that the adversary’s main target is to bring the network down by attacking the controller.

Chapter 6

Proposed Solution

6.1 System Model

6.1.1 Threat Model

In the attack model used in this work, it is assumed that the attacker can be located inside or outside the network. The main target of the attacker is to bring down the SDN controller of the network rendering the network useless. The attacker may use zombie machines in order to perform a DDoS attack on the controller. In this solution, the network devices such as switches, routers and end devices are not part of the defence mechanism. The mechanism does not handle encrypted traffic, nor does it use IDS to detect attacks. The proposed solution can work in conjunction with other security measures that attempt to scramble the network devices IP address periodically.

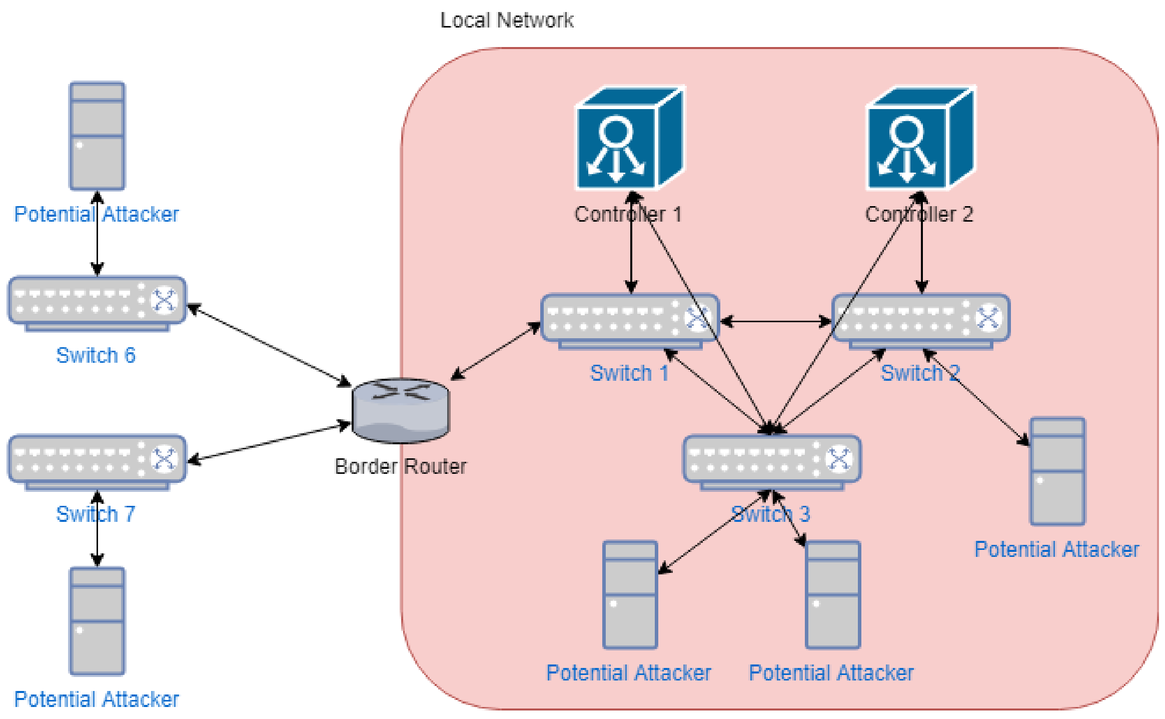


Figure 14 Threat Model

6.2 Proposed Solution Overview

The approach will attempt to secure the controller from DDoS attacks and exploration attacks while still providing acceptable performance to normal traffic. The main challenges include supporting scalability, hiding information from attackers, and protecting the infrastructure availability. To this end, a distributed controller system will be used as the building stone for the MTD mechanism. Since by design, distributed controllers will have the capability to withstand DDoS attacks to an extent, adding the MTD mechanism on top of them will boost the availability of the system greatly. The next sections describe the structure of distributed controllers' systems and explain the proposed method to secure them in detail.

6.2.1 Distributed Controller Network

A distributed controller model is shown in the figure below. The controllers communicate with each other for updates and maintain a virtually centralized state of the network. The OpenFlow switches are connected through a secure channel to their respective controller. In addition to that, the switches have their normal communication links with other switches regardless of the controller they belong to.

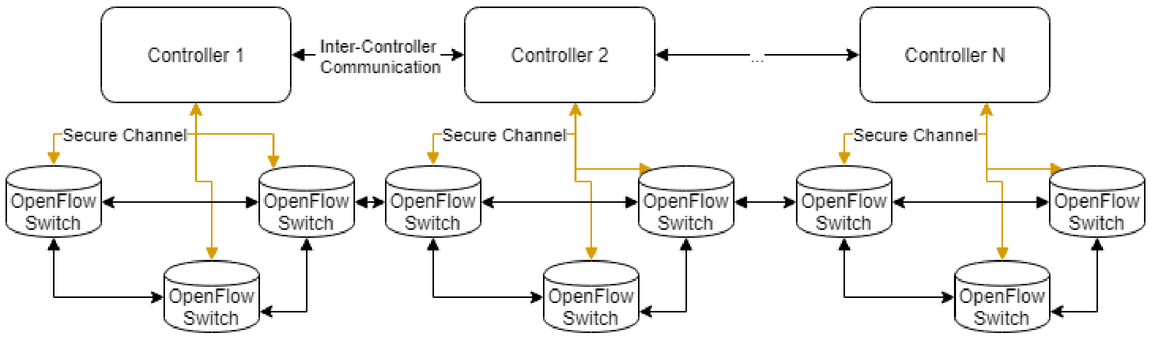


Figure 15 Distributed Controller Topology

The model shown above is often described as flat SDN control. This is since the network is divided into horizontal areas with each area being managed by a controller. This model provides reduced control latency and improved resiliency.

6.2.2 Moving Target Defence Algorithm – Groups

In order to secure the network against attacks, an MTD mechanism is designed and implemented. The algorithm for this mechanism is shown below. It assumes that X controllers and N devices (switches and hosts) exist in the topology. The controllers will be distributed into two groups. Every *S* second, the algorithm will re-assign the devices into the other group controllers equally. While a group is active, the other group will go into idle mode where the members will wait until the algorithm wakes them up, or if a failure occurs in the network.

The mechanism assigns to each device in the network a unique identifier. This identifier is used when changing the device master. Each device will have a corresponding identifier and the identifiers are updated at the beginning of each cycle. When new devices are discovered by the controller, they will be assigned new identifiers.

Algorithm 1: MTD Groups Pseudocode

```
Assuming Two Controller Groups
INPUT:  S (float), ControllersList (List), CurrentActiveGroup (int)
1.      WHILE (TRUE)
2.          SLEEP (S SEC)
3.          ControllersList = OnlineControllersList();
4.          FOREACH Device IN ControllersList DO:
5.              IF CurrentActiveGroup == 1:
6.                  AssignDeviceToMaster(Device, Group_2)
7.              ELSE
8.                  AssignDeviceToMaster(Device, Group_1)
```

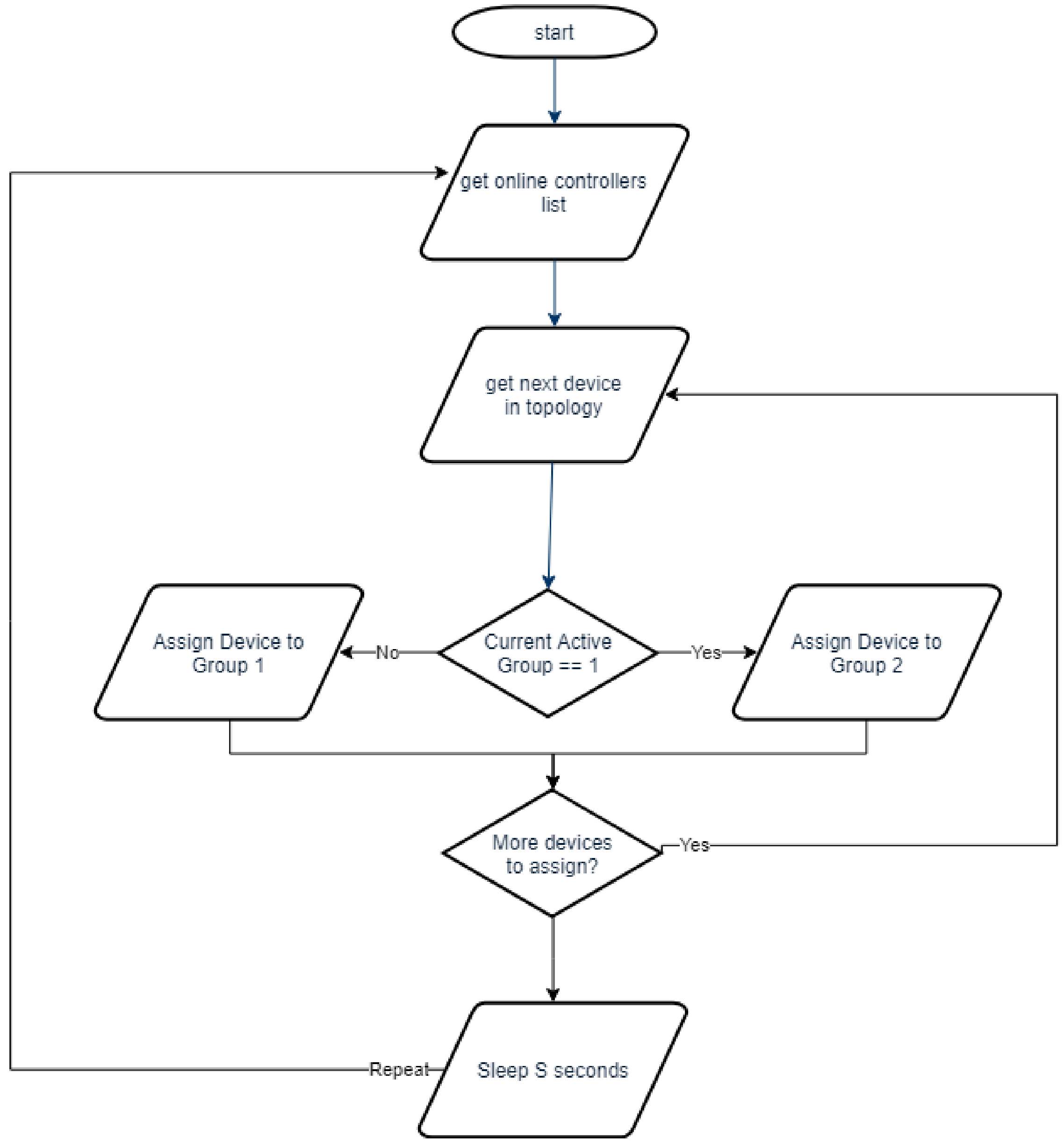


Figure 16 Algorithm 1 Flow Chart

By using this algorithm, an attacker will not be able to identify a target to bring down. An important design decision is the length of the switch interval. This interval dictates when the mechanism will switch between the controller groups. If this interval is set to a small value, the switch will happen too frequently which will disrupt the network and impact performance heavily. However, if it is set to a long period, it will give enough time to the attacker to explore and analyse the network structure. In order to determine a suitable value, several experiments are conducted, and the results are used to find a middle ground value that does not impact the system behaviour extensively while moving around the resources often enough to deceive potential attackers.

The MTD mechanism is implemented in Python. It issues commands to the ONOS cluster and receives information from it such as current topology and active devices. The mechanism is tested on a virtual machine, the details are discussed in the next section.

6.2.3 Moving Target Defence Algorithm – Random

The algorithm above is modified to produce a new one that does not rely on groups but rather considers the whole cluster when making choices. The difference between the algorithms is that the second one will attempt to alternate between half of the available controllers randomly. This makes it even harder to analyse the structure of the system. The algorithm takes care to not choose the same master in subsequent choices. This is to have more controllers involved and to not over burden a single node.

Algorithm 2: MTD Randoms Pseudocode

```
INPUT   S (float), ControllersList (List)
1.      WHILE (TRUE)
2.          SLEEP (S SEC)
3.          FOREACH Device IN ControllersList DO:
4.              AssignDeviceToMaster(Device, RANDOM_Controller)
```

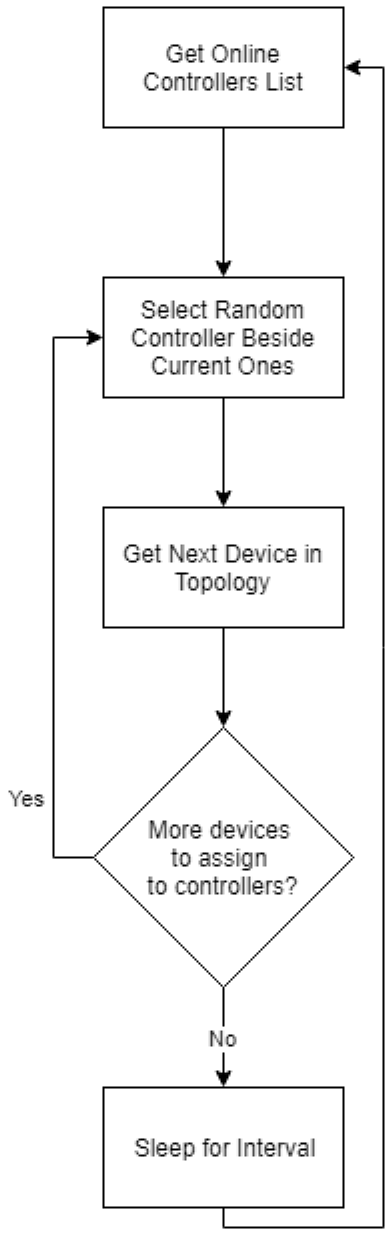

Get Online Controllers List
Select Random Controller Beside Current Ones
Get Next Device in Topology
More devices to assign to controllers?
Yes
No
Sleep for Interval

Figure 17 Algorithm 2 Flow Chart

Chapter 7

Results Discussion

The experiments are built on top of several platforms and systems. This includes a virtualization software, distributed SDN controllers, SDN network emulation system and other tools. Below is a highlight of the most important software that enabled this work.

7.1 Supporting Software

7.1.1 Virtualization Software

The experiment is built using a virtualization software. There are several virtualization software's that can be used to emulate operating systems. VirtualBox was used in this work; however, the same procedure can be done using VirtualBox or any other virtualization software.

In virtualization, there are two main entities, the host and guest machines. The host machine is the one running directly on the hardware while the guest machine is the one running inside the virtualization software. A guest machine can be configured with custom specifications as needed. For example, RAM can be set as 2 GB. This is of course subject to the hardware that is being used. A virtual machine cannot use more than the available resources. In this work, VirtualBox is used to run the required virtual machines.

7.1.2 Mininet

Mininet [48] is a network emulator which can be used to create a network of virtual hosts, switches, controllers, and links. Mininet hosts run a standard version of Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking. Mininet networks can run real code including standard Unix/Linux network applications as well as the real Linux kernel and network stack.

Mininet can be invoked through Python programs which is a powerful way to utilize it. It has a rich API available that allows the network topology to be designed and deployed easily for simulations. In this work, a topology is built using Mininet and then it is connected to the remote controller. This provides isolation of the components making it easier to debug and identify problems.

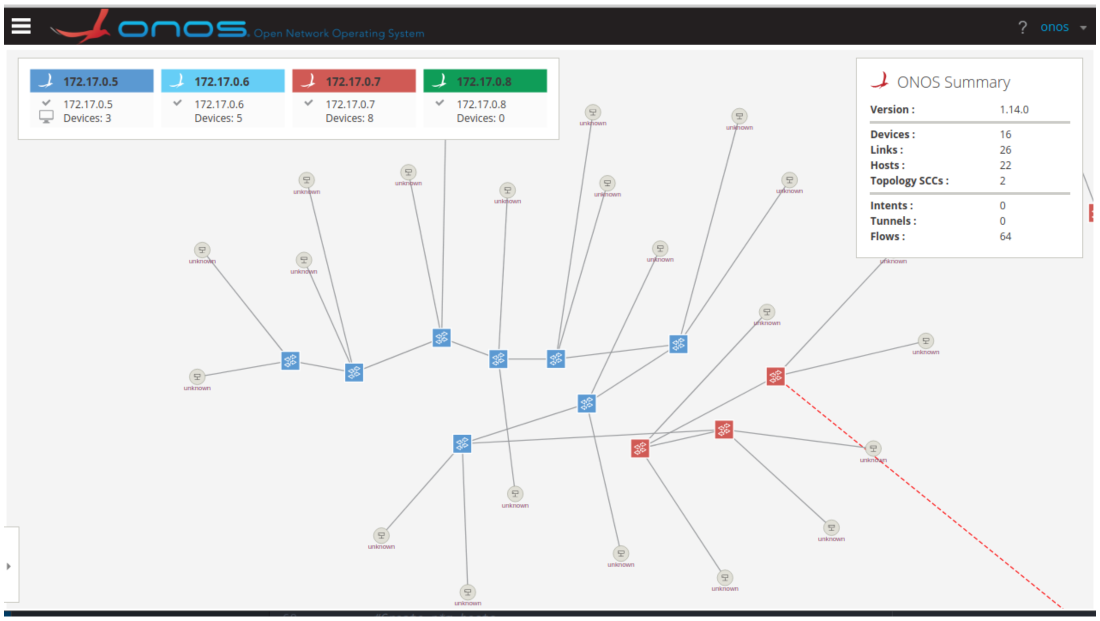
7.1.3 ONOS

In this work, the distributed controller network is built using the Open Network Operating System (ONOS) [19]. It supports OpenFlow but can be easily extended for other southbound protocols. ONOS is **a** high throughput, fault-tolerant and, more importantly, can automatically optimize itself after a fault. For this work, the controller will be used to implement a moving target defence mechanism. The controller is running on a virtual machine and connected to the Mininet environment for testing.

ONOS has a rich API that can be used to interact with it. It also allows developers to create applications and program them with Java then load them into the system. It has both a command line (CLI) and graphical user interface (GUI) that can be used to control it [19]. ONOS allocates the network devices into controllers that will act as masters for these devices. Each controller will have a subset of the network to manage. When a controller fails, its devices will be assigned to another controller. This boosts the network availability significantly.

7.1.4 Atomix

Atomix [19] is a framework that allows the construction of fault-tolerant and scalable distributed systems. It can manage clusters and detect failures and has rich JAVA and REST APIs. Atomix is used in conjunction with ONOS for data storage and coordination of clusters.



172.17.0.5	172.17.0.5 Devices: 3
172.17.0.6	172.17.0.6 Devices: 5
172.17.0.7	172.17.0.7 Devices: 8
172.17.0.8	172.17.0.8 Devices: 0

ONOS Summary

Version :	1.14.0
Devices :	16
Links :	26
Hosts :	22
Topology SCCs :	2
Intents :	0
Tunnels :	0
Flows :	64

Figure 18 ONOS Web UI

7.1.5 Architecture

The next figure shows the overall architecture of the system. ONOS clusters access Atomix clusters for data and coordination. They also control the network by installing flow rules into the OpenFlow switches active in Mininet. The MTD mechanism reads information from the ONOS clusters and makes decisions to alter the topology as needed. It does this by issuing commands to the ONOS clusters which in turn will adapt accordingly.

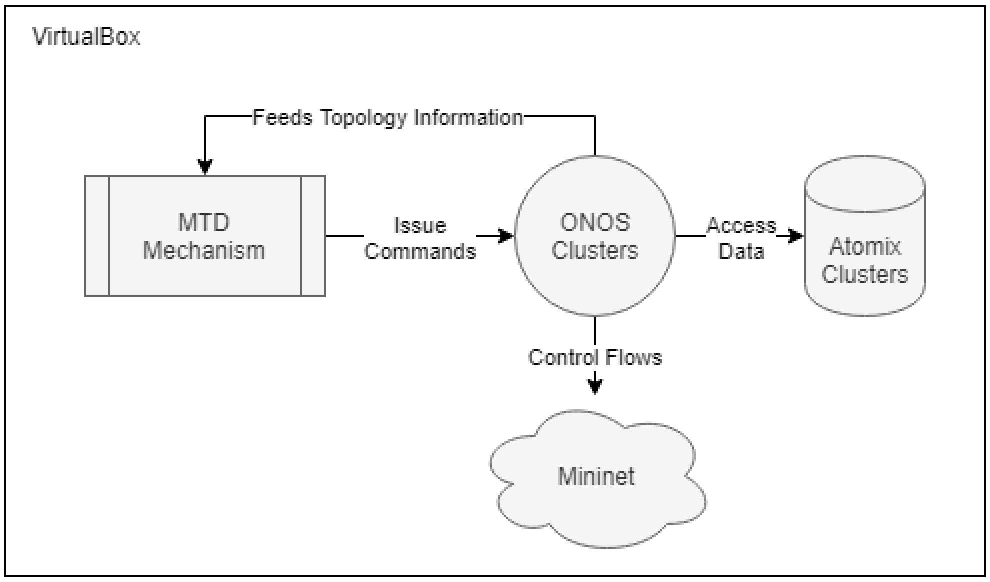


Figure 19 Overall System Architecture

7.2 Simulation Setup

In order to test the solution a computer is used that will contain several virtualized systems. These systems will interact with each other within the virtualized space. The computer uses an Intel i7-6700 CPU clocked at 4.00 GHz. It is loaded with a 16 GB RAM and the virtualized space is allocated 100 GB of drive space.

7.3 Experiment 1: Packet Loss

7.3.1 Network Topology

In this experiment, a network of linearly connected switches is built. Each switch connects two hosts yielding a topology like the figure below. The network is made of N switches and M end devices.

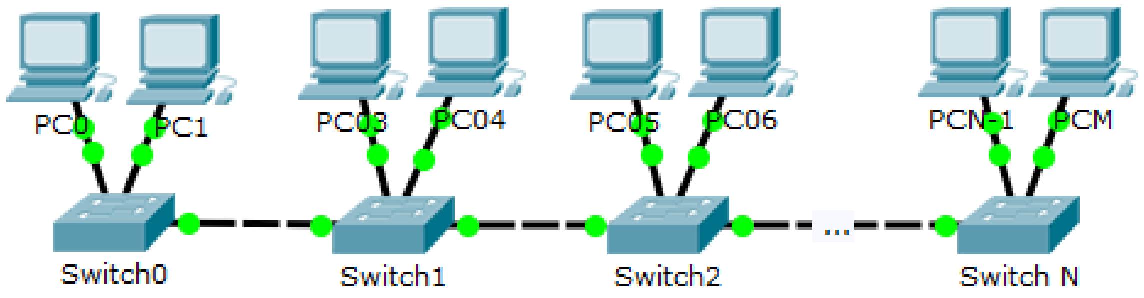


Figure 20 Network Topology

7.3.2 Controllers Setup

For this experiment, four distributed instances of ONOS will be used. In addition, three instances of Atomix are used for data storage and coordination of the controllers. The controllers’ topology would look like the figure below. Each instance of ONOS is connected to all three Atomix instances. It is possible to use more Atomix instances and/or ONOS nodes, but for the sake of simplicity and to make it easier to analyse the system the below is used.

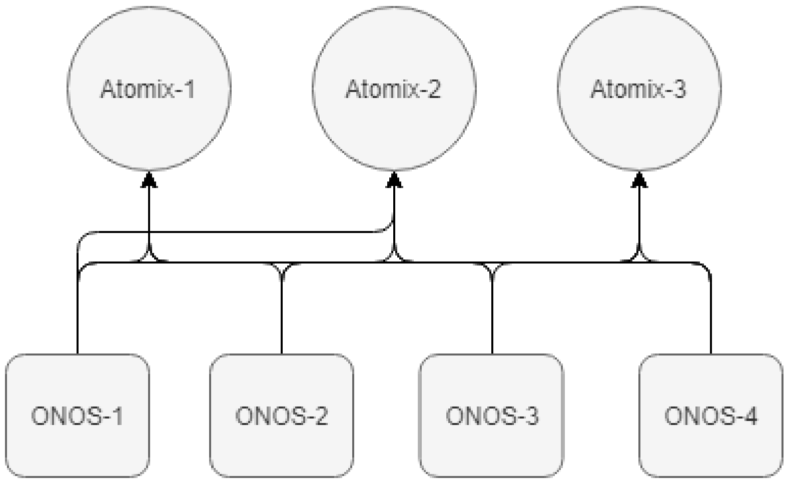


Figure 21 Controllers' Topology

Furthermore, for the MTD mechanism to operate, the ONOS nodes are divided into two groups as shown in the table. The mechanism alternates these groups between two states, active and inactive. During the active states, members of that group will be responsible of the network control while the other group goes into idle mode. The idle group will wait until the MTD mechanism wakes them and then will take over ownership of the nodes while the other group goes idle.

ONOS NODE	GROUP
ONOS-1	Group 1
ONOS-2	Group 2
ONOS-3	Group 1
ONOS-4	Group 2

Table 3 ONOS Groups

In this experiment, each host will send an ICMP packet to each other host in the network. The objective is to find out the average packet loss while the MTD security mechanism is active and compare it to normal system behaviour. The below results were gathered for different lengths of switching intervals. The experiment was repeated 15 times over 4 hours with each cycle taking approximately 16 minutes and the results were averaged to what is shown in the table Below. With 12 switches, we have a total of 24 hosts connected. The mechanism is tested over several switching intervals. The average packet loss without the mechanism is constant as it is the normal system behaviour. It can be observed from the results that the mechanism will perform poorly when the switch occurs frequently. However, as the time is increased, it stabilizes to an acceptable amount.

Number of Switches	Switch Interval (sec)	Average Packet Loss % (With MTD)	Average Packet Loss % (Without MTD)
12	5	0.229	0.012
12	6	0.024	0.012
12	7	0.036	0.012
12	8	0.036	0.012
12	9	0.024	0.012
12	10	0.024	0.012
12	15	0.012	0.012

Table 4 Packet Loss Percentage Comparison 1 – Algorithm 1

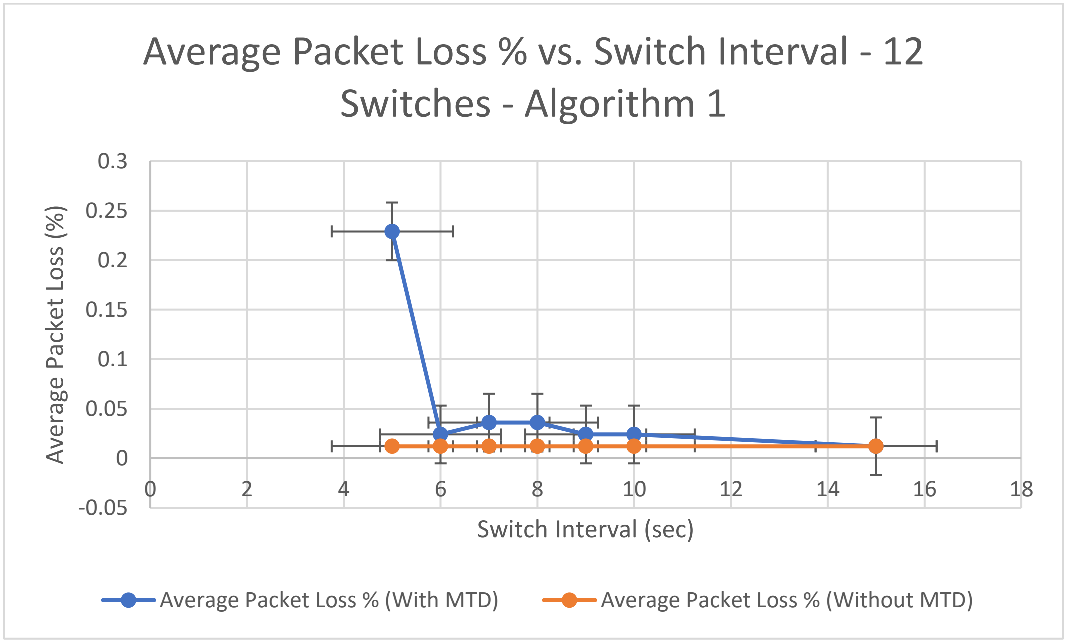


Figure 22 Average Packet Loss Percentage – 12 Switches – Algorithm 1

The curve shown in the figures depicting the average packet loss start with a high percentage due to the high number of changes in the network devices master controller. After the time is increased the number of dropped packets plummets substantially since the controllers will have more time to learn the network topology before changing the master controllers again.

The same experiment is performed using algorithm 2. The results in terms of packet loss does not vary that much from the first algorithm. However, the randomness introduced will make it harder to explore the current network and find a controller to target.

Number of Switches	Switch Interval (sec)	Average Packet Loss % (With MTD)	Average Packet Loss % (Without MTD)
12	7	0.174	0.042
12	9	0.100	0.042
12	11	0.085	0.042
12	13	0.074	0.042
12	15	0.074	0.042
12	17	0.060	0.042
12	19	0.065	0.042

Table 5 Packet Loss Percentage Comparison 1 – Algorithm 2

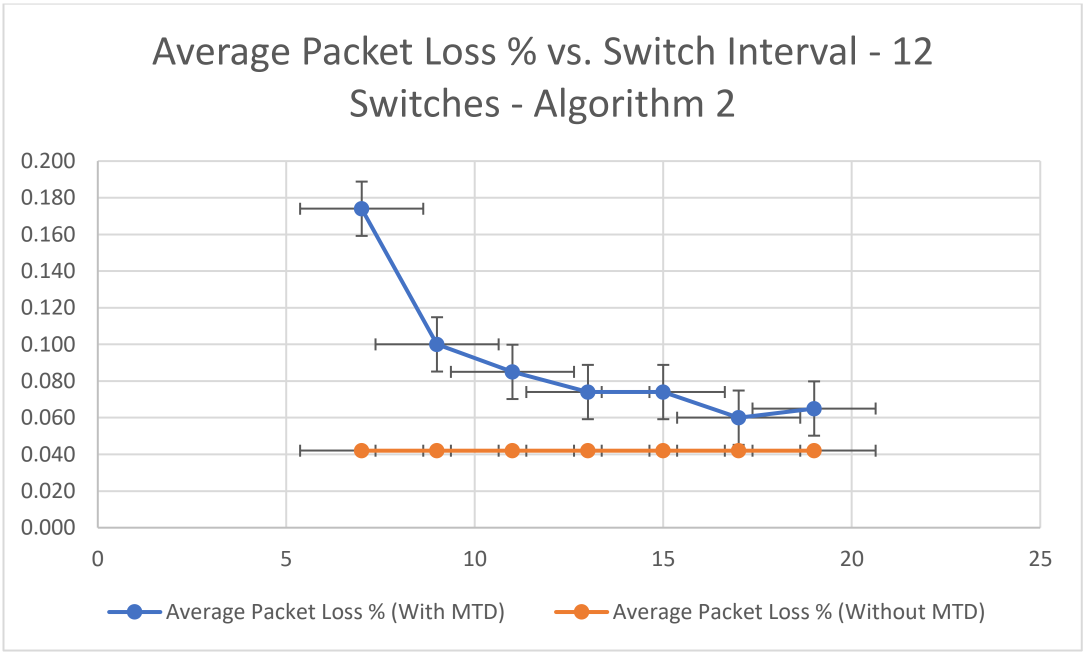


Figure 23 Average Packet Loss Percentage – 8 Switches – Algorithm 2

7.4 Experiment 2: CPU Utilization (Mechanism Overhead)

In this experiment the purpose is to find out the CPU usage of the mechanism before it is tested with an actual attack to measure its overhead. A topology similar to what was shown in the previous experiment is ran and the random algorithm is used to control the master nodes. The CPU utilization is captured for normal system behaviour (without the MTD tool running) and then its captured again while the tool is running.

The below result was captured for CPU utilization. It is important to note that this utilization is for the user processes and does not include the utilization of the operating system kernel that the experiment is running inside. This reduces the noise from other processes running on the kernel level and gives better perspective of tool CPU usage.

It can be observed that when the tool invokes the change on the nodes master, there is a spike in CPU utilization. Since this experiment was running with a 10 second sleep interval, a spike can be seen every 10 seconds in the graph.

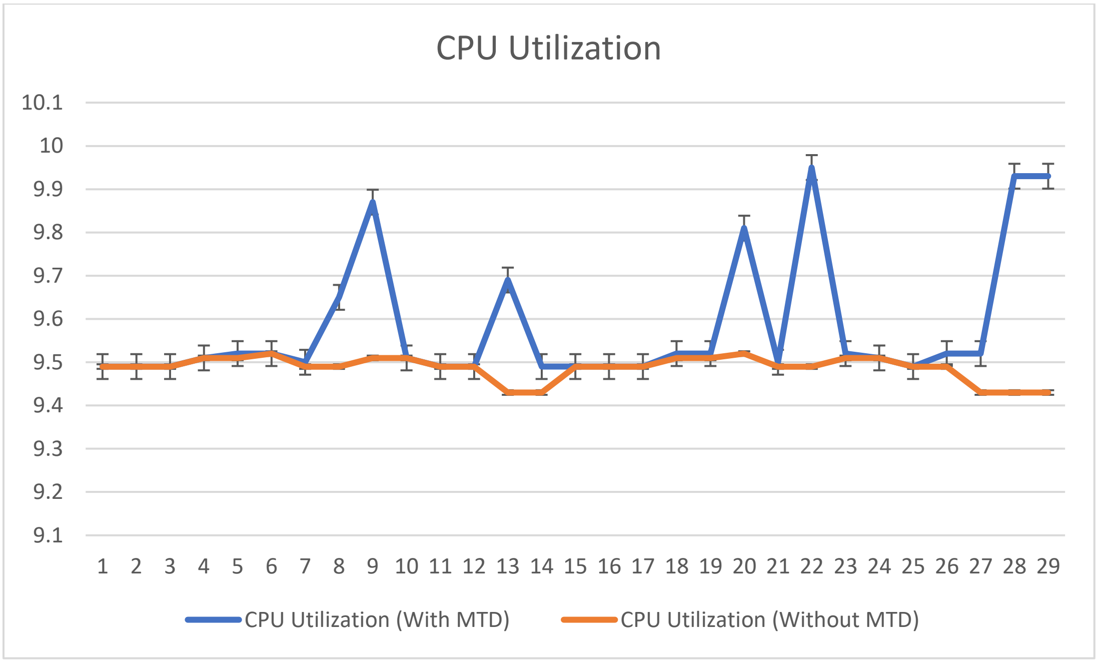


Figure 24 CPU Utilization

7.5 Network Bandwidth Consumption

With respect to the network overhead introduced by the mechanism, it is quite minimal. Consider the example of the 12 switches setup used in the previous experiments. In order to make the change between the master controllers, one packet is sent to each device from the newly appointed controller. This packet will direct the device to be governed by the now active controller while the previous master goes idle.

The figure below shows the relation between the number of packets sent in order to enable the mechanism to work. It can be observed that it is a meagre amount and will not impact the network bandwidth at all.

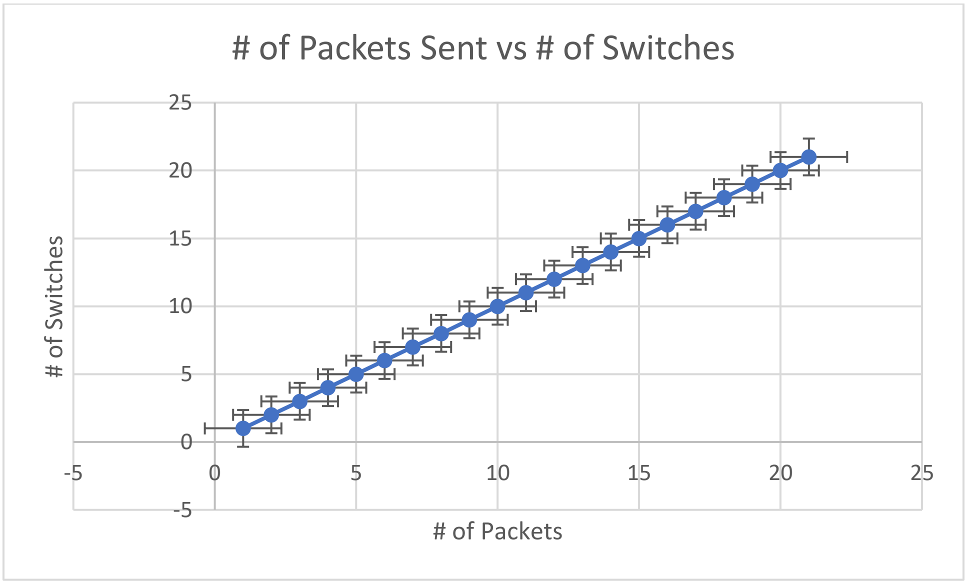


Figure 25 Network Bandwidth Overhead

7.6 Proposed Solution Validation

To validate the solution in this thesis, we reproduce the work shown by the authors of [38] and compare our results with it. The solution shown in the paper attempts to protect the network by introducing multiple controllers to the network. A flooding attack is launched on the network and the CPUs readings of the controllers are collected to show that the attack is being mitigated. The attack is made of three parts: the reconnaissance stage, the attack stage, the persistence stage.

The reconnaissance stage is an important step of the attack as it determines whether the attack can be launched or not. This is because the blind DDoS attack used is only effective against SDN-based networks. This stage also helps with determining the range of IP addresses to attempt when launching the attack so that it helps with optimizing the attack.

The attack stage is where the actual attack is launches to bring down the network. In this stage a huge number of packets is sent to the controller in order to bring it down. The paper simulates various types of packets attacks such as TCP, UDP, and ICMP. Here we replicate TCP based attacks for the sake of simplicity.

The last stage is the persistence stage. In this stage the attacker would attempt to persist something on the controller to facilitate future attacks. This could be a backdoor for example. In this thesis we do not focus on these types of attacks and the proposed solution does not attempt to protect against them thus the assumption is that no such attack will be made.

In the reproduced paper experiment, a randomly selected IP is attacked with packets of the size of 64 bytes. Four switches are attacked and It is assumed in the simulation that the attack flows sent respectively are A1, A2, A3 and A4, and attack flow rate is 200 Mbps×4 (e.g.A1 = A2 = A3 = A4 =200 Mbps).

The figures below show the reproduced results of the paper. It can be seen that with a single controller, the attack destroys the controller and brings its CPU capacity so high. However, when employing multiple controllers, the attack is unable to fully bring down the network.

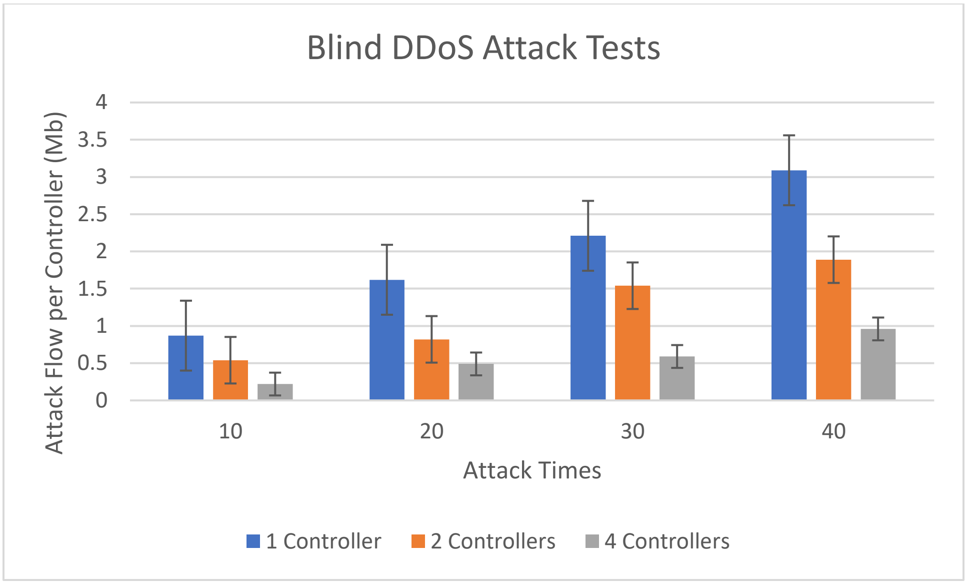


Figure 26 Blind DDoS Attack Tests (Reproduced Paper Results)

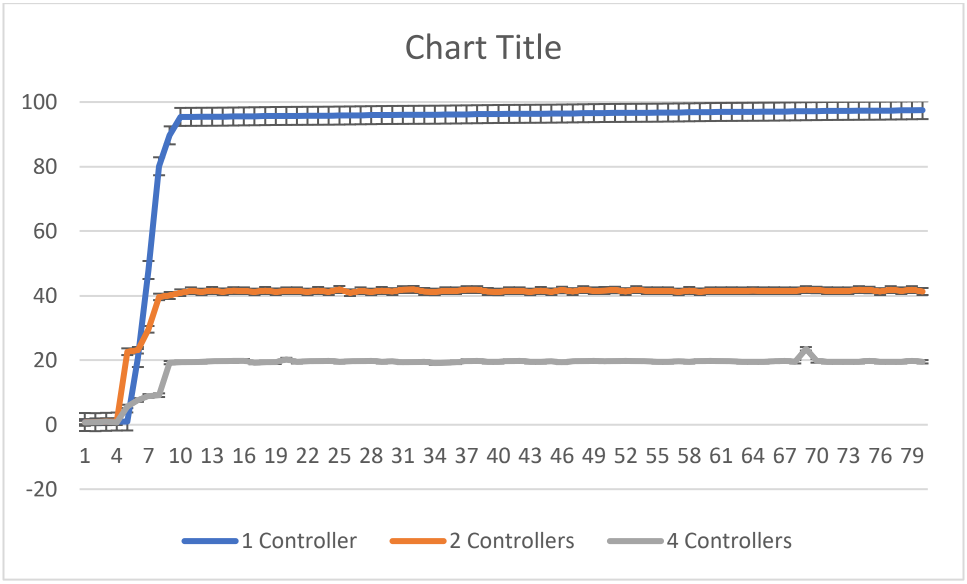


Figure 27 MTD Against Blind DDoS Attack (Reproduced Paper Results)

7.7 Proposed Solution Blind DDoS Attack

Next a blind DDoS attack is launched against the proposed solution setup. The same attack used in the reproduced paper is launched again. A randomly selected IP is attacked with packets of the size of 64 bytes. With similar parameters as the previously shown attack flows. In the proposed solution attack test however, the test is done on four controllers' setup. This is because the solution depends on having multiple controllers to operate.

After a first look at the results of the proposed solution the main difference can be observed. The proposed solution will have an individual CPU usage percentage higher than the solution in the reproduced paper. This is due to having fewer controllers working concurrently as the proposed solution makes use of idle controllers instead of having them all operate at the same time. However, the solution is still capable of operating the network on acceptable levels under the attack.

On average, the proposed solution CPU usage percentage is similar to the reproduced paper. The results below show that during idle time, the controllers will have low CPU consumption while the other active ones will take on the burden. As they keep alternating, the controllers will also alternate the burden of the attack instead of accumulating it over time.

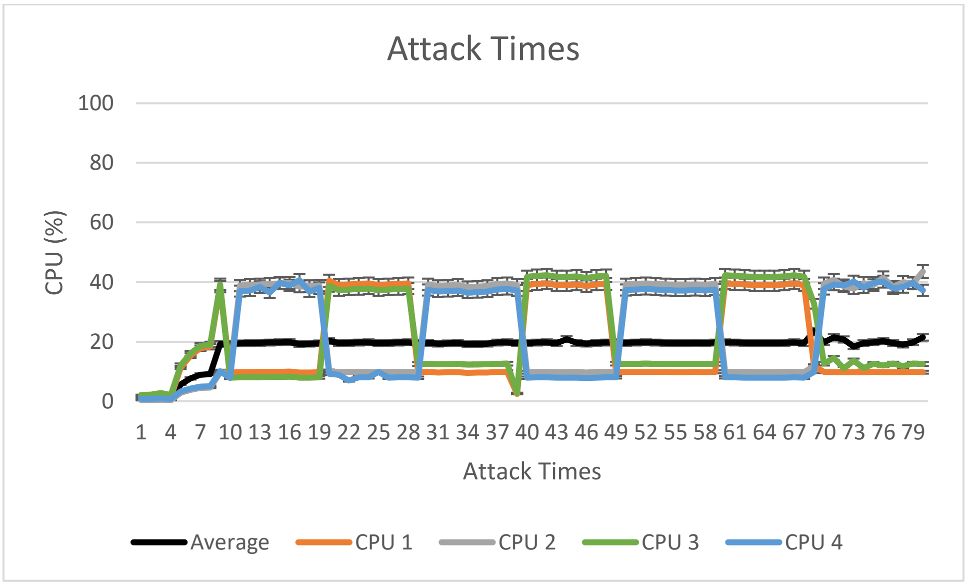


Figure 28 MTD Against Blind DDoS Attack

7.8 Proposed Solution Limitations

The solution provides protection against DDoS attacks by employing a distributed controller scheme. This dictates that the network resources should be able to handle the attack capacity.

The proposed solution will not be able to handle a vicious attack that is capable of bringing down all the controllers in the system. If an attacker is capable of sending a huge number of packets such that all the controllers in the system are overloaded or are brought down, then the network will be breached.

The solution proposed is unable to protect against persisted attacks. An example of a persisted attack is sending a malware that installs a backdoor in the network. In that case, the solution will not be able to protect against such an attack as it does not analyse the network traffic but simply relies on the number of packets sent in the network.

Furthermore, the solution aim is to protect the SDN controller rather than the network resources themselves. It is meant to be used in conjunction with other solutions that can protect the network resources such as routers and switches. Therefore, attacks that target the router or switches of the network will not be mitigated by the solution.

In addition, the solution is not capable of smartly balancing the load on the controllers. It simply chooses the devices assignment to controllers based on their numbers. If it is possible to assess the load incurred by the network devices, for example using the average number of new packets sent to a controller, it would enhance the solution considerably.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

The SDN paradigm is one of the developing platforms in the field. It offers a new way to control and configure the network without relying on the traditional de-centralized methods. By offering the separation of the data and control planes, it becomes easier to control the network. The SDN controller that lies at the heart of this paradigm serves a crucial role to enable this platform. This renders it a target for potential malicious entities.

The current works that were explored in this research focused on protecting the network assets excluding the controller. This gap gives an opening in the system which can be exploited to bring it down though a DDoS attack. In order to secure the system against this possible breach, the controller needs to be secured. It was shown that protecting the controller could incur an overhead on the system that needs to be minimized.

In this work, an SDN distributed controller system is secured through the employment of MTD. The distributed controller approach provides the network with the robustness it needs to survive DDoS attacks.

With respect to the objectives proposed at the beginning we managed to achieve the following:

- **Saving Network Bandwidth:** the implemented solution uses a minimal number of messages to achieve its purpose. This is shown in section 7.4 of the thesis.
- **Self-Configuration:** the implemented solution automatically adapts the network by re-assigning the devices to a controller after each cycle of sleep.

- **Security:** the implemented solution will be able to withstand DDoS attacks as long as not all of the controllers are brought down. The network will be able to operate as long as some of the distributed controllers are online.
- **Scalability:** the implemented solution is built using a distributed controller platform. By definition this makes it scalable. When the number of devices in the network increases, more controllers can be added to accommodate this change.

Several experiments were conducted to test the performance of the implemented approach. The main factors to test were how long does it take to discover the network topology through a connectivity test. Secondly, CPU impact induced by the mechanism is evaluated. It was shown that an acceptable system performance can be achieved while still securing the system against potential attackers. Experiments were conducted to verify the integrity of the mechanism introduced.

8.2 Future Work

For future work using a testbed rather than simulating the mechanism over virtual machines would show more accurate results. By nature, virtualization machines are limited by the capabilities of the host machine. Thus, to test this solution over a large network, a machine with great resources is needed. Having an actual testbed that can be configured as needed to simulate the network would allow for more accurate results.

Furthermore, this mechanism protects against attacks targeting the SDN controllers, if the data storage nodes (i.e. Atomix nodes) were brought down, the system will also collapse. Since ONOS relies on the Atomix nodes for storage and coordination, it makes the nodes a vital part of the system. Securing the Atomix nodes will provide a more robust system.

Another improvement that is possible on the proposed solution is to provide a load balancing mechanism. Instead of alternating the controllers based on the number of devices they control, assessing the controllers load expected to happen from the devices connected then making an informed decision on which devices to assign to which controller will enhance the solution.

References

[1] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine,* vol. 51, no. 2, pp. 114 - 119, 2013.

[2] K. Benzekki, A. E. Fergougui and A. E. Elalaoui, "Software-defined networking (SDN): a survey," *Security and Communication Networks,* p. 5803–5833, 2016.

[3] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Communications Magazine,* vol. 51, no. 7, pp. 36 - 43, 2013.

[4] L. Yang, R. Dantu, T. Anderson and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework," 2004.

[5] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart and A. Vahdat, "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network," 2015.

[6] I. S. M. Portal, "Open Systems Interconnection - Basic Reference Model: Naming and addressing," [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/s014258_ISO_IEC_7498-4_1989(E).zip. [Accessed 20 March 2019].

[7] I. O. f. Standardization, "ISO Standards Maintenance Portal," [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/s014258_ISO_IEC_7498-4_1989(E).zip. [Accessed 21 March 2019].

[8] A. Montazerolghaem, M. H. Yaghmaee and A. Leon-Garcia, "OpenSIP: Toward Software-Defined SIP Networking," *IEEE Transactions on Network and Service Management,* p. 184–199, 2017.

[9] D. B. Rawat and S. R. Reddy, "Software Defined Networking Architecture, Security and Energy Efficiency: A Survey," *IEEE Communications Surveys & Tutorials,* vol. 19, no. 1, pp. 325 - 346, 2016 .

[10] S. Gutz, A. Story, C. Schlesinger and N. Foster, “Splendid isolation: A slice abstraction for software-defined networks,” in *Proceedings of the first workshop on Hot topics in software defined networks*, Helsinki, Finland, 2012.

[11] C. Monsanto, J. Reich, N. Foster, J. Rexford and D. Walker, “Composing software-defined networks,” in *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, Berkeley, CA, USA, 2013.

[12] F. Hu, Q. Hao and K. Bao, “A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation,” *IEEE Communications Surveys & Tutorials,* vol. 16, no. 4, pp. 2181 - 2206, 2014.

[13] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, “Software-Defined Networking: A Comprehensive Survey,” *Proceedings of the IEEE,* vol. 103, no. 1, pp. 14 - 76, 2015.

[14] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado and R. Sherwood, “On Controller Performance in Software-Defined Networks,” in *Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, San Jose, 2012.

[15] D. Erickson, “The beacon openflow controller,” in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, Hong Kong, China, 2013.

[16] “Floodlight,” [Online]. Available: http://www.projectfloodlight.org/floodlight/.

[17] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhuy, R. Ramanathany, Y. Iwataz, H. Inouez, T. Hamaz and S. Shenker, “Onix: A Distributed Control Platform for Large-scale Production Networks,” in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, Berkeley, CA, USA, 2010.

[18] A. Tootoonchian and Y. Ganjali, “HyperFlow: A Distributed Control Plane for OpenFlow,” in *Proceedings of the internet network management conference on Research on enterprise networking*, San Jose, 2010.

[19] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow and G. Parulkar, “ONOS: towards an open, distributed SDN OS,” in *Proceedings of the third workshop on Hot topics in software defined networking*, Chicago, Illinois, USA, 2014.

[20] S. H. Yeganeh and Y. Ganjali, “Kandoo: A framework for efficient and scalable offloading of control applications,” in *1st Workshop Hot Topics Softw. Defined Netw.*, 2012.

[21] “OpenDaylight, A Linux Foundation Collaborative Project,” 2013. [Online]. Available: http://www.opendaylight.org.

[22] M. McCauley, “POX,” [Online]. Available: http://www.noxrepo.org/..

[23] N. T. a. T. Corporation, “RYU network operating system,” 2012. [Online]. Available: http://osrg.github.com/ryu/.

[24] G. Appenzeller, “SNAC,” 2011. [Online]. Available: http://www. openflowhub.org/display/Snac.

[25] Y. Takamiya and N. Karanatsios, “Trema OpenFlow controller framework,” 2012. [Online]. Available: https://github.com/trema/trema..

[26] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” *ACM SIGCOMM Computer Communication Review,* vol. 39, no. 2, pp. 69-74, 2008.

[27] “Open Networking Foundation,” [Online]. Available: https://www.opennetworking.org/sdn-resources/sdn-defined.

[28] C. Lei, T. Jinglei, J.-L. Tan, Y.-C. Zhang and X.-H. Liu, “Moving Target Defense Techniques: A Survey,” *Security and Communication Networks,* vol. 2, pp. 1-25, 2018.

[29] R. Zhuang, S. A. DeLoach and X. Ou, “Towards a Theory of Moving Target Defense,” in *Proceedings of the First ACM Workshop on Moving Target Defense*, Scottsdale, Arizona, USA, 2014 .

[30] P. Manadhata and J. M. Wing, “An Attack Surface Metric,” *IEEE Transactions on Software Engineering,* vol. 37, no. 3, pp. 371 - 386, 2010.

[31] A. Chowdhary, A. Alshamrani, D. Huang and H. Liang, “MTD Analysis and evaluation framework in Software Defined Network (MASON),” in *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, Tempe, AZ, USA, 2018 .

[32] R. L. Neupane, T. Neely, N. Chettri, M. Vassell, Y. Zhang, P. Calyam and R. Durairajan, “Dolus: Cyber Defense using Pretense against DDoS Attacks in Cloud Platforms,” in *Proceedings of the 19th International Conference on Distributed Computing and Networking*, Varanasi, India, 2018.

[33] T. OConnor, W. Enck, W. M. Petullo and A. Verma, “PivotWall: SDN-Based Information Flow Control,” in *Proceedings of the Symposium on SDN Research*, Los Angeles, CA, USA, 2018 .

[34] E. Al-Shaer, “Toward Network Configuration Randomization for Moving Target Defense,” in *Moving Target Defense*, New York, NY, Springer, 2011, pp. 153-159.

[35] J. Xu, P. Guo, M. Zhao, R. F. Erbacher, M. Zhu and P. Liu, “Comparing Different Moving Target Defense Techniques,” in *Proceedings of the First ACM Workshop on Moving Target Defense*, New York, NY, USA, 2014.

[36] J. H. Jafarian, E. Al-Shaer and Q. Duan, “Openflow random host mutation: transparent moving target defense using software defined networking,” in *Proceedings of the first workshop on Hot topics in software defined networks*, Helsinki, Finland, 2012.

[37] P. Kampanakis, H. Perros and T. Beyene, “SDN-based solutions for Moving Target Defense network protection,” in *World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Sydney, NSW, Australia, 2014.

[38] D. Ma, Z. Xu and D. Lin, “Defending Blind DDoS Attack on SDN Based on Moving Target Defense,” in *International Conference on Security and Privacy in Communication Systems*, 2015.

[39] A. Chowdhary, S. Pisharody and D. Huang, “SDN based Scalable MTD solution in Cloud Network,” in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, Vienna, Austria, 2016.

[40] J. Wang, F. Xiao, J. Huang, D. Zha, H. Hu and H. Zhan, “CHAOS: an SDN-based Moving Target Defense System,” *Security and Communication Networks,* vol. 2017, 2017.

[41] D. C. MacFarland and C. A. Shue, “The SDN Shuffle: Creating a Moving-Target Defense using Host-based Software-Defined Networking,” in *Proceedings of the Second ACM Workshop on Moving Target Defense*, Denver, Colorado, USA, 2015.

[42] J. Steinberger, B. Kuhnert, C. Dietzzy, L. Ball, A. Sperottoy, H. Baier, A. Prasy and G. Dreo, “DDoS Defense using MTD and SDN,” in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, Taipei, Taiwan, 2018.

[43] S.-Y. Chang, Y. Park and B. B. A. Babu, “Fast IP Hopping Randomization to Secure Hop-by-Hop Access in SDN,” *IEEE Transactions on Network and Service Management,* pp. 1-1, 2018.

[44] Y. Wang, J. Yi, J. Guo, Y. Qiao, M. Qi and Q. Chen, “A Semistructured Random Identifier Protocol for Anonymous Communication in SDN Network,” *Security and Communication Networks,* 2018.

[45] G. Zeyu, Z. Xingming and M. Qing, “MDSA: Security Scheduling Mechanism for a Reliable SDN Control Layer Based on Mimic Defense,” in *Recent Developments in Intelligent Computing, Communication and Devices*, Singapore, 2018.

[46] F. Gillani, E. Al-Shaer and Q. Duan, “In-design Resilient SDN Control Plane and Elastic Forwarding Against Aggressive DDoS Attacks,” in *5th ACM Workshop on Moving Target Defense*, Toronto, 2018.

[47] A. Chowdhary, S. Sengupta, A. Alshamrani, D. Huang and A. Sabur, “Adaptive MTD Security using Markov Game Modeling,” in *International Conference on Computing, Networking and Communications (ICNC)*, Honolulu, 2018.

[48] B. Lantz and B. O’Connor, “Mininet,” 2018. [Online]. Available: http://mininet.org/.

[49] A. Tootoonchian and Y. Ganjali, “HyperFlow: A distributed control plane for OpenFlow,” in *Internet Netw.Manage. Conf. Res. Enterprise Netw.*, 2010.

Vitae

Name: Ahmed Salman Hasan Al Naser

Nationality: Saudi

Date of Birth: 6/23/1992

Email: ahmed.alnaser2007@gmail.com

Address: Al-Awjam, Eastern Province, Saudi Arabia, P.O 18007

Academic Background: Ahmad Al-Nasser is currently a KFUPM master student enrolled in the Computer Networks program. He earned his BSc in Information & Computer Science from the Computer Science Department, at KFUPM. His research interests include Fog Computing and Software-Defined Networking. Published a paper titled "Implementation of a hybrid wind-solar desalination plant from an Internet of Things (IoT) perspective on a network simulation tool" in the Applied Computing and Informatics journal.