Impact of Network Security on the SDN Controller Performance

By

Carlton Kodzai

Submitted to the Department of Electrical Engineering,

University of Cape Town, in partial fulfilment of the requirements

for the degree

Master of Engineering

Faculty of Engineering and the Built Environment

UNIVERSITY OF CAPE TOWN

Supervisor:

Dr Joyce Mwangama

15 October 2020

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science in Engineering in the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

Author ……***Carlton Kodzai***

Signed by candidate

Acknowledgement

I would like to thank Dr Joyce Mwangama my supervisor for the support, great feedback and providing me with direction on research areas of this dissertation. It was a pleasant experience working with her and I'm grateful for the time and opportunity she gave me.

I would like to thank my family for giving me support and encouragement throughout the research.

I would like to thank the Lord God for giving me the strength and wisdom for me to complete this assignment.

Abstract

Internet Protocol network architectures are gradually evolving from legacy flat networks to new modern software defined networking approaches. This evolution is crucial as it provides the ideal supporting network structure, architecture and framework that supports the technologies that are also evolving in software-based systems like Network Functions Virtualization (NFV). The connectivity requirements resulting from this paradigm shift in technology is being driven by new bandwidth requirements emanating from the huge number of new use cases from 5G networks and Internet of things (IoT) future technologies. Network security remains a key critical requirement of these new modern network architectures to deliver a highly available, reliable service and guaranteed quality of service. Unprotected networks will usually experience service interruptions and cases of system non-availability due to network attacks such as denial-of services and virus attacks which can render key network components unusable or totally unavailable.

With the centralized approach of the Software Defined Networking architecture, the SDN controller becomes a key network point that is susceptible to internal and external attacks from hackers and many forms of network breaches. It being the heart of the SDN network makes it a single point of failure and it is crucial that the security of the controller is guaranteed to avoid unnecessary irrecoverable loss of valuable production time, data and money.

The SDN controller design should be guided by a robust security policy framework with a very sound remedy and business continuity plan in the event of any form of a security attack. Security designs and research work in SDN controllers have been done with focus on achieving the most reliable and scalable platforms through self-healing and replication processes.

In this dissertation the research that was done proposed a security solution for the SDN controller and evaluated the impact of the security solution on the overall SDN controller performance. As part of the research work literature review of the SDN controller and related technology carried out. The SDN controller interfaces were analyzed and the security threats that attack interfaces were explored. With link to a robust security framework a security solution was used in the experiments that analyzed the attacks from the external network sources which focused on securing the southbound interface by use of a netfilter with iptables firewall on the SDN controller. The SDN controller was subjected to denial service attack packets and the impact of the mitigation action observed on the SDN controller resources. Given that the network security layer introduced an additional overhead on the SDN controller's processors the security feature negatively affected the controller performance. The impact of the security overhead will inform on the future designs and possibly achieve a trade-off point between the level of security of the network and overall system performance due to security policies. The research analyzed and determined the performance impact of this crucial design aspect and how the additional loading due to network security affected the SDN controller normal operation.

Declaration..... i

Acknowledgements..... ii

Abstract.....iii

Table of contents..... iv

List of Figures..... v

List of Tables.....vi

LIST OF ACRONYMS and ABBREVIATIONS vii

Table of Contents

CHAPTER 1 1

Introduction.....1

1.1 Background 1

1.2 Problem Description 2

1.3 Aims and Objectives.....3

1.4 Scope of works.....4

1.5 Chapter Outline 5

CHAPTER 2 8

2 Literature Review.....8

2.1.1 Chapter Overview 8

2.2 Overview of Software Defined Networking (SDN).....8

2.2.1 SDN Architecture 8

2.2.2 SDN Controller..... 10

2.2.3 SDN Open Virtual Switches (OVS)..... 10

2.2.4 SDN OpenFlow Protocol..... 11

2.2.5 OpenDaylight Controller (ODL)..... 11

2.2.6 OpenDaylight Controller Architecture 12

2.3 SDN Security Overview.....12

2.3.1 SDN security threat vectors 13

2.3.2 Control - Data plane attacks 13

2.3.3 DDoS based attacks 14

2.3.4 Application layer attacks 16

2.4 Review of SDN security solutions 17

2.4.1 Linux based Network Security 20

2.4.2 SDN Network Security Design 21
2.4.3 Iptables firewall design principles 21
2.5 Chapter Summary 22
3 CHAPTER 3 METHODOLOGY 23
3.1 Chapter Overview 23
3.2 Introduction 23
3.3 SDN Controller security solution requirements 23
3.3.1 Proposed solution functional requirements 24
3.3.2 Analysis of Functional Requirements 25
3.4 Proposed solution Performance requirements 25
3.4.1 Central processing unit utilization 26
3.4.2 Memory utilization 26
3.4.3 Response times 26
3.4.4 SDN application latency 26
3.4.5 Analysis of Performance Requirements 26
3.5 SDN controller and packet forwarders deployment 27
3.6 Iptables and Netfilter deployment framework 28
3.7 DDoS Source Deployment 29
3.8 Resource monitoring module (SNMP server) 29
3.9 Tools and Technologies 30
3.9.1 Controller plane simulations design considerations 30
3.9.2 OpenDaylight 31
3.9.3 Mininet 32
3.9.4 OpenFlow Switches 32
3.9.5 Windows Hyper-V 33
3.9.6 Ubuntu 16.04 33
3.9.7 Wireshark packet analyzer 34
3.9.8 OpenNMS NMS tool 34
3.9.9 DDoS attack tools 34
3.9.10 APIs and Protocols 35
3.9.11 Mininet python API 35
3.10 Chapter Summary 35
4 CHAPTER 4 36
4.1 Chapter Overview 36
4.2 Introduction 36
4.3 Testbed Setup and Design 36
4.3.1 Hardware and physical layout design 37
4.3.2 VMs logical network design 38

4.3.3 Data plane design 39
4.3.4 Control plane design 39
4.3.5 DDoS traffic source design 40
4.4 Iptables based Firewall implementation 41
4.4.1 Application layer call flows 42
4.5 Chapter summary 44
5 CHAPTER FIVE 45
EXPERIMENTAL PERFORMANCE EVALUATION AND VALIDATION 45
5.1 Chapter Overview 45
5.2 Experimental Evaluation Scenarios 45
5.2.1 DDoS attack launch and verification 45
5.2.2 DDoS mitigation functionality verification and measurements 47
5.3 Iptables effect on ODL performance measurements 48
5.3.1 ODL performance benchmark testing 48
5.3.2 Benchmarking experiment results 51
5.3.3 Analysis of results (Scenario 1) 53
5.4 Iptables effect on ODL controller measurement (No DDoS) 53
5.4.1 Results of the experiment 54
5.4.2 Analysis of results 56
5.5 DDoS mitigation on ODL performance load testing and measurement 57
5.5.1 Analysis of DDoS mitigation impact on ODL CPU, Memory and response rate 59
5.5.2 Impact of iptables firewall on SDN network southbound API Latency results ... 59
5.6 Chapter Summary 61
6 CHAPTER 6 63
CONCLUSION 63
6.1 Introduction 63
6.2 Conclusion 63
6.3 Future work 65
6.3.1 DDoS Detection algorithms 65
6.3.2 Packet forwarder Network size 65
6.4 Chapter Summary 65
7 References 66
8 Appendix A: Experiment 72

List of Figures

Figure 1.1- *Growth trends for DDoS attacks* [2] 2

Figure 1.2 - *Block diagram for design components*…….. . 4

Figure 2.1- *SDN layered architecture* [7] 9

Figure 2.2 *OpenFlow message flow* [14] 11

Figure 2.3 *OpenDaylight architecture [3]* 12

Figure 2.4 *Summary of threat vectors* [7]..... 13

Figure 2.5 *DDoS attacks categories* [29] 15

Figure 2.6 *TCP/IP 3-way handshake process* [30]..... 16

Figure 2.7 *Distributed SDN controller architecture* [8]..... 19

Figure 2.8 *High level for proposed solution* 21

Figure 2.9 *Iptables blueprint and logic schema*..... 22

Figure 3.1 *Hyper-V virtual machine deployment framework*..... 27

Figure 3.2 *Iptables firewall rules flow*……..28

Figure 3.3 *SNMP server placement in the SDN architecture* 30

Figure 4.1 *OpenDaylight layout* [17] 33

Figure 4.2 *OpenDaylight architecture for the Beryllium version used in the PoC* 35

Figure 4.3 *Hardware and physical layout design setup*..... 37

Figure 4.4 *Proposed logical network design* 38

Figure 4.5 *SDN network created as viewed in ODL Dlux portal*..... 40

Figure 4.6 *Application call flows* 43

Figure 5.1 *hping3 launch commands for DDoS attack initiation*..... 46

Figure 5.2 *Wireshark capture showing DDoS attack initiated from VM4 on flooding port 80 on 10.254.0.1 with SYN packets*..... 46

Figure 5.3 *Traffic statistics on VM1 showing DDoS traffic source on eth0* 47

Figure 5.4 *Traffic statistics on vM4 showing traffic DDoS traffic*..... 47

Figure 5.5 *Wireshark captures showing DDoS mitigation* 48

Figure 5.6 *Traffic volumes captured on VM4 during the attack from VM1* 48
Figure 5.7 *SNMP status on VM1* 49
Figure 5.8 *Output from VM1 showing OpenDaylight application running* 49
Figure 5.9 *Mininet launch parameters* 50
Figure 5.10 *dump command results* 50
Figure 5.11 *links command results* 50
Figure 5.12 *net* command results 50
Figure 5.13 *Pingall command results* 51
Figure 5.14 *CPU usage statistics on ODL controller host for experiment 5.3.1* 51
Figure 5.15 *Memory utilization of ODL host VM1 for experiment 5.3.1* 52
Figure 5.16 *Response times for the ODL host for experiment 5.3.1* 52
Figure 5.17 *Latency in milliseconds for Mininet SDN network setup* 53
Figure 5.18 *Iptables policy rules table* 53
Figure 5.19 *CPU extracted with iptables turned ON experiment 2* 55
Figure 5.20 *Memory utilisation experiment 2* 55
Figure 5.21 *Response times for experiment 2* 55
Figure 5.22 *OpenDaylight memory utilization* [48] 56
Figure 5.23 *Impact of number of switches on CPU* [45] 56
Figure 5.24 *Latency comparison for the same number of switches* 57
Figure 5.25 *CPU usage with DDoS active and mitigation active* 58
Figure 5.26 *Memory utilization with DDoS active and firewall online* 58
Figure 5.27 *Response times with DDoS active and firewall online* 58
Figure 5.28 *Southbound API latency* 59
Figure 5.29 *Firewall impact on latency assessment* 60
Figure 5.30 A*verage system load KPI for the SDN controller* 60
Figure 5.31 *TCP errors and failures during mitigation* 61
Figure 8.1 *Installation path for ODL software* 73

Figure 8.2 *ODL GUI topology view* 74

Figure 8.3 *Commands for activating iptables rules* 74

Figure 8.4 *Iptables rules activation commands* 75

Figure 8.5 *OpenNMS portal available on http://10.254.0.3:8980/* 76

Figure 8.6 *SNMP agent configuration in VM1* 76

Figure 8.7 *Access control-list for SNMP server authorisation* 77

Figure 8.8 *Hyper-V host configuration portal* 78

Figure 8.9 *Hyper-V virtual switch settings* 78

Figure 8.10 *Test between VM1 and VM2* 79

Figure 8.11 *Test between VM1 and VM3* 79

Figure 8.12 *Test between VM1 and VM4* 80

List of tables

Table 2-1 *Summary of Security attack types [2]* 17

Table 4-1 *Experimental setup virtual machine roles* 35

Table 4-2 *Mininet launch parameters* 39

Table 4-3 *Ingress traffic rules design guidelines* 41

Table 4-4 *Egress traffic rules design guidelines* 42

Table 5-1 *Service tests for traffic traversing the iptables firewall* 54

Table 8-1 *Hardware specifications used in the experiments* 72

LIST OF ACRONYMS and ABBREVIATIONS

API	Application Programming Interface
CPU	Central Processing Unit
DDoS	Distributed Denial of Service
DMZ	Demilitarised Zone
DPI	Deep Packet Inspection
GUI	Graphical User interface
IDS	Intrusion Detection System
IP	Internet Protocol
IPS	Intrusion Prevention System
MAC	Media Access Control
NETCONF	Network Configuration Protocol
NFV	Network Functions virtualisation
ODL	OpenDaylight
OSI	Open Systems Interconnection
OVS	Open Virtual Switch
SDN	Software Defined Networking
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TLS	Transport layer security
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network

CHAPTER 1

Introduction

1.1 Background

Software Defined Networking (SDN) provides an appropriate platform and architecture to efficiently support new dynamic connectivity requirements presented in upcoming new use cases and opportunities in the telecommunication industry. It provides a centralized view of the Internet Protocol network and ensures that all network devices in the network are managed from one central point by a controller giving a global network view. SDN comes with several benefits to the networking technology including improved latency between nodes, programmability of the network, simplified network provisioning, improved load balancing, hardware savings and simplified centralized administration of the network layout [1]. These benefits realized from the SDN deployments result in overall improved network performance through automation of configurations and easy provisioning of new services which lowers the overall operating costs for the service providers [2].

Due to the continuous growth in connectivity requirements and new use cases the demand to have an elastic network that is dynamically configurable becomes inevitable. Current research has shown huge potential growth opportunities towards automation in the modern requirements such as smart cities, smart homes, e-health, energy, modern construction and Industry 4.0 through machine to machine and Internet of things (IoT) capable devices [1]. By inspection the above new services demand a highly dynamic, automated and stable flexible network. The SDN architecture must support highly scalable and secure networks through interoperating with 5G based core networks [1]. The SDN network must be highly secure and reliable to avoid any downtimes as this can lead to serious catastrophic incidents and huge revenue losses. As the SDN architecture is programmable it presents a double-edged sword and juxtaposition scenario as it exposes the security weakness of the software-based systems due to software bugs and modern hacking [3]. Engineers can install security applications on the controller northbound interface which involves exposing this interface to potential attacks. The SDN controller being the core, central node in the architecture becomes a node of interest that is highly targeted by many hackers. Network attacks have been on the rise with the global ransomware attacks in 2017 costing many firms such as US pharmaceutical giant Merck a revenue loss of up to 300 million dollars through database extortion, email blocking and computer access locking [4].

Figure 1.1 shows the growing trends of distributed denial of service (DDoS) external attacks, a highly probable security threat to SDN controller in operator core networks. These security attacks also take away the available bandwidth intended for legitimate users who requires to access the network resources and further impacts on the network operator revenue by taking away productive time. Security deployments in SDN architectures will require an end to end approach as security should be designed to cover all the planes of the SDN architecture. To exhaustively secure the SDN controller we need to have a broad security framework that

guides and covers all the activities to be carried out in prevention or mitigation in the event of such a security breach. Such response activities will broadly cover the attack detection, mitigation mechanisms and recovery strategies.

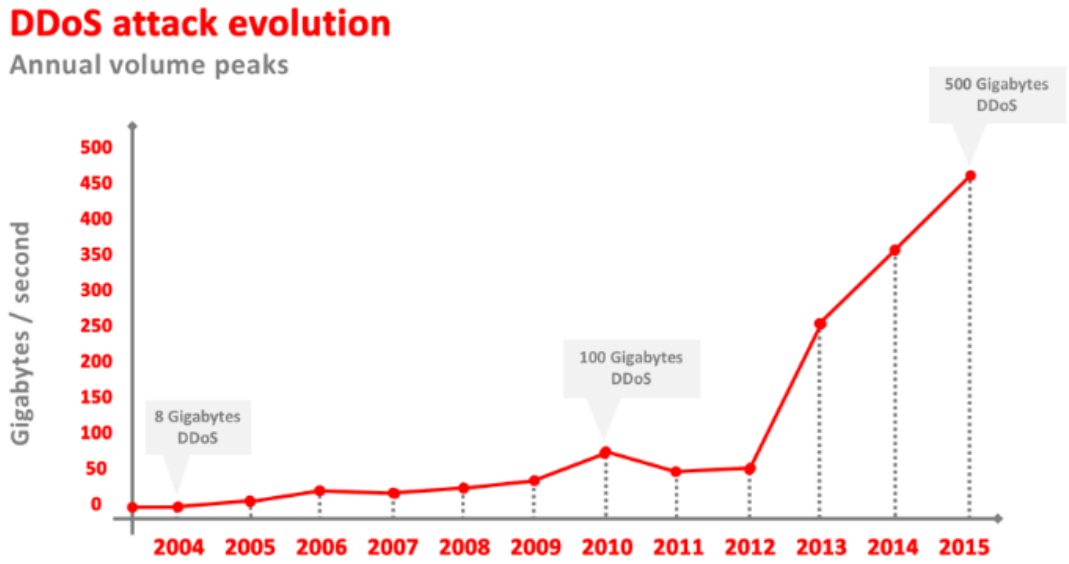


Figure 0.1- *Growth trends for DDoS attacks* **[2]**

The security framework in the SDN architecture will span from hardware or software perimeter security of the controller up to software related risks as external hackers will utilize any available weak entry point to initiate a breach. In legacy networks perimeter security is implemented and enforced through the deployment of physical firewalls that utilize policies with block or allow actions to various traffic types that traverses the network. The framework is now different in SDN based architectures as there are now 3 different planes that have many configuration changes due to continuous changing network conditions [2]. The risk conditions in SDN architectures have escalated and mutated spanning from hardware sources up to software related risks which requires a totally different strategy to counter the risk.

1.2 Problem Description

The SDN controller remains a core component of any SDN based ecosystem responsible for the decision making and handling of all unknown traffic packets detected in the packet forwarding plane. It will contain the security policies that in turn are expected to protect itself (SDN controller) from attacks. Securing the controller will start from the SDN system design and how the controller is secured as it interoperates with other network elements in the end to end IP network solution. The SDN controller system should be designed with performance, load, scalability and security-based considerations. The design should strike a balance between the system performance and the desired security. The balance is required as some security policies will affect the system performance by introducing latency as they require more processing resources from the elements implementing the security solution. In traditional networks we have seen that security was mostly implemented via network

firewalls with Intrusion detection or prevention modules which had limitations of failing to block some application-based threats such as distributed denial of service (DDoS).
Cyber-attacks can also pose security threats to the SDN controller applications in the form of software bugs and malware (viruses, trojans and worms) [5]. Solutions like Deep Packet Inspection (DPI) are available to intercept and mitigate the cyber -attack threats to a certain extend. The DPI mitigation techniques involve the detection and analysis of the intruder application signatures in the packet data segments flows. The DPI engine will then block the suspected malicious traffic based on these detected abnormal signatures. DPI solutions rely on correct protocol signatures to correctly detect malicious traffic which can be bypassed by encryption in some cases. The ongoing challenge in DPI-based solutions becomes the continuous updating of its protocol list databases on newly blacklisted signatures picked from new applications released on the internet. The solutions briefly discussed above will not implement all security features and will be inadequate to defend the SDN controller against other security threats like spoofing, sniffer attacks and man in the middle attacks. The solutions discussed by many authors tackle and recommend the network security by using dedicated standalone firewalls which can be costly to deploy as standalone appliances. Other security solutions that use protocol signatures can be bypassed through installation of virtual private networks and packet encryption which provides shielding to traffic inspection from various security inspection technologies. With these limitations of current security designs, a strong cost-effective security perimeter design is essential in the rollout of all SDN controller topologies to have guaranteed uptime of the controller. In this research we implement an application-based security solution using opensource Linux iptables netfilter and to determine the impact of this security implementation on the SDN controller system performance**.**

1.3 Aims and Objectives

The research involved the implementation of a Linux based network security solution on the SDN controller southbound interface and the evaluation of the impact of the additional processing load to the SDN controller performance. The research capitalised on the possibility of the integration of the SDN controller deployed in Linux operating system environment with a Linux based firewall as an application on the same host. The design approach attempted to counter the DDoS security threat as highlighted in the problem statement and keep the attack outside the controller by providing a secure ring fence around the SDN controller. The perimeter-based proposed solution addressed the inherent limitations of firewall deployments and analyzed the traffic packets at the application layer. The security solution coexisted with the SDN controller and resides on the same infrastructure with the controller to reduce deployment costs. The impact of the added security feature on the overall system SDN controller performance was analyzed. The design solution implemented a perimeter security and removed the need of a standalone firewall solution which is external to the SDN controller as proposed by other authors in [8]. In order to accomplish the aims of the research work, the following objectives were realized:

- Review of security threats, security solutions and their limitations in current designs that affect the SDN controller operation.

- Design and implementation of a network security/firewall solution using Linux iptables to secure the SDN controller by blocking DDoS traffic.
- Modelling of the security policies in Linux iptables that inspects traffic and provide security required on the SDN controller against DDoS attacks.
- Analyzing the effect of increasing the number of policies on the controller to overall network throughput and device processing capability.
- Determined the impact of the network security deployment on the SDN controller's system resources utilization through analyzing the central processing unit, memory of the machine hosting the controller, response rate and southbound interface latency.

1.4 Scope of works

The proposed overall design will involve setting up of four critical components and modules used in the simulation testing and analysis of results. The core components are namely the SDN external controller, iptables network firewall, packet forwarding network in Mininet and network monitoring system as shown in Figure 1.2.

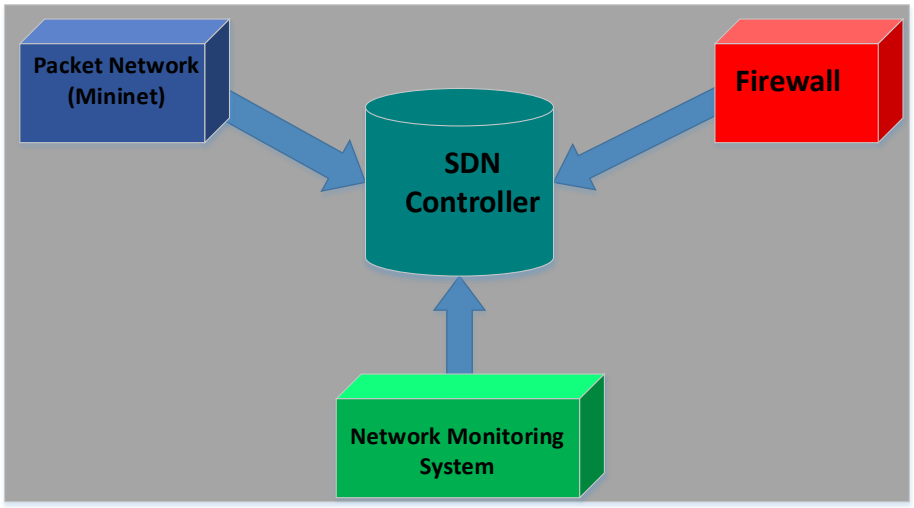


Figure 0.2 - *Block diagram for design components*

The dissertation will focus on setting up the SDN controller using the OpenDaylight project and connect the controller to the packet forwarding plane using Mininet Open Virtual Switches (OVS). The choice and shortlisting of OpenDaylight as the preferred SDN controller was based on OpenDaylight being the most preferred industry standard, open platform with easy modularisation and customisation [6]. The controller is designed to support and adapt to small/large networks covering several use cases making it an applicable controller close to the real-life traffic scenarios. The controller is an open source platform with support available on OpenDaylight forums and many online sources [7]. OpenDaylight provides other advantages with capability of detecting the topology in Mininet. Using its inbuilt modules, it can prevent network loops for meshed networks which can potentially cause performance issues in the simulations. The SDN controller uses a kernel-based machine which is opensource, web-based user management and it has a huge community for support [8]. The OpenDaylight controller

and Mininet packet forwarding segments will be deployed on separate virtual machines and interconnected by virtual switches in Microsoft hypervisor tool. The dissertation will cover deployment of Linux Iptables policy rules on the same host as the SDN controller to provide the network security layer the for the controller. The security threat is simulated by use of hping3 tool as the DDoS traffic source. For capturing the results, we will utilise OpenNMS monitoring tool that will monitor the SDN controller using its inbuilt monitoring features to capture the CPU utilisation, memory utilization and other key performances parameters. The dissertation simulations are based on computer-based software simulations that are running on multiple opensource software. The opensource software includes the hypervisor, Linux base software, OpenDaylight controller, OpenNMS, DDoS source and Mininet. All the modules will interoperate to produce the desired scenarios on the independent virtual machines required in the simulations to meet all the objectives. Due to the nature of the computer-based simulations being done on the virtual machines the dissertation will assume the following:

- That the OpenDaylight runs the same way with no defects on any of the Linux base operating system e.g. Ubuntu, Fedora, CentOS or FreeBSD.
- That version of OpenDaylight bin/karaf package chosen will vary with tests results and more stable versions available will be used for the simulations.
- That deployment and testing will be restricted to one SDN controller due to limited computing resources on the testing environment that will be hosting the virtual machines.
- Test results extracted from network monitoring tool can be assumed accurate.

1.5 Chapter Outline

This chapter gives the background of how the IP network must adapt to accommodate new use cases and requirements in the growing modern digital society through software defined networking. Though this network evolution is exciting and essential, the chapter looked at the security risks threatening this new model and how other businesses have been impacted in the past. Whilst perimeter security had traditionally been provided by standalone firewall boxes the chapter introduces the security design in which security resides in the SDN controller infrastructure to reduce deployment costs. The chapter proceeds to define the research aims, objectives, scope and assumptions that will be considered. With the goal of achieving the optimum SDN controller security and performance key performance indicators to be analysed are introduced and how the security implementation will affect them will be investigated.

Chapter 2 presents the foundation and key building technologies of the SDN architecture exploring the 3 key planes that the architecture possesses. The chapter reviews the current security solutions offered by the other authors reviewing their capabilities and limitations. The chapter then introduces the OpenDaylight SDN controller which is the chosen SDN controller to be used in the experiments. An in-depth review of the OpenDaylight controller is done as the selected external controller and how it is affected by the global security challenges. A

literature review on SDN architecture is done to understand the key interfaces used for integration and explores the various security threats vectors that the SDN controller is exposed to. The chapter concludes by introducing the proposed network firewall solution using Linux iptables which is a key control which cements the proposed security solution.

Chapter 3 In chapter 3 we introduce and analyse the functional and performance requirements of the proposed security solution. The analysis of these two requirements feeds and formulates the expected deliverables of the proposed solution. The SDN architecture deployment framework to be used for the proposed security is introduced. The framework consists of the an SDN data plane network in Mininet, control plane SDN controller using OpenDaylight and security solution using iptables netfilter firewall. The chapter then explains how all the components of the overall solution are deployed by use of virtualisation software and technology to separate these logical functions. The chapter also introduces the SDN APIs that are used as integration points in the experimental network setup. The chapter covers the firewall rules logic and algorithm and concludes by introducing the role and use of the network monitoring module in the overall solution for performance results capturing.

Chapter 4 presents the tools and technologies to be utilised in the proof of concept implementation and explanations are given on how they are integrated to produce a working model. This chapter introduces the proposed design components from the tools and overall design operation. The design components covering each part of the SDN architecture are introduced and how they are integrated and communicate with each other during the testing phase. The testbed setup is also introduced as it presents the logical design of the solution and how all design components interoperate. The call flows which constitutes the messages that are generated and exchanged in all the traffic cases between the nodes in the experimental test scenarios evaluation are lastly presented in the conclusion of the chapter.

Chapter 5 presents the 4 key experiments that are crafted from the design for testing the objectives presented in chapter one. The experimental evaluations done in this chapter comprises of the DDoS traffic measurements and verifications, mitigation testing on the proposed firewall and DDoS attack impact assessment on the SDN controller performance. Results of each experiment are discussed and analysed to verify proper operation and coherence with the objectives. The experimental results analysis is done to measure the effectiveness of the firewall solution and to determine the optimum working limits of the SDN controller when it is subjected to the DDoS attack. Impact assessment on the security attack's effect on the SDN controller is analysed by inspection of the trends exhibited by variations of the results in the measured KPIs.

Chapter 6 gives a summary of the dissertation and the work done in all the chapters. To start the trend analysis in the growth of security attacks in new IP-based network technologies is presented. The SDN controller is no exception and is prone to security attacks due to its centralised architecture and threat vectors targeting the SDN planes are discussed. The Linux iptables netfilter firewall design is presented as a security solution to the SDN's most frequent attack source of DDoS. The deployment framework describing the solution operation and

interoperability with the SDN network is presented. The iptables capability to mitigate DDoS attacks is illustrated and the impact of the mitigation to SDN controller performance is shown. The dissertation illustrates the possibility of mitigating DDoS based attacks using Linux iptables coexisting with the SDN controller and how the SDN controller can be optimised to balance performance and required crucial security. It is also in this chapter that future work in DDoS detection mechanisms and SDN scalability tests on iptables is recommended. The proposed solution in summary gives benefits of an easily manageable and integrated firewall solution on the SDN controller which offers lower deployment costs.

CHAPTER 2

2 Literature Review

2.1.1 Chapter Overview

This chapter will look at the key building technologies of the SDN architecture and analyse the security vulnerabilities on each SDN interface. This analysis is key in constructing the required security for the SDN controller. The method of attack on the controller is presented and how existing security solutions are lacking in addressing this security challenge. The chapter ends by looking at the iptables concept and how they can be modelled and utilised to construct a firewall capable of securing the SDN controller with minimal SDN controller system overhead.

2.2 Overview of Software Defined Networking (SDN)

2.2.1 SDN Architecture

As introduced in section 1.1 the SDN technology decouples the control functions and packet forwarding layers from legacy unit proprietary boxes and presents them as separate functions managed by a central SDN controller. The SDN architecture consists of three planes namely application plane, control plane and data plane as shown in Figure 2.1 below. In the SDN architecture, the control plane consists of an SDN controller which is a decoupled function from the data plane. The data plane comprises of packet forwarding hardware such as routers, switches or middleboxes appliances (Intrusion detection systems, load balancers and firewalls). The SDN controller interacts with the packet forwarding switches using open source protocols such as OpenFlow protocol. The OpenFlow protocol will dynamically deploy configurations and policies to the OpenFlow enabled switches [2] residing in the packet forwarding plane.

OpenFlow runs on the southbound (API) interface of the architecture whilst other protocols like network configuration protocol (NETCONF) and REST will run on the northbound application plane for management activities. It is key to look at the key components of the SDN architecture and the various functions of each layered plane so as to realise this architecture's capabilities in addressing the use cases described in section 1.1.

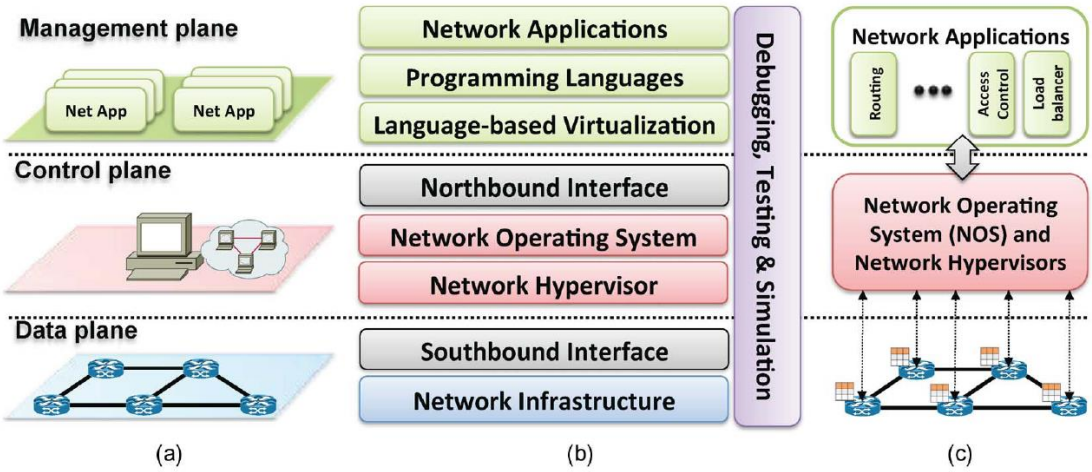


Figure 2.1- *SDN layered architecture* **[9]**

a) **Data plane (Infrastructure Layer)** is the set of network devices that perfom packet forwarding functions of the network. The packet forwarding devices can be hardware or software based and are configured dynamically by receiving the configuration rules from the controller. The configuration rules are propagated through Openflow protocol through the southbound interfaces from the SDN controller. The packet forwarding devices will include routers, switches or appliances similar to the legacy networks without the embedded control network intelligince. In legacy networks, packet forwarding decisions in routers is based mostly on destination IP address only whilst in SDN, packet forwarding actions are determined by flows of packets between both source and destination devices [1].

b) **The Control plane** is considered as the brain [7] of the SDN based network which is responsible for all the decisions and configurations changes delivered to the packet forwarding devices. The control plane will house the SDN controller whose security is a key component of this research. The SDN controller contains the rules set, logic and decisions that are forwarded to the data plane through the southbound API using the Openflow protocol. It thus becomes a crucial element that controls the whole network [9].

c) **The Southbound interface** is the application programmable interface that defines the set of instructions that are deployed to the packet forwarding devices via the OpenFlow protocol. They provide the connecting bridges between packet forwarding devices and control devices [2].

d) **The Northbound Interface** is the application programmable interface that interacts with northbound applications in the architecture. The interface enables middleboxes like load balancers, intrusion detection systems and routing algorithms to be deployed as services in the SDN architecture [2].

2.2.2 SDN Controller

The SDN controller is the core [2] of the SDN network with the centralized logic, policy database and all the required set of rules providing predefined actions to traffic flows that passes through the forwarding plane. The SDN controller enables the SDN network to be programmable and to dynamically adapt to new network conditions. There are several SDN controllers from different vendors which have been released and some still under enhancements using different programming languages e.g. C++, python and Java. The programming languages have their pros and cons with python-based controllers having issues with multithreading and C++ based ones having issues with memory management [10]. There are several SDN controller types which include Beacon, RYU, NOX, POX, Floodlight, Maestro and OpenDaylight. For this dissertation, the SDN controller to be used in the experimental setup is the Java based OpenDaylight project due to better documentation and flexibility [7]. The controller is the preferred SDN controller as it is one of the preferred industry standards firstly deployed by leading vendors like Huawei and Cisco. In basic terms, the SDN controller will have flows proactively or reactively setup when traffic flows in the data plane. In proactive mode, the controller will have preinstalled rules before the traffic traverses the network. In reactive mode flows are installed when the unknown traffic is detected and as it passes the data plane network. Various modules in the SDN controller are discussed in [11] with the key module being the module for routing services to the data plane. The routing service module will have the topology manager and the link discoverable node responsible for discovering and maintaining the status of the links between the controller and the packet forwarding switches using the OpenFlow protocol [11]. SDN controllers can be deployed in distributed setups for large networks to increase resilience, capacity and network system uptime [12]. The distributed solution can be capitalized as a security enhancement which reduces system outage and scaling limitations by having the control function distributed.

2.2.3 SDN Open Virtual Switches (OVS)

The OVS switches are housed in the data plane of the SDN architecture responsible for handling Open System Interconnection model (OSI) model IP layer 2 and 3 packet forwarding actions on the user plane. The OpenFlow OVS switch has one or more tables which contains rules used for decision making [11]. Data flows are forwarded based on rules preconfigured or modified as the packets traverses through the network. OVS switches support standard OpenFlow protocol and connects to the SDN controller via the Southbound interface as shown in the diagram below [13]. OVS switches are critical as they provide several benefits including traffic filtering, quality of service and some level of security through traffic separation in virtual local area network tags.

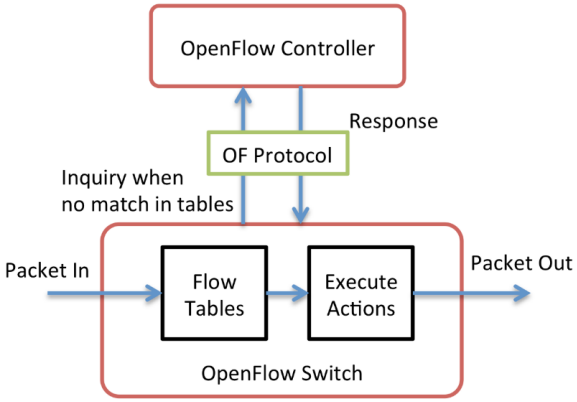


Figure 2.2 *OpenFlow message flow* **[13]**

2.2.4 SDN OpenFlow Protocol

OpenFlow is the key standard communication protocol defined by Open Network Foundation (ONF) that operates on the southbound interface between the OVS switches and the SDN controller [1]. The protocol enables the dynamic flow table updating in OVS switches in cases of new traffic detection on the data plane. OpenFlow is a TCP/IP based transport protocol that operates on dedicated port numbers 6633 and 6653. In [13] the key 3 types of OpenFlow messages are explained and can be summarized as the controller to switch flow (asynchronous) initiated by the controller, asynchronous from the switches to the controller and symmetric messages send without solicitation in both directions [7]. OpenFlow operation involves the recursive packet matching in a series of flow tables in the OVS switch. The packet matching is done by preset priority tags which determine the respective actions and rules to be executed to the traffic e.g. dropping, blocking, forwarding or modifying [7]. The table that analyses the flows is named the table-miss flow entry [14] which is configurable with options to drop the packet, send it to the controller or pushes it to the next table if a rule match is not found in the OVS switch. The successive flow tables in the OVS switches are traversed until the last table with 0 priority is reached which determines the last possible action that can be applied to that packet.

2.2.5 OpenDaylight Controller (ODL)

The controller to be used in this dissertation is the OpenDaylight SDN controller. It is developed in java programing language and supports the OpenFlow protocol required on the southbound API. The OpenDaylight project was founded in 2013 by Linux foundation [1]. Analysis done by [15] summaries the metrics which can be considered in selecting OpenDaylight as a preferred controller in Proof of Concept (POC) implementations. The metrics that can be used include the scalability and performance capabilities of the controller. ODL is a full fabric platform, open source and is supported by the Linux foundation. It provides easy integration and open APIs to other protocols like Openstack used in NFV deployments. It's maturity and reliability on SDN platforms also makes it the preferred controller for this research. OpenDaylight is also slowly becoming the industry's preferred controller by many

service providers (multivendor) with the support and documentation also readily available [12]. The preference is also due to its capability to be backward compatible to legacy IP technology. This makes it possible for a smooth migration and transition to form a new composite converged platform comprising of legacy and modern network segments.

2.2.6 OpenDaylight Controller Architecture

The Figure 2.3 shows the generic framework and layout for the ODL controller and how it merges with other nodes (OVS switches) in the SDN architecture on both the southbound and northbound interfaces [16]. The ODL apache karaf distribution controller works using OpenFlow v1.0 or 1.3 on the southbound interface connecting to OVS switches as an external controller. The ODL controller supports southbound interfaces like the exterior Border Gateway Protocol (BGP) for integration to legacy networks. The integration is done through BGP peering and routes exchanges. The southbound API contains modules such as path Computation Element (PCE) mechanism, open vswitch database management protocol (OVSDB) and crucial Simple Network Management (SNMP) protocol used to monitor the controller by generating SNMP traps for performance reporting [15]. The ODL controller run on most Linux open source platforms with the ODL controller application software installed as software apache karaf container which can be downloaded and customized easily from OpenDaylight online sources.

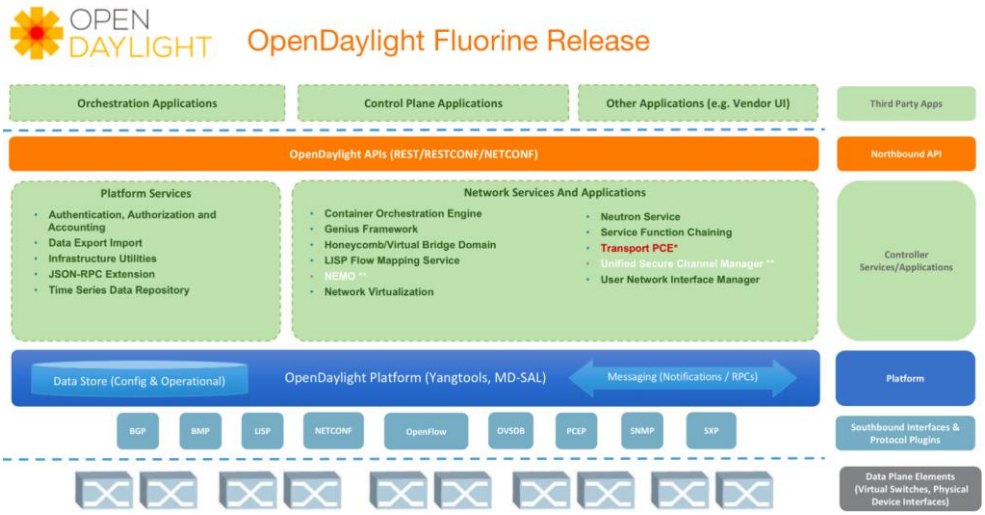


Figure 2.3 *OpenDaylight architecture* [3]

2.3 SDN Security Overview

Security is a key element in every network design affecting both the legacy networks and SDN based architectures. As highlighted in section 1.1 , network availability is key and has to be maintained at all costs for business to function normally and service to be delivered to users. In the current legacy networks, security is mostly deployed through network firewalls with specific modules like Intrusion detection and intrusion prevention functions [2]. The firewalls makes use of static access control lists defining allowed/disallowed hosts. Some advanced

networks makes use of other middlebox appliances like load balancers and deep packet inspection techniques in order to filter and protect the network services from unauthorised attacks at layers higher than the network layer of the OSI 7 layer model. Most firewalls and middlebox appliances are vendor proprietary with use of different commands which makes the adminstration and management of these platforms complex. Due to the static nature of these appliances it would imply that any configuration change would have to be manually done and whatever the current capacity provisioned in these boxes [17] has to handle whatever the traffic conditions that are present until an upgrade is carried out. This presents a challenge with all the new use cases as the network should be elastic and is required to absorb any new dynamic traffic conditions.

2.3.1 SDN security threat vectors

The security concerns in legacy networks still need to be addressed in SDN ecosystems though the approach is different. The static firewalls and middle box appliances will be migrated to the SDN architectures and deployed as services on these platforms as opposed to the vendor specific hardware appliances. By centralising the control function on the SDN controller we increase the risk of network failure over a single point of failure. Securing the SDN controller becomes a critical task to ensure that the network is operating as expected. The SDN controller security has to be deployed from various dimensions and has to cover all angles that can be used as entry points by hackers. The SDN controller security is reviewed by analysing the main key threat vectors affecting the SDN architecture intergration points and are summarised below in Figure 2.4 [9].

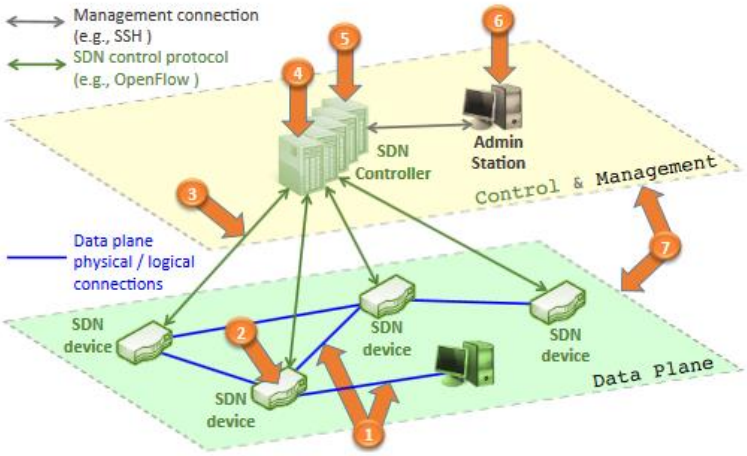


Figure 2.4 *Summary of threat vectors* **[9]**

2.3.2 Control - Data plane attacks

It is apparent that the SDN controller contains the database on how to handle known and unknown traffic classes as it passes through the data plane. The unknown traffic is forwarded to the controller for handling actions thereby exposing the SDN controller to attacks from

within [18].The most perilous risk to controller security is the DDoS attacks which can come in many forms such as fake and forged malicous traffic. An attacker can introduce forged flows that can result in DDoS attacks on the controller as discussed above. Some attackers discussed in [19] show how an attacker can install a listening node which inspects and captures packets illegally using it to attack the network. This type of internal attack is very complex given that the originator posses as a legitmate user of the network. The hacker can generate forged traffic and manupulate the access lists, compromise the hosts and insert malware infecting the SDN controller and packet forwarding switches [7]. Eavesdropping is also a key threat discussed in [20] attacking the vulnerabilities on the southbound SDN interface. Eavesdropping as explained in [21] is achieved by intercepting and inspecting the communication channel between switch to switch and switch to controller communication. The communication channel between the SDN controller and the OpenFlow switches is secured through use of secure socket layer (SSL) and transport security layer (TSL) protocols. OpenFlow uses TSL [22] which is optional making the security weak and opens up to possible black hole and man in the middle attacks.

Man in the middle attack is described in [21] as being similar to eavesdropping where a node places itself between the controller and the packet forwarding switches posing as the controller and infects the switches. Additional discussions done in [20] , [22] and [23] label it as a prevalent security risk in the SDN architecture as the man in the middle attack inspects and manupulate the network by broadcasting compromised policy rules and affecting negatively the integrity of the control messages in the network. The man in the middle attack is highly undesirable as it opens up for other forms of attacks [21] like Identity spoofing , packet sniffing and blackhole attacks. Identity spoofing is done by controller IP address theft with the compromising host taking over the network controll functions [19]. Authors in [24] describe this identity theft of the controller as a hijacked or rogue controller which gives the attacker access to the whole network and presents opportunities of injecting other forms of attacks to bring the SDN infrastructure down. Blackhole attacks are similar to man in the middle attacks which intercept genuine traffic between packet forwarder and the controller [21] but instead of responding normally the packets and requests are dropped.

2.3.3 DDoS based attacks

DDoS attacks are undoubtably the most prevalent form of attacks for the SDN architecture attacking the control plane and mostly emaniting fom the packet forwarding plane when querying the controller for unknown traffic. Authors in [20], [21], [22] and [23] all describe how the control plane is attacked through flooding of the controller with flow decision requests which consumes the controller resources and inhibits its capability to handle legitimate traffic. Ayesha Imran [23] explains how DDoS attacks results in flooding of switch flow entry tables known as switch flow table poisoning [25] which eventually affects the controller processing power capability [26]. Christos Bouras in [21] also describes the impact of a DDoS attack in 5G networks and how the network time protocol (NTP) amplification and malformed packets are utilised in DDoS attacks to perfom centralised attacks on both the packet forwarders and the SDN controller on the NTP protocol.

DDoS attacks are broadly differentiated on the attack mode (how they target) and targeted resource (what they target) as shown in Figure 2.5. This categorisation is described in [27] and [28] with targeted DDoS attacks focussing on critical resources for the SDN hosting equipment such as the CPU and the memory used by the machine. By attacking these resources the attacker will block and restrict other authorised users from accessing the key business applications causing outage on the SDN controller which leads to high revenue loss for operators. In terms of the attack mode type the DDoS is launched by using other hacker technologies such as man in the middle ,IP spoofing and injection of malicious traffic with the same dier consequences of disabling the SDN controller inoperable. These attacks will come as direct or indirect methods [28] by expoliting udp based protocols such as SNMP ,DNS, NTP or ICMP. In some cases other unprotected hosts are utilised and expolited as proxies for launching the DDoS attack on the SDN controller.

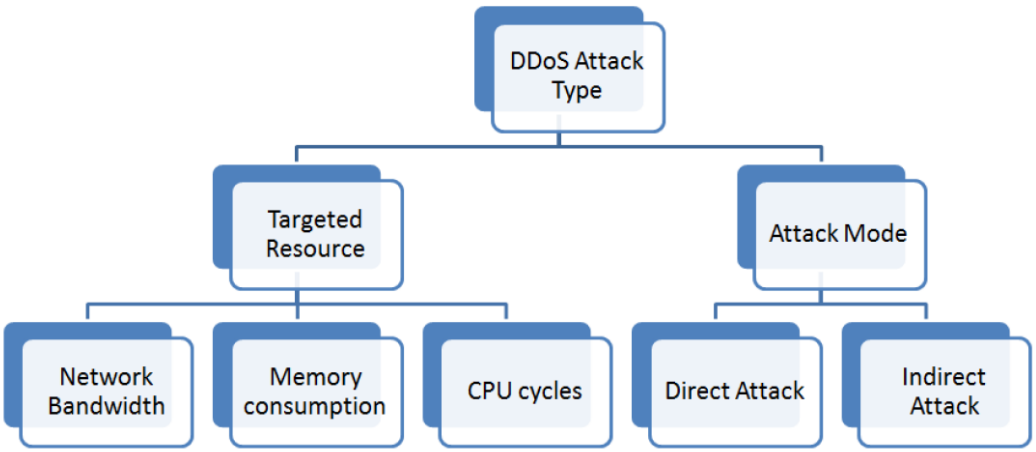


Figure 2.5 *DDoS attacks categories* **[28]**

All TCP/IP communications between servers and clients are initiated by a three-way handshake process making it the foundation for all connections established by the TCP protocol. Without the three-way handshake the server will not open the requested port and will not allow any communication to the client [29], [30]. DDoS attacks are performed by exploiting this three-way handshake process and the attack algorithm is summarized in the points below.

- Client/user will request a connection by sending a SYN packet to the server.
- The server responses to the client with an acknowledgement packet SYN-ACK.
- The client will send an ACK and the handshake is completed, and connection established.

The Figure 2.6 summarises the 3 way handshake and how the flows can be exploited by attackers to initiate a breach. Hackers and DDoS sources exploit the 3-way handshake by flooding the server with SYN packets referencing the IP address of a compromised host as the source in the hacking process also named IP spoofing [31].

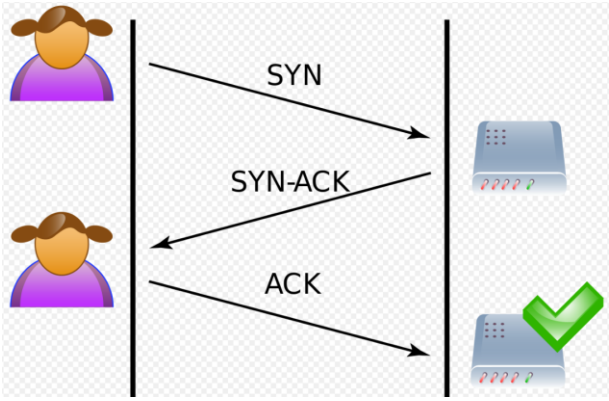


Figure 2.6 *TCP/IP 3-way handshake process* **[29]**

The server responds with SYN-ACK to the request, but the message response never reaches the intended destination and creates half-opened sessions which consume excessive resources on the server. With the flooding of these SYN messages the server is heavily loaded and fails to respond to legitimate traffic due to excessive incomplete connection requests. With respect to the scope of this dissertation DDoS attacks that are utilised in testing are the targeted resource attack type. In the Proof of Concept (POC) the attacker will target the ODL SDN controller residing in the control plane with intent of expolitation of the cpu and memory resources of the controller hosting machine.

2.3.4 Application layer attacks

Similar to attacks on the control plane communicatons threat vectors also exist on the application plane attacking the SDN controller due to lack of strong trust relationships [9]. Application layer threats disccussed by the authors in [20] and [22] include security issues in software, password related attacks and adminstrative station attacks exploiting the weak or no authentication on the northbound interface.This security breach is achieved through exploiting malicious applications capabilities ,weak policies and sniffing features of some customised applications [23] on the northbound interface. The authors in [22] explain how some applications use information collected from deep packet inspection to attack the controller. This loophole is made possible by the lack of encryption on the northbound interface and introduces hacker opportunity of injecting attack sources described in [20] such as malware trojan or worms.

Other areas and angles of attack are explained in [2] and [17] in which the application layer attacks can be extended to subthreats that target the administration stations that access the SDN controllers. These machines can be used as entry points for breaches and used to attack the network. The SDN framework allows integration of northbound interface to other third party systems which opens up the controller to malicious applications from these intergrated third party systems [25]. The integration to third party systems enforces the controller to open up to other possible sources of attacks. Table 2-1 summarises the attack types on the SDN architecture as discussed in sections 2.4.2 and 2.4.4 describing the respective aspect that is compromised on the controller.

Table 2-1 Summary of Security attack types [2]

Attack	Targeted SDN Layer	Affected Security Aspect		
		Availability	*Confidentiality*	*Integrity*
Distributed Denial of Service	Control, Data	x		
Denial of Service	Control, Data	x		
Hijacked/ Rogue Controller	Control, Data, App	x	x	x
Malicious Applications	App		x	x
Man-in-the-middle	Control, Data, Control-data link		x	x
Black-hole	Control, Data, Control-data link	x	x	
Eavesdropping	Control, Data, App		x	

Attack	Affected security feature	Affects data plane	Affects control plane	Affects southbound interface
Data modification/forging	Integrity	Yes	Yes	Yes
Traffic hijacking	Integrity	Yes	No	Yes
Controller hijacking	Availability	Yes	Yes	Yes
Denial of service	Availability	Yes	Yes	Yes
Lack of TLS adaptation	Confidentiality, integrity	Yes	Yes	Yes
ARP poisoning	Integrity	Yes	No	No
LLDP spoofing	Integrity	Yes	No	No
Side channel attack	Confidentiality	Yes	Yes	No
TCAM exhaustion	Availability	Yes	No	No

2.4 Review of SDN security solutions

The preceding sections have been looking at the security threats of the SDN architecture targeting all the SDN layers. There are a number of mitigation techniques proposed by several authors to counter the security threats that affect the SDN controller security. As discussed in chapter 1 Denial of Service Attack is one of the most prevalent source of network attacks with serious downtime and financial impact to businesses. In [2] there are discussions on how the SDN network is greatly affected and threatened by DDoS attacks at both the control and forwarding planes of the architecture [19], [32]. DDoS attacks when launched presents opportunities for other forms of attacks once the controller is made vulnerable hence it is critical that we counter them first.A number of solutions are discussed and evaluated by the authors [2] with each solution having its own merits and limitations. One such solution is discussed firstly by Ayesha in [23] is the integration of the controller to an Intrusion detection system (IDS) or Intrusion prevention (IPS) system. The IPS/IDS systems will detect and mitigate abnormal flows and malicious traffic that are flooded to the controller from the packet forwader network. Intrusion detection system methodologies are presented in [33] which use of signature based detection of a pattern of protocols and anormally based detection schemas for DDoS detection by the IDS nodes. Stateful protocol analysis and inspection is also discussed as an additional methodology for IDS detection which is closely similar to anormally based detection. It is capable of analysing traffic at the transport and network layer and can detect abnormal traffic streams presented to these layers. However IDS/IPS modules are deployed as dedicated appliances that sit outside the controller architecture making them an additional cost in deployment. When deployed int the SDN network they will increase the latency for flow rules updating by introducing an additional hop that increases the controller response times. DDoS attacks exposes the controller to other forms of security threats such as faked and forged flows which are deployed as man in the middle attacks. These threats target the southbound controller API security [9] and other solutions are presented to counter the risks. Secondly, mitigation of malicious traffic is discussed in [34] by use of a transport layer security (TLS) and secure shell sockets layer (SSL) technology which comes with mutual authentication capability between the controller and the packet forwarding switches. However the current

specification of OpenFlow though it includes use of TLS the standard is not specified. This provides a limitation in which a full security specification is required for the protcol to be used to provide the necessary security against DDoS attacks. Authors in [9] share how man in the middle attacks have been deployed by exploiting the vulnerabilities of the TLS/SSL weaknesses as it does not gurantee full proof security to the controller.Activation of the TSL or SSL features to counter man in the middle attacks as a solution results in increased processing on the controller.

Thirdly, authors in [17] and [35] explore how by use of replication of the controllers we can improve security by having a solution which contains more than one controller to avoid total service outage due to one controller being offline. By use of logical centralisation [35] the multiple controllers are programmed such that the network can view the clustered controllers as one logical controller and reduce complexity on the packet forwaders segment.In addition replication controlling schemes are explained in [36] and can be integrated to conflict resolution methods with injection and implementation of extensive authentication models on the southbond interface to improve efficiency of the solution.The replication solution however does not mitigate the source of the attack but focuses on providing redundancy in the event of a controller breach. In addition to replication in [17] there is suggestion of expanding packet forwarder switches caching memory to increase resilience of the switches when exposed to malicious attacks and flooding of fake flows which consumes the switches resources. This solution again only assists in sustaining an attack and does not prevent one. It is also described as costly in [21] due to additional high specification requirements of the infrastructure.

With focus on the SDN controller security the Defense4all feature in the OpenDaylight controller is discussed by many authors in [14] , [16] and [21] as it provides security for the OpenDaylight controller for mitigation against DDoS attacks. The feature sits on the OpenDaylight controller REST API northbound interface [37] and is installed optionally as a plugin. Defense4all continously monitors the protected traffic by counting flows and matching the captured statistics to a predefinded baseline as per configuration and use case. Defense4all through its security agent DefensePro [37] is configured to mark and monitor traffic for at least one week (Peace time) by analyzing the protocols signatures. Once the agent is fully populated with respective rules the application will begin detecting security breaches which require confirmation on a dashboard before any action is performed. When mitigating the threat, the DefensePro agent will divert all affected traffic to an external node which cleans it and reinject it after the mitigation is completed. Defense4all is however not a standalone mitigating solution for the DDoS attacks but requires integration to an IPS through a provionsed interface by redirecting the detected traffic that needs to be cleaned [38].

In other discussions done in [35] offerings on possible solutions are given for countering DDOS attacks in the two planes which include rule aggregation, decreasing switch-controller delay for forwarding plane whilst controller placement and controller replication will work for the controller plane. These methods focus only on the resilience design of the controller.

Some authors discuss various SDN controller architectures that can be used in delivering a highly reliable, scalable and secure SDN network [22]. The solutions to counter the threats of a highjacked /rogue controller by having a multicontroller ,distributed [39] or role based

controller [12] are discussed.This solution is design-based and expensive due to the replication requirements of the controller infrastructure. The deployment of additional controllers will reap the benefits of satisfying business continuity requirements by providing backup infrastructure and provide load balancing platforms which makes the the network more secure and elastic under heavy load. Distributed controllers will have the controllers duplicated in different physical locations for load balancing and reduces security risks on one controller site. The controllers will communicate via westbound and eastbound APIs as shown in Figure 2.7 when synchronizing configurations [12] and performing other system updates. The design eradicates the single point of failure by having a redundant controller which can take over once the primary controller is down. The westbound interfaces can be secured by implementing encrypted IPSEC tunnels for the communication between controller as this information is highly sensitive and should be secured as well.

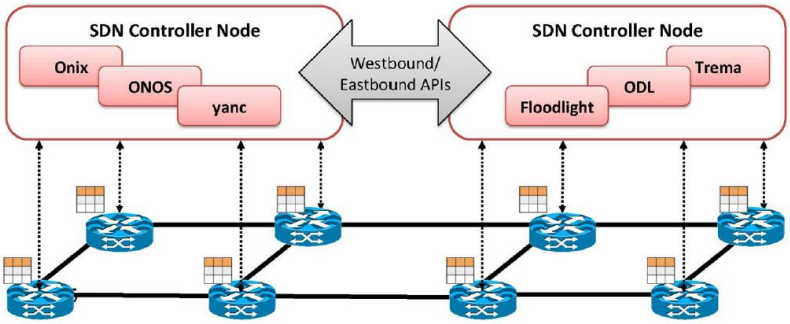


Figure 2.7 *Distributed SDN controller architecture* **[12]**

The other security threats namely black hole attack, eavesdropping and man in the middle attacks are mostly linked to DDoS attacks as the source. The solutions that address DDoS attacks will also mitigate these attacks with one key solution being packet encryption and tunnel bypassing discussed in [17]. The packet encryption solution as described in [26] is a way of protecting the data and messages between the controller and packet forwarders which avoid hacking and unwanted intrusion. With intense packet encryption and tunnel bypassing [17], man in the middle attacks and eavesdropping attempts are curtailed and information is traversed securely on the southbound and northbound interface. Other authors employ the concept of moving target defense [32] which is accomplished by swinging a virtual IP of the control plane which makes the attack complex and difficult. Since attacks can also emanate on the northbound application plane of the controller solutions are proposed to solve this risk by using double encryption of the REST API messages [23],administrative complex passwords [26],software updating and patching to the most up to date versions in the northbound management software.

The SDN controller can also be attacked at the Linux Operating system (OS) level such that all the OS vulnerabilities have to be patched and the OS should be hardened. There a number of best practice linux hardening options that can be deployed such as use of linux security extensions , encryption of data communications, use of SElinux and maintaining the linux kernel in up to date version state [7].

Lastly perimeter security is one of the key security design that requires to be implemented to secure the SDN controller. The perimeter security spans from physical security by ensuring that the controller is housed in a secure physical location with extremely stingent access control policies [2]. Access should only be to provided to approved personel to avoid physical sabotage or tampering to this highly sensitive equipment.The perimeter security also involves protecting the SDN controller from external unauthorised intrusion through deployement of dedicated security appliances. The perimeter is provided by a network firewall by providing borderline edge security and only allowing access to trusted networks defined on the firewall access lists and policies [20]. Perimeter based security solutions with external firewalls do not eradicate all security threats to the SDN controller and poses some limitations due to added hardware.Some threats that are generated internally can be missed as it focusses on the external front prohibiting external networks from accessing the internal network.

With this background of security solutions provided in the discussions above the research will focus on perimeter security within the SDN controller as most solutions discussed above try to deal and mitigate a controller that has already been attacked. It became apparent that we need to ensure that the SDN external network is extremely secured and all threats kept outside and reduces the need of investing in the resilience design of the SDN controller.

2.4.1 Linux based Network Security

Network security is a key requirement of any SDN network to deliver a highly available and Quality of Service for its use cases. Unprotected networks will usually experience service interruptions and cases of system non-availability due to network attacks such as Denial-of Services as explained in [40] and in section 2.4.3 in the previous chapter. These attacks can render key network components unusable or totally unavailable. The research will utilize open source Linux based firewalls that can be easily integrated to the host running the OpenDaylight controller application. Other authors in [41] had proposed perimeter firewalls that sits outside the controller but demand additional hardware and presents an additional point of failure. Figure 2.8 shows a high-level schema for the logical placement of the controller firewall in the SDN network. For the research the firewall will reside on the same host as the controller though logically it will be a separate function as shown in Figure 2.8.

A demilitarized zone (DMZ) as shown in Figure 2.8 above is a security segment on a firewall where the protected host (controller) is accessed by OVS switches from the outside network. This zone shows the protect network in which the SDN controller will reside. The objective of the firewall is protecting any unauthorized access of the resources in this zone [41].

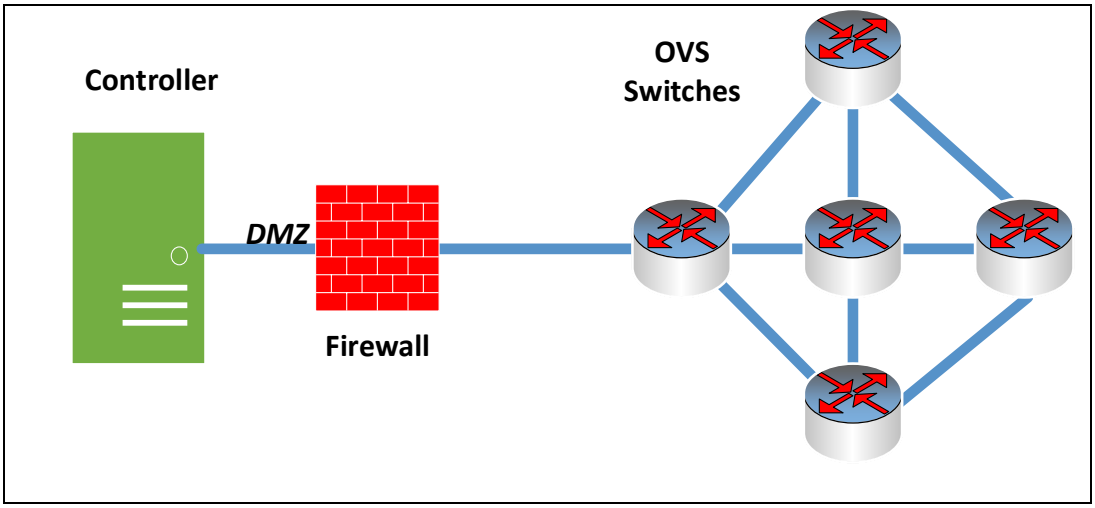


Figure 2.8 *High level for proposed firewall placement in the solution*

2.4.2 SDN Network Security Design

The Linux firewall in the design will provide perimeter-based protection and ring-fencing between the controller and the OVS packet forwarder switches. The two types of firewall capabilities discussed in [42] summarizes the need for a packet filtering firewall that operates at layer 4 of OSI 7-layer model and application filtering firewall that operates at transport layer 7. On application firewall mode the protocol, data and service signatures of the traffic are inspected e.g. HTTP, SFTP. This inspection is tied with allow or deny decisions as defined in the security policies. In packet filtering mode the firewall will provide advanced packet filtering by use of access list rules executed by identifying traffic flows based on source or destination port, source or destination IP addresses. Using the OSI 7-layer model for IP, the firewall will detect and mark any traffic between layer 3 through layer 7 and apply appropriate actions based on the defined rules. In this research our security rules database will comprise of service-based layer 7 rules and packet layer 4 based rules defined in the firewall separating the trusted and untrusted network segments.

2.4.3 Iptables firewall design principles

Linux Iptables is the base critical concept of netfilter used to develop the firewall in this research as it provides the security control point which protects the SDN controller. The iptables firewall provides the first line of defense from external attacks and DDoS mitigation by packet filtering through inspecting IP packets that traverses the Linux kernel. Iptables is a stateful firewall that inspects the state of the connection that passes through it. Iptables firewall logic named netfilter in [43] and summarized in [41] is composed of three tables which are network address translation(NAT), packet filtering and modifying table (mangle table). For the research, the table we will utilize mostly is the packet filtering table capable of filtering based on parameter metrics like IP address, URL, port and MAC address. The iptables logic entails matching predefined rules based on a security policy framework and applying set of actions as defined by the security policies and routed to the specific chain as per designed

policy. The most common actions include ACCEPT, REJECT, DROP and FORWARD decisions which are defined in all polices in the database to take respective action to the traffic that is inspected by the firewall. The diagram in Figure 2.9 shows a logical layout of the iptables modules and schema and how it is mapped to develop the respective actions on the traffic that passes through each module.

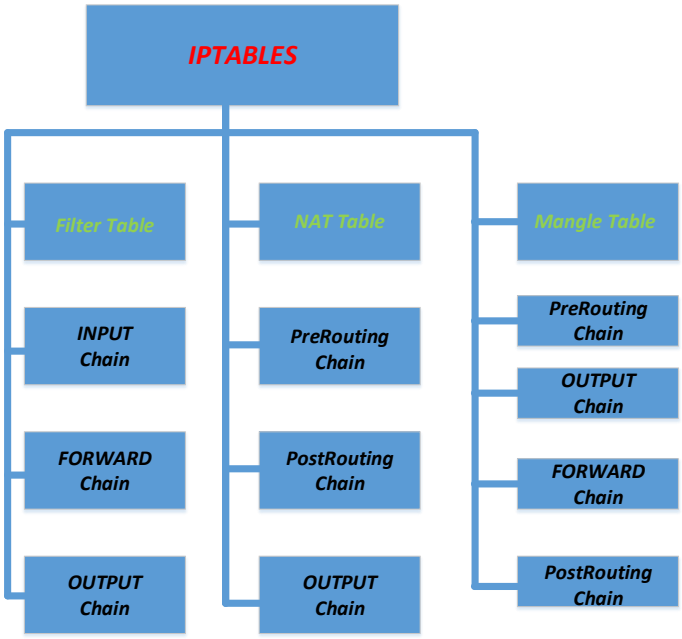


Figure 2.9 *Iptables blueprint and logical schema*

The research will design and build a packet filtering security solution to secure the SDN southbound interface by inspecting all traffic coming from OVS switches and evaluating the impact on the controller functionality KPIs whilst exposed to a DDoS attack.

2.5 Chapter Summary

This chapter focused on the building blocks of the research solution by looking at the SDN architecture, its components and underlying new SDN technology. Of importance is the overview of the SDN controller (OpenDaylight) that is selected for the PoC implementation due to the various benefits it offers as explained in section 2.3. The OpenDaylight controller features are looked at and how they integrate to other components in the SDN ecosystem is presented. The importance of securing the SDN controller which is the heart of the SDN network from any external attacks was emphasized and some solutions that have been proposed and implemented by some in the field to counter the threats was discussed. In the chapter the security threats' sources that target the SDN controller were also covered. The methods used in these proposed solutions were reviewed and which threats vectors they address was covered including their limitations in mitigating the current security risks. A high-level review of the proposed security solution for network security implementation of the SDN controller using opensource iptables on Linux was also done.

3 CHAPTER 3 METHODOLOGY

3.1 Chapter Overview

In this chapter in section 3.2 I will introduce the SDN controller security solution requirements and link them to the current security strategies that other authors have in the field considered. The functional requirements of the security solution will be analyzed in 3.2.1 and the expected design functionality areas will be presented. Additionally, the performance requirements and metrics are presented in 3.4 showing how they will be measured for SDN controller performance impact assessment. The deployment method and how it will interact with the packet data network is introduced in 3.4. Section 3.5 will discuss the high level netfilter iptables framework that will be used in the firewall setup and how the firewall logic will work in the mitigation of threats. In section 3.6 we will introduce the DDoS source to be used in the experiment. In section 3.7 we will briefly look at the resource monitor server using OpenNMS and how it was a key component of the design in collection and tracking the KPIs required in results analysis. Section 3.7 will also present the placement for this monitoring module and how it communicates with the elements in the overall proposed solution. Section 3.9 will look at the tools and technologies required used in the construction of the testbed. Lastly 3.10 will summarize the chapter by presenting the end to end high level design and discussions on how its 3 key components will interoperate in the proposed solution.

3.2 Introduction

The dissertation's objective is to motivate and eradicate the requirement of an additional security appliance by implementing a security solution using a Linux firewall to mitigate DDoS attacks on the SDN controller. To succeed in this implementation, we need to look at the requirements for deployment of a secure controller and its related benchmark performance KPIs for normal SDN network operation. We continuously see that the security design considerations for the SDN based architectures remain critical in the evolution of the SDN technology due security vulnerabilities brought by centralizing the control logic. This inevitably requires great attention to provide stability to the new SDN solution. The SDN controller remains the prime area of target by hackers and requires a strong defense against external networks and software-based intrusions.

3.3 SDN Controller security solution requirements

The security threat vectors presented in section 2.2 shows us the SDN controller attack points and will dictate the security solution requirements essential in providing the SDN controller security assurance. A brief history of the controller security is done in [32] where the authors explained how SDN controller security has been developed between 2008 to 2015 with much focus on ONOS and OpenDaylight the industry SDN controller leaders. The authors cite how in the SDN controller security journey security countermeasures were developed and designed to mitigate only specific vulnerabilities identified in the live and experimental deployments. There are a vast number of security design requirements for the SDN controller

given the multiple attack angles that can be utilized by hackers. With respect to the SDN controller the security of the controller should address the following broad areas listed below. These requirements are analyzed and used in formulation of the proposed security solution.

- **Application /user authorization -** this relates to the SDN controller management interface security by access restriction to sensitive data, password security and protection of system data from unauthorized users and applications [44], [45]. This requirement is implemented on the SDN northbound interface which mostly integrate to third party systems.
- **Authentication capabilities on interfaces of SDN controllers** -this is an SDN controller operation requirement that expects authentication capabilities of OpenFlow protocol for switch to controller communication. Delivery of the security is implemented by encrypting the OpenFlow communication channel by use of TLS or IPSEC transport protocols [45].
- **System and application isolation –** this requirement cover all the general security aspects of the SDN controller environment. The hardware and software vulnerabilities should be addressed by looking at the hypervisor security, VM security, host operating system software security and physical host security. Physical and logical application or system isolation should be supported.
- **Anti-DoS function-** Anti-Dos mitigation is a key security requirement on both the northbound and southbound interfaces of the SDN controller to prevent excessive resource exploitation by external attack sources [44]. This requirement is the most crucial based on security attacks trends presented in Figure 1.1 in chapter 1. It will become the primary security requirement for the proposed iptables solution.

Arguably analysis done by the authors in [2], [21], [23], [25] and [46] have all ranked distributed Denial-of-service (DDoS) as the main threat to SDN controller security with devastating effects on the controller performance. Lobna and Mohamed Faten Zhani [27] explain how DoS attacks affects the controller workload, control plane bandwidth, flow table usage and network bandwidth usage. With background of DDoS being the most prevalent form of attack and analysis done in section 2.2.3 the solution is designed for the Anti-DoS design requirement compliance on the SDN southbound interface.

3.3.1 Proposed solution functional requirements

The functional requirements define how the proposed solution will work, behave and deliver a working model under a set of specific conditions. It covers the proposed system features and capabilities of what each component of the overall solution will provide in delivery of Anti-DoS functionality. The key functionality points are listed below.

a) The proposed firewall should be able to perform ingress and egress packet filtering by attaching block, allow or drop actions based on matches referenced from a list of rules in the firewall policy database. The rules are composed by whitelisting and blacklisting traffic parameters by use of transport-layer information, IP Source Address or

destination IP Address, Protocol Header flag (TCP, ICMP) and TCP or UDP source or destination ports [45].

b) The firewall should be able to ALLOW whitelisted traffic based on the rules composed from the combination of metrics listed in point 1.

c) The firewall packet filtering method should be able to ALLOW the OpenFlow communication channel setup and allow SDN network creation between the remote SDN controller and packet forwarder network. The setup should be done within acceptable latencies of 5-10ms for the proposed linear topology presented in section 4.3.1 [47].

d) The firewall solution should be able detect and block SYN packet attack-based DDoS traffic based on rules matches defined in 1 and 2 above.

e) The firewall should be able to detect policy conflict and implement resolution logic to avoid policy loops and leaks.

3.3.2 Analysis of Functional Requirements

The functional requirements above ensure and sets the guidelines and framework for the design of a security solution for the SDN controller which mitigates DDoS attacks. In point 1 the packet filtering capability will ensure that traffic destined for the SDN controller is analyzed and filtered correctly. The firewall should allow proper communication and operation of essential protocols such as OpenFlow, HTTP (dlux) for controller remote management and SNMP protocol for SDN performance KPIs measurement. The stateful inspection of allowed traffic is done by lookups of the destination ports, source IP address and protocols as defined in the firewall rules. With packet filtering the firewall ensures that all open ports on the controller which are not used in the solution are inaccessible from any external network sources. With reference to SDN network setup the firewall should allow OpenFlow messages to be exchanged between the controller and the packet network within acceptable latencies as the packet filtering will introduce additional system overheads due to application traffic processing. Anti-DoS detection and blocking logic is expected to be activated by the firewall rules to prevent host excessive resource utilization in the design with minimal effect on the SDN core functionally on the southbound interface. The ACCEPT and DROP firewall actions are efficiently defined sequentially in order of set priorities to ensure that the policies do not conflict, and the most accurate rules are called first in the chain as a way of preventing policy conflict.

3.4 Proposed solution Performance requirements

This section describes how well the SDN controller will work whilst coexisting with the proposed security solution on the same host. The key elements that are used and measured in the performance evaluation of the security solution's impact on the OpenDaylight controller are presented. Whilst many performance attributes are available for performance design consideration we focus on the CPU utilization, memory utilization, response rate (latency) and SDN application latency. These metrics are chosen as they are easily measurable [24] and

useful in determining the software behavior, working limits of OpenDaylight and hardware system sizing for the OpenDaylight controller in future deployments.

3.4.1 Central processing unit utilization

This metric will measure the CPU utilization of the OpenDaylight controller application on the hosting machine under specific load conditions. The CPU utilization by the controller is calculated as a function of the provisioned CPU resources in the hypervisor as governed in the hardware specifications design shown in appendix A2.

3.4.2 Memory utilization

The memory utilization of the OpenDaylight is key as it defines the working limits for the controller software application. Closely linked to the CPU parameter its key for system sizing as a hardware specification parameter. It is also provisioned by the hypervisor creating the virtual machine hosting the controller.

3.4.3 Response times

This defines the rate at which the OpenDaylight controller will respond to the data network elements' requests in the OpenFlow communication channel [48]. The metric is measured by use of the ping tool which utilizes the ICMP echo request packet send to the SDN controller IP. The metric is measured in times units (milliseconds) and is recorded by capturing the latency on the ethernet port of the controller which receives all the IP traffic [23].

3.4.4 SDN application latency

This metric is key as it measures the OpenFlow application end to end latency between the packet data network and the OpenDaylight controller. This metric will measure the SDN controller application execution time which averages as the processing round trip delay in creating an SDN network. This performance parameter is measured by capturing the time it takes to create and setup the SDN network in Mininet network emulator software.

3.4.5 Analysis of Performance Requirements

The CPU is a hardware computing resource that is provisioned by the hypervisor tool when setting up the virtual machines. Recommended CPU for OpenDaylight software is 4 cores as shared in appendix A2. The CPU utilization of the SDN controller host is measured by the OpenNMS server by use of SNMP protocol. The CPU utilization is verified by extracting the Ubuntu system resource monitor which gives a more real time report of the SDN controller CPU at any instance. Memory utilization is closely linked to system load performance metric or CPU metric and provisioned by the hypervisor tool. The recommended memory size for ODL is 6GB of random-access memory (RAM) as shared in the hardware specification in appendix A2. Latency will be measured by use of ICMP protocol to verify communication between hosts to determine the response rate of the OpenDaylight controller. This is done by use of ping tool and will be automated by the OpenNMS server. The network creation latency

is measured from Mininet virtual machine as it is setup the SDN network with remote controller. The end to end SDN network creation latency will represent the ODL capacity to respond to requests from OpenFlow switches [23].

3.5 SDN controller and packet forwarders deployment

As highlighted earlier in section 1.4 the SDN controller will be deployed using the OpenDaylight controller connecting to packet forwarding switches deployed in Mininet. The ODL controller will be deployed on a virtual machine created using Microsoft Hyper -V tool. The hypervisor will allocate the virtual machines resources which includes CPU and RAM required by the OpenDaylight karaf container application. The virtual machines will communicate with each other by use of a virtual switch which is defined in bridge mode to create one local private network. The bridge mode on the virtual switch will also enable the VMs to communicate with external networks as its attached to the host physical network interface card. Figure 3.1 shows how the virtual machines are connected to the virtual switch via the virtual network interface cards of the virtual machines. A series of experiments will be done and simulated using Mininet software that simulates the packet forwarding network with hosts, OpenFlow switches and is configured to connect to the OpenDaylight remote controller. Mininet runs on Linux and the code is done in Python scripts as the Mininet-python API is already included in the Mininet software. As expected an IP addressing scheme is the foundation upon which a successful logical design is built. The IP addressing plan for the experiment is based on use of private IPv4 addresses in class A (10.254.0.0/28). The IP block would provide an adequate range of addresses for the controller, packet forwarder switches, DDoS source and SNMP server host IP for resource monitoring functions.

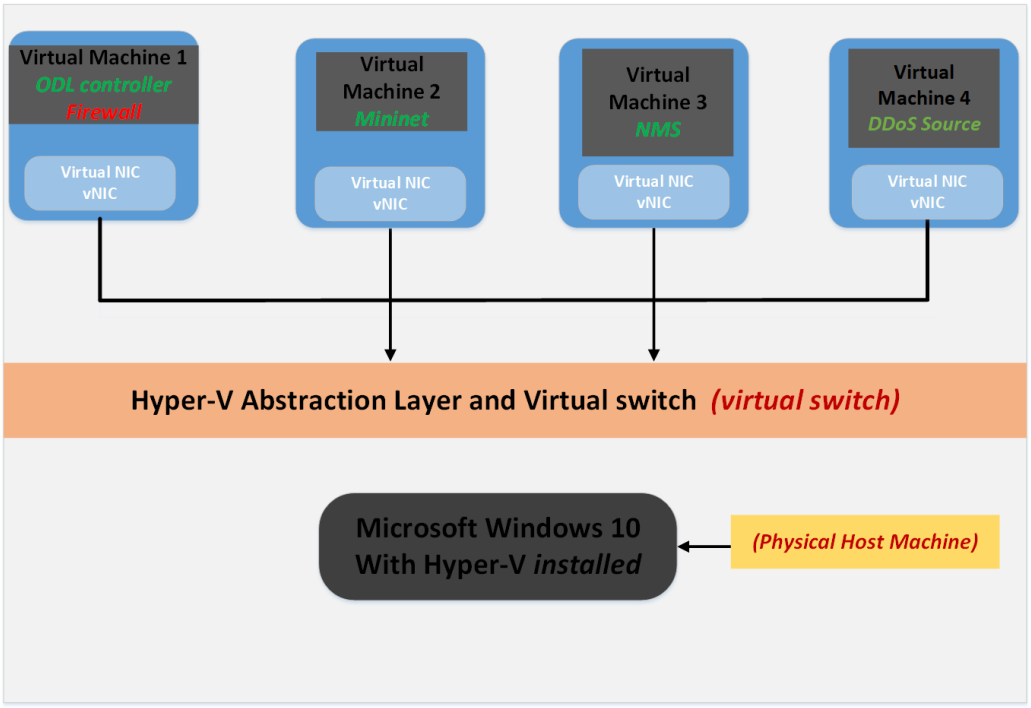


Figure 3.1 *Hyper-V virtual machine deployment framework*

3.6 Iptables and Netfilter deployment framework

In this section we will introduce the firewall solution for the OpenDaylight controller with focus on the firewall placement in SDN architecture, firewall integration points to the packet data network and packet flow logic for threat mitigation actions. These 3 key areas will build the security framework in which the firewall is configured and how it operates in the mitigation process. The dissertation will utilize the iptables tool merged and coexisting with the SDN OpenDaylight controller in building security policies that protect the SDN controller. The iptables tool performs firewall functions and IPv4 packet data filtering by calling five hooks that are attached to the kernel network stack [49]. Figure 3.2 shows the iptables IPv4 hook traversal showing the respective decisions points for all traffic packets to be analyzed by the firewall. The dissertation will utilize the filter table for defining the access control lists for traffic destined for controller and outgoing traffic from the controller. The other tables in the firewall namely Mangle table and NAT table are for other traffic manipulation use cases will and will not be useful in the experiments.

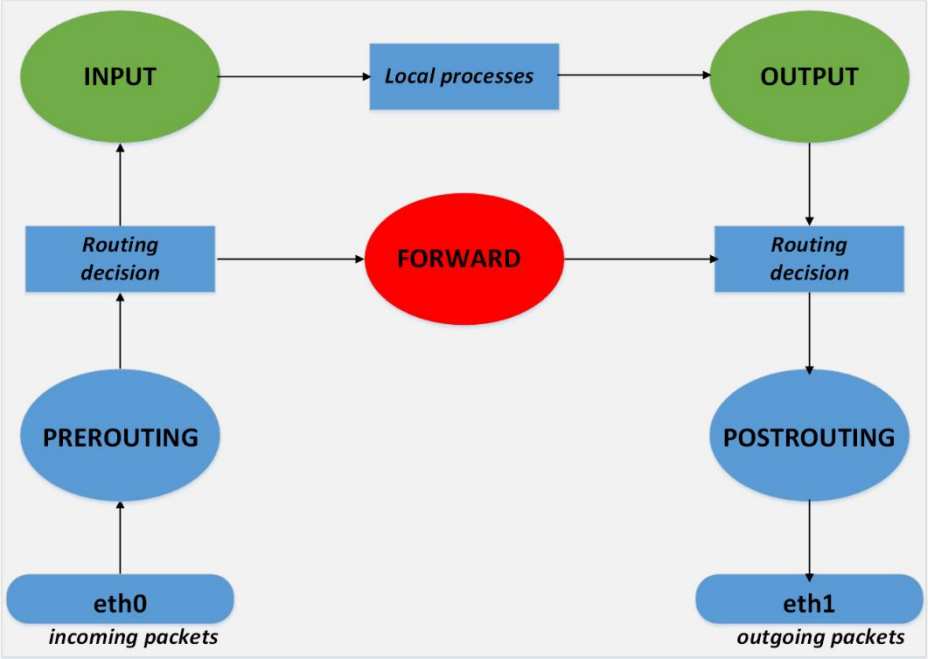


Figure 3.2 *Iptables firewall rules flow*

The iptables firewall in Figure 3.2 will have 3 key chains (INPUT, OUTPUT and FORWARD) which are called when the ingress traffic enters firewall. Firstly, traffic will enter the INPUT chain which is attached to actions that will ACCEPT, REJECT, FORWARD or DROP traffic as defined by the access control lists. The firewall can protect the SDN controller at the data link layer by using filtering rules based on metrics discussed in section 3.2.1. It will then look at the network layer security by applying the rules that inspect and only ACCEPT the source IPs of the packet forwarders. The resource monitoring server which is responsible for collecting the controller performance KPIs is whitelisted to enable traps and alarm collection. All traffic that does not match the rules listed in the iptables will be dropped. At the application layer the iptables

firewall will ACCEPT the traffic based on the destination ports for OpenFlow application (6633 or 6653), SNMP port 161 and OpenDaylight user experience (DLUX) application on port 8181. The latter is required for the SDN management and topology representation and can be launched using any web browser from a host with IP connectivity to the controller. Optionally, the FORWARD action can be used to enhance visibility of source traffic and attach external mitigation by redirecting marked traffic to a syslog server for further analysis [43]. With this framework for iptables we will be able to evaluate the impact of the mitigation of unknown traffic from hosts not listed in the rules as they flood the network with requests there by simulating a DDoS attack on the SDN controller. With use of the iptables application on the SDN controller we should manage to achieve the same functionality of a firewall with less hardware resources in the deployment. This method becomes cheaper to deploy as it removes the need of a dedicated firewall appliance which would be more expensive due to added hardware for the additional host required in the experimental simulations.

3.7 DDoS Source Deployment

A DDoS source is essential in the controller performance benchmarking and mitigation [27] testing of the iptables firewall presented in section 3.3. This will determine the effectiveness of the security solution and can be used in future SDN design considerations. Many versions of DDoS tools are available for security penetration testing and packet filtering rules validation but for this dissertation hping3 tool used [27]. The tool is Linux based and was chosen as it easily integrates with the other SDN modules in the security solution testing.

3.8 Resource monitoring module (SNMP server)

The design includes a resource monitoring module responsible for collection, reporting and storage of all KPIs for the SDN controller virtual machine. The resource monitoring is achieved by deployment of an open source SNMP server that collect traps from the SDN controller host. The SNMP server is deployed using OpenNMS software which is capable of application and server monitoring providing critical performance information of the SDN controller. The SNMP server will act as the SNMP manager which sends SNMP requests on UDP port 161 to the SNMP agent. In our design the SNMP agent will be installed on the SDN controller and will send SNMP traps to the server on UDP port 162 once requested [8]. The SNMP agent will have managed information base (MIBs) which contains object identifiers which defines which parameters that can be read from it by the SNMP manager. The SNMP server and agent associations are established by use of a community name string parameter that is shared between the two entities. Due to the scale of the project we will utilize SNMP version v2c. It is adequate and suitable for a network setup of this size and does not require additional parameters like security passwords between the agent or manager present in advanced SNMP versions 3. The OpenNMS server was chosen as it's easy to configure, provides a clear graphical view of the host performance for easy analysis and can retain historical performance data. In Figure 3.3 below, we show how the SNMP server will be placed in the overall end to end network design and how it interoperates with the SDN infrastructure nodes. Residing external to the firewall the SNMP server will communicate with the SDN controller host via

the same shared southbound facing interface which is utilized by the packet forwarder hosts. In the firewall we will define policies to enable the SNMP server or agent communication by configuring the ingress and egress policies in iptables to whitelist the SNMP traffic.

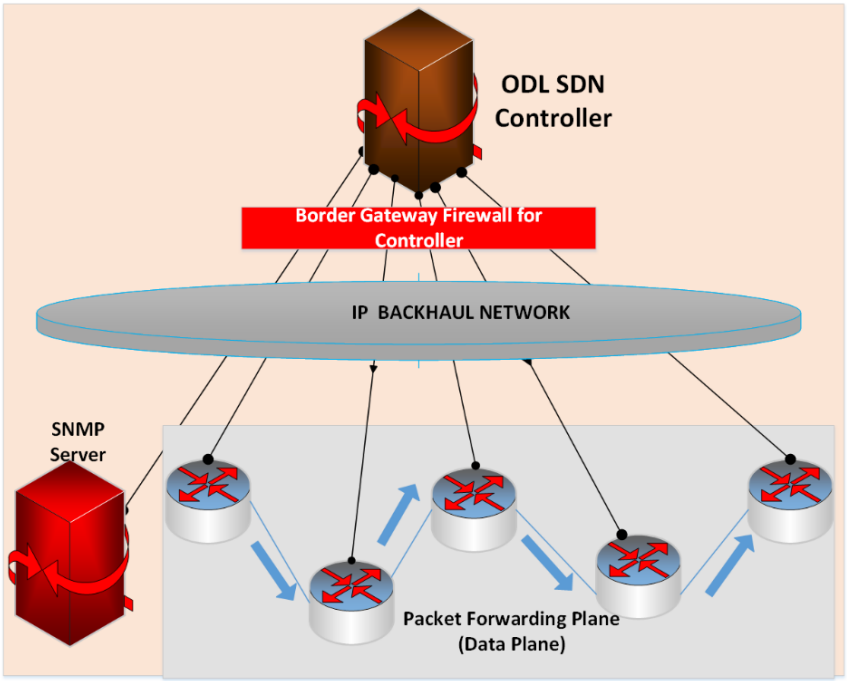


Figure 3.3 *SNMP server placement in the SDN architecture*

3.9 Tools and Technologies

This section lists the tools and technologies used in the construction of test platforms used in the experiments. All the tools utilized in the PoC are based on Linux Debian Ubuntu opensource software. The specific software versions and integration interfaces for all the tools is explained further including the build-up in delivery of a converged secure SDN platform.

3.9.1 Controller plane simulations design considerations

The controller used for the experiment is the OpenDaylight controller which is a programmable controller in line with the modern-day networking fundamentals. The controller is based on a Java software architecture runs on the Linux kernel or windows platform operating systems and can be deployed in any generic hardware. The OpenDaylight controller will run key core services as shown in Figure 3.4. It includes the topology service for tracking how the packet forwarding switches are connected to each other [7]. This is done by instructing the switches to send link layer discovery protocol (LLDP) packets with details on how the packets arrive from the switch which the SDN controller uses to build the topology of the network.

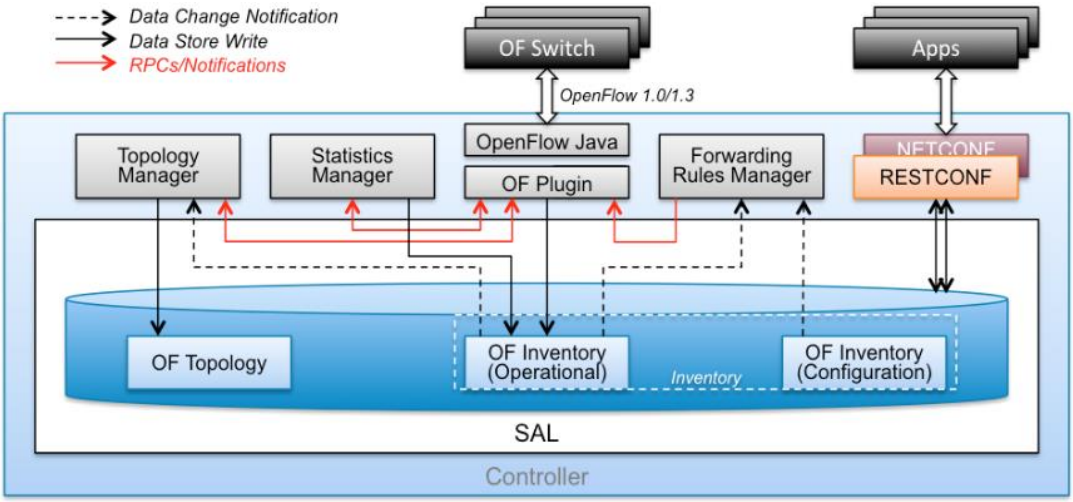


Figure 3.4 *OpenDaylight layout* [16]

ODL controller has an inventory service which it uses to check the version of the OpenFlow protocol running on the packet forwarder switches and their related features. It includes a statistics service responsible for statistics and information on all flow counters and a forwarding rule manager for packet handling instructions for all unknown traffic. Lastly it contains a host tracking service used for locating IP and MAC addresses for all hosts connected to the network and the location of the hosts that reside in the topology.

3.9.2 OpenDaylight

The SDN controller for the PoC will be based on OpenDaylight version 0.4.4-Beryllium-SR4. The high-level framework and architecture for this version of ODL is shown in the Figure 3.5 below. The architecture is generic for most ODL versions and Figure 3.5 shows the OpenDaylight framework details with all supported protocols, features and internal architecture exhibiting all the possible integration points to external systems. Exploring the OpenDaylight controller framework from top to bottom we can see that it consists of two key layers with business and network logic applications on the northbound interface and protocol plugins on the southbound interface [16]. The northbound API supported by representational state transfer (REST) interface exposes the controller to third party applications that are used for configuration and management of the devices. The southbound interface supports SDN protocols such as OpenFlow, network protocols such as BGP, PCEP, NETCONF, OVSDB which are optional in configurations [37]. These protocols can be optionally installed as plugins to the controller depending on system design. Deployed as a remote controller to the Mininet network in the packet forwarding plane the OpenDaylight controller integrates with the switches using OpenFlow protocol.

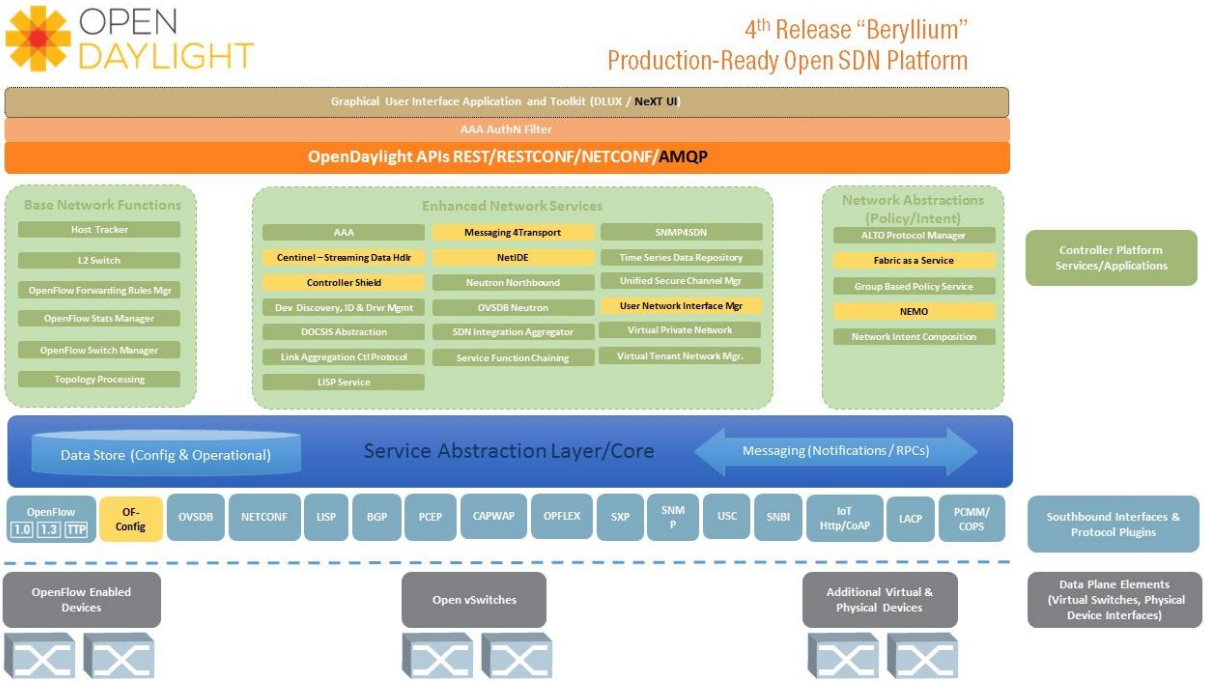


Figure 3.5 *OpenDaylight architecture for the Beryllium version used in the PoC*

3.9.3 Mininet

In the PoC we make use of Mininet which is an SDN based network emulation software for simulating and creating virtual hosts, SDN switches and a reference controller. Mininet is a key modelling tool for SDN-based networks that will provide results on how the OpenDaylight controller will behave under specific predetermined security conditions. Mininet in the PoC is installed on Ubuntu desktop OS virtual machine and will connect to OpenDaylight controller as a remote controller through a LAN virtual switch created on the virtualization software Windows Hyper-V. The external controller is referenced by specifying a remote controller and its associated IP address. Mininet software is chosen due to its flexibility, scalability and ability to be configured for any custom topologies and features [13]. It also supports OpenFlow 1.0 or 1.3 which is the industry accepted standard with multi-vendor controller compatibility, a key prerequisite in building the SDN based networks. The Mininet software was created in python and has a python API which will be used to create custom networks on the network simulation and prototype requirements [50]. The software can be easily integrated to packet monitoring software like Wireshark to analyze the OpenFlow messages used in the connection setup.

3.9.4 OpenFlow Switches

OpenFlow switches are key elements in the SDN based networking providing data plane routing and traffic forwarding for the hosts. Introduced in chapter 2 the OpenFlow switches are simulated in the Mininet software and connect to the OpenDaylight remote controller using the OpenFlow protocol. OpenFlow switches contain flow or group tables which performs

packet lookups and forward traffic based on reactive or preloaded flow actions for the traffic type in question [14]. The OpenFlow switch forwarding actions are determined by installed traffic rules installed by interrogating the OpenDaylight controller. This occurs if it encounters a table miss for no matching flows in switch rules database. In querying the controller, the OpenFlow switch sends a PACKET IN message. The PACKET IN message encapsulates the original traffic from the hosts or TCP packet header information and a buffer ID for the stored packet for future reference. The controller will send a PACKET OUT message to the switch with handling instructions as a FLOW-MOD modification which has various associated actions to forward, flood or drop the packet [13]. With this advanced logic the OpenFlow switches can construct the group tables with appropriate actions to all future packets by referencing the buffer IDs of previously installed actions in the flow tables.

3.9.5 Windows Hyper-V

The Windows Hyper-V (hypervisor) is a virtualization tool that allows server or desktop and storage virtualization on the same physical machine. The software tool was introduced on Windows server 2012 or Windows 10 desktop operating systems versions [51]. The Hyper-V application allows exclusive creation of independent computer hosts as virtual machines that operate independent of each other. The tool can be used to run multiple operating systems which include Linux, old windows operating systems thereby enabling easy testing and experimental trials of software development. In summary the Hyper-V tool is capable of dynamically adding storage, Random Access Memory and networking properties without powering off the host machine. With the Hyper-V tool, the allocated Virtual Machine resources will be dynamically adjusted which ensures efficient use of the physical host machine. The tool also comes with no additional license requirements making it superior to other tools used for virtualization tasks. In the PoC using the Hyper-V tool we will create 4 VMs for the ODL controller, Mininet, NMS server and DDoS traffic generator host shown in the below. In terms of IP network connectivity, the VMs are interconnected through a virtual switch created by Hyper-V tool and attached to the physical host machine network interface card.

3.9.6 Ubuntu 16.04

Ubuntu 16.04 is an open-source Linux distribution operating system which can run in desktop or server mode. The operating system is easy to install, allows customization and installation of relevant packages to the user depending on use cases. Using the Hyper-V tool described in section 3.9.5 in the PoC we will create 4 VMs for the ODL controller and Mininet packet forwarder network using Linux Debian Ubuntu version 16.04 as the base operating system for our applications. This Linux kernel was chosen because it supports all the SDN protocols and all software packages required in the PoC. Ubuntu 16.04 supports Java application (required by ODL), SNMP protocol (host KPI monitoring), iptables (firewall solution), Mininet and many other open source protocols. The Ubuntu software also comes with no licensing requirements and relatively good support with documentation on the internet and Linux Debian based forums.

3.9.7 Wireshark packet analyzer

Wireshark is an IP packet analyzing tool capable of analyzing network protocols in real-time or offline modes. Wireshark is an opensource tool which runs on multiple platforms including Windows, macOS, Linux and FreeBSD with deep packet inspection and decryption capabilities on some secure protocol like IPSEC, SNMPv3 and WEP. In the PoC implementation we will use Wireshark to analyze the DDoS messages between the ODL controller and the VM4 generating the DDoS traffic. OpenFlow messages generated between the OpenFlow switches and the ODL controller can also be extracted by Wireshark making it a very useful tool in traffic analysis. Running in the background Wireshark will capture all IP packets generated by the network interface cards both on WIFI, local area network card and virtual interfaces. On the Wireshark tool will use filters to select the appropriate interface for traffic listening and the respective parameters (IP address, protocol or MAC address) of interest as a way of extracting relevant traffic and to conserve disk space.

3.9.8 OpenNMS NMS tool

OpenNMS is a popular free network monitoring software with multiple applications in network monitoring, proactive troubleshooting and some deep packet inspection use cases. The software can monitor the hardware status and performance metrics like CPU, latency, memory for the hosts. The OpenNMS software will use SNMP and ICMP protocols for data collection through polling of host device under monitoring. The devices to be monitored should have SNMP service enabled and running with preconfigured common SNMP community strings which are accepted by the OpenNMS server. The OpenNMS server will be deployed on VM3 and will poll the ODL VM1 server as we test the security mitigation impact on ODL server performance. The OpenNMS server should be able to provide usage statistics of the CPU trends and latencies of the ODL server in graphical output which is used in the performance results and evaluation.

3.9.9 DDoS attack tools

DDoS attacks stand out as the highest and most probable attack source for the centralized SDN controller with devastating effects [29]. In the PoC implementation we will make use of open source DDoS attacking software configured to attack the controller. Many various attack tools are available online with different benefits and attack angles including volume-based attacks to affect bandwidth, protocol based and application-based sources attacking specific applications. For the experiments in the PoC we will use the hping3 Linux based command line tool for generating DDoS traffic that targets the controller. In [30] the author explained how DDoS attacks are launched by exploiting ICMP messages in the form of SYN flood named smurf attack or SYN flood attack. This method is similar to the attack method we will use in the experiments.

3.9.10 APIs and Protocols

Simple Network Monitoring Protocol is an application layer protocol used for collection of management information on network devices. [8]. The protocol will be used in the PoC to collect management information for the ODL controller.

OpenFlow is the southbound protocol which enables communication between the OVS switches and the ODL controller [13]. By use of the OpenFlow channel created between switch and controller OpenFlow enables controller to switch communication. In the PoC OpenFlow is required for SDN network setup to ODL controller through the iptables firewall.

Open V switch database (OVSDB) is a switch management protocol that is programable and used in the SDN network deployments [14]. With the use of OpenFlow the OVSDB is utilized in the data forwarding network in the experiments.

3.9.11 Mininet python API

The Mininet module in the PoC is the core software for the packet forwarding network simulations. The simulations will be done through configuration and executing a python-based script by use of the in-built python API in the emulator software. The PoC will use custom python scripts to construct the packet forwarding networks making configurations easy to customize parameters such as topology, OpenFlow version, remote controller IP address and OpenFlow port for connection to the ODL controller. The custom network option will provide easy deployment, scalability and provides easy modifications to the network configurations due to the different permutations in the various test cases.

3.10 Chapter Summary

In summary, the chapter introduces the key functional and performance requirements to be considered in building the components required to deliver the iptables based firewall for the SDN controller. The chapter introduces the deployment environment for these components and framework that will be used for their interoperability. The experimental integration points are discussed in this chapter and required protocols to enable the components to communicate are explained. With focus on security the iptables firewall foundation and framework is explained and how the logic in threat mitigation will operate. As performance monitoring of the controller is a key output and deliverable of the dissertation the resource monitor server was introduced and how it will be integrated to the SDN infrastructure for statistical data collection required in the analysis of the results. The chapter presented the framework required for the implementation of the proposed solution presented in detail in chapter 4. Lastly the chapter presented the tools and technologies used in the control plane infrastructure using OpenDaylight controller.

4 CHAPTER 4

PROOF OF CONCEPT IMPLEMENTATION

4.1 Chapter Overview

In this chapter we introduce in detail the tools used in the design, the integration points for sub-modules and deployment of the solution in a virtual environment. The chapter is outlined as follows: 4.2 looks at technologies used in the control plane infrastructure using OpenDaylight controller. In this section we introduce the testbed and overall design constituting the hardware and physical layout, logical network design and control to data plane designs. Section 4.3 introduces the Iptables based proposed solution which mitigates the DDoS traffic. The associated application call flows between the design components is introduced. Section 4.4 provides a summary of the chapter.

4.2 Introduction

In the previous chapter we introduced the high-level design and requirements of the discrete components that build the prototype forming the security solution under assessment in this dissertation. The security solution proposed is centered around the SDN controller which is the main target of attacks in the SDN technology as highlighted in the security related risks paper in [18] and [25]. Additional SDN security solutions discussions are done by in [20] on self-defending mechanisms that have been used in the past on RYU and ES-Flooding controllers. These designs were achieved by use of traffic counting and monitoring software to detect potential threats. The solution in [20] heavily depends on IPS or IDS integration for control plane security and requires activation of flows [52] statistics collection, anomaly detection, and anomaly mitigation modules in the controller under security testing. The methods presented in the two papers [20], [52] can be considered slow in mitigation as the convergence time for threat detection is high as traffic threats are first analyzed and marked before traffic blocking can be activated. This is not ideal as mitigation response as it should always be close to real time. The netfilter iptables becomes a key module in the proposed design which is discussed in this chapter and how it will control and manage all traffic traversing the SDN ODL controller.

4.3 Testbed Setup and Design

This section explains how all the protocols, tools and technologies introduced earlier are utilized in the delivery of a PoC solution for the dissertation. This section looks in detail at how these tools are integrated in the delivery of the following key design areas: physical layout design, network logical design, firewall design implementation for mitigation and call flows for interoperability of applications.

Table 4-1 Experimental setup virtual machine roles

Host ID	IP address	Role	Applications
VM1	10.254.0.1	ODL controller Firewall	OpenDaylight, Iptables, OpenFlow, SNMP, HTTP
VM2	10.254.0.2	Mininet	OpenFlow
VM3	10.254.0.3	NMS server	SNMP
VM4	10.254.0.4	DDoS source	SSH, HTTP

4.3.1 Hardware and physical layout design

The 4 key nodes in the PoC are deployed as VMs by use of the Microsoft virtualization software Hyper-V and are viewed as standalone logically separated hosts. In Figure 4.3 exhibits and gives details of the role of each VM in the PoC, VM host IP address and applications the VMs will be running. It includes the physical layout of VM1, VM2, VM3 and VM4 in the virtualization environment for easy traffic flows interpretation and logic. The 4 VMs are configured in Hyper-V software with custom system requirements specifications including RAM, CPU, operating system and HDD space due to different software requirements of each application. The specific system requirements used in the VMs is shown in appendix section A2.

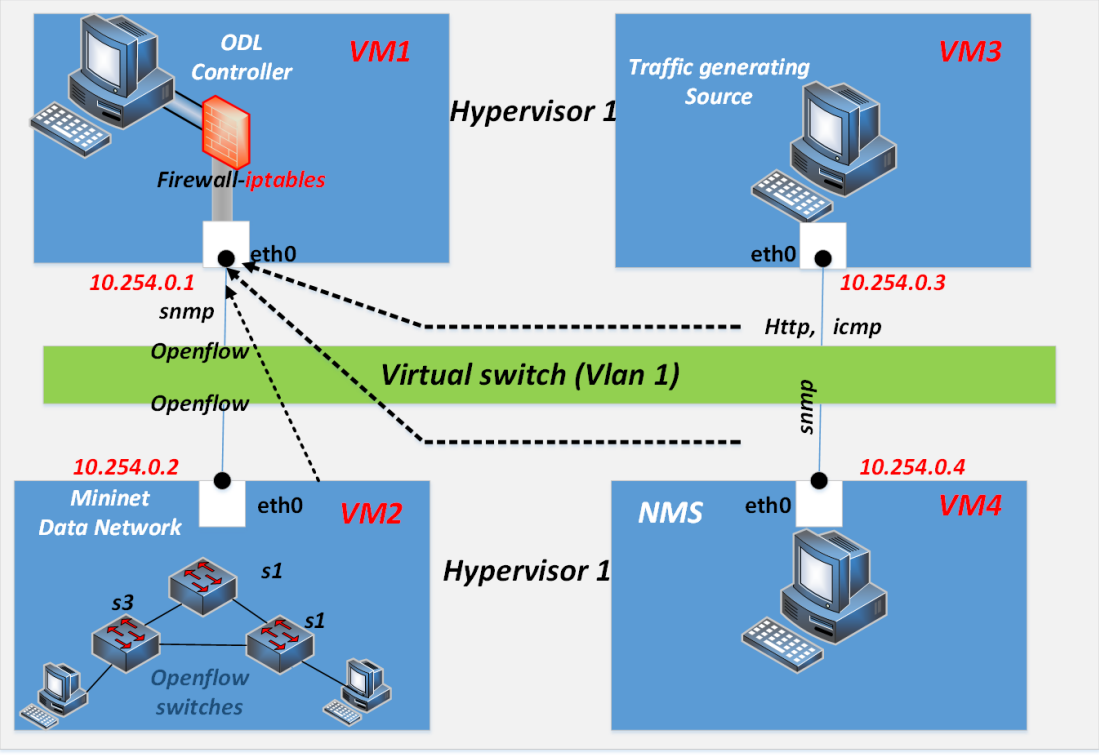


Figure 4.1 *Hardware and physical layout design setup*

The VMs are allocated static IP addressing as opposed to DHCP as per IP addressing plan highlighted earlier in section 4.2.6. Static addressing structure will ensure that the hardcoded rules in iptables are accurate and maintained matching the correct traffic sources for efficient packet filtering. As highlighted in 3.2 the 3 VMs are interconnected by use of a virtual switch for all internal communication utilizing vlan 1 which interconnects them in one logical network. A virtual LAN, as its name suggests, is a concept of logically segmenting switch ports that reside on the same network. This result is the creation of a broadcast domain on the switch and enables all the VMs to be logically connected.

4.3.2 VMs logical network design

The logical network design defines how all TCP/IP enabled devices (VMs) will communicate in the PoC. Following the physical structure discussed above the four VMs will communicate using the OSI model layer 3 with unique IP addresses. The hosts will communicate with each other through a virtual switch on the network subnet of 10.254.0.0/28. Connectivity between the VMs is verified by ICMP tests done from each host. The packet forwarding VM2 machine will setup the SDN network with the ODL controller and presents its own topology as defined in the Mininet network parameters discussed in the next section. VM1 and VM3 will communicate using SNMP protocol for traps collection whilst VM4 will use the hping3 tool over the same physical virtual switch through polling the application ports of 161 and 80 as allowed in the iptables policies. The connectivity and the flow of traffic between the VMs in illustrated in Figure 4.4.

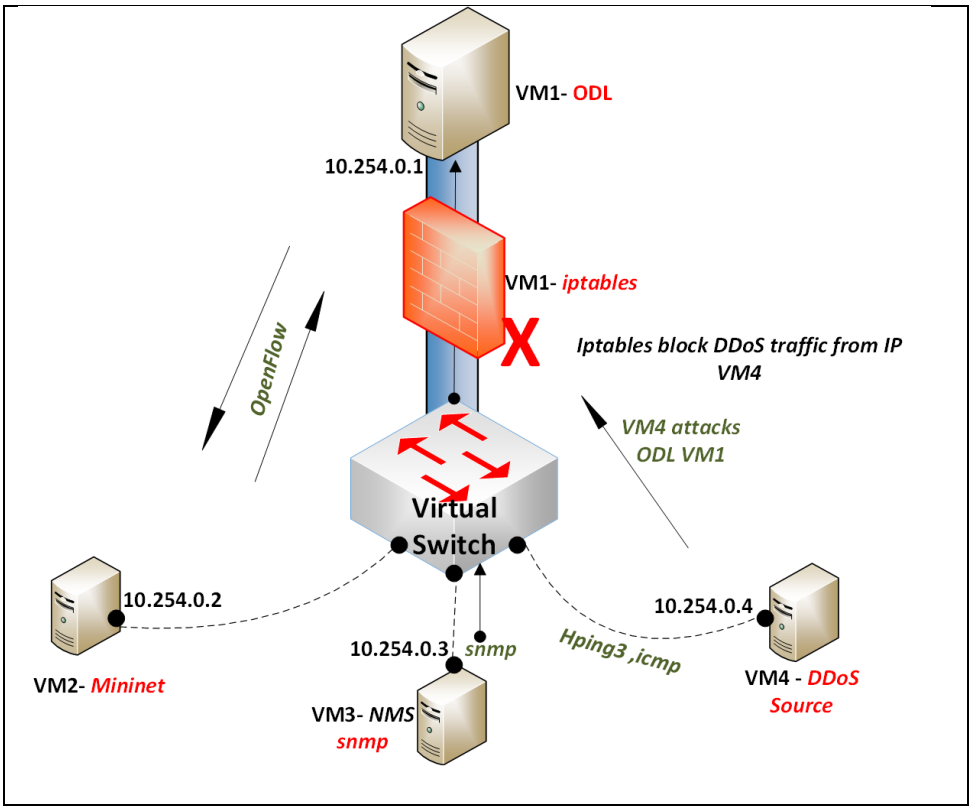


Figure 4.2 *Proposed logical testbed layout*

4.3.3 Data plane design

The data plane design is based on a Mininet build on VM2 which connects to a remote controller via a virtual switch. Containing the packet forwarder network VM2 host is whitelisted in the iptables firewalls to enable connection to the controller with OpenFlow on port 6633. The Mininet network is configured as per design parameters shown in the table 4.2 below and reference is made to the role of each parameter setting. The Mininet OpenFlow based switches will create an OpenFlow secure channel by referencing the IP addresses of the external ODL controller to setup the channel [13]. The SDN network inventory and the number of connected nodes is verified by issuing dump and net commands which lists all the elements created in the network. The output provides all the hosts, switches and controller details and the respective OpenFlow channel session IDs named PIDs. Results for the network created, topology and ethernet links verifications is shared in the section 5.3.1. The topology of the network and allocated MAC addresses are counter checked by the ODL Dlux application which shows how the hosts are connected to switches or controller and the associated ethernet links.

Table 4-2 Mininet launch parameters

Parameter reference	Configuration	Comment
Topology	--topo linear, 4	Linear topology
Controller	Controller = remote, ip = 10.254.0.1	Remote controller in VM1 with IP address 10.254.0.1
Southbound protocol	Openflow13	Connect using OpenFlow version 1.3
Host MAC address	--mac	Include mac address of hosts

Additionally, the flow of messages between controller and switches are checked by use of Wireshark application which shows the OpenFlow protocol message details.

4.3.4 Control plane design

The control plane design is centered around the ODL controller, the southbound interface and OpenFlow protocol integrated to the packet forwarding network. The ODL setup in the PoC will have two key ODL features installed namely odl-l2switch-switch and odl-dlux-core. The odl-l2switch-switch is used for enabling the OpenFlow protocol and southbound API communication [53]. The odl-dlux-ore is used for GUI web-based URL management access for the ODL controller. The network setup is viewed by ODL dlux application which is accessible on http://10.254.0.1:8181/index.html. The topology that was extracted from the URL is shown in Figure 4.5 below. These key features are required in supporting OpenFlow protocol and

activating remote web-based management on the controller respectively. Specific detailed steps of the ODL installation done on VM1 are shared in detail in appendix section A.3. It should be noted that VM1 which hosts the ODL application requires the highest system resources in terms of RAM and CPU due to the heavy java based OpenDaylight application.

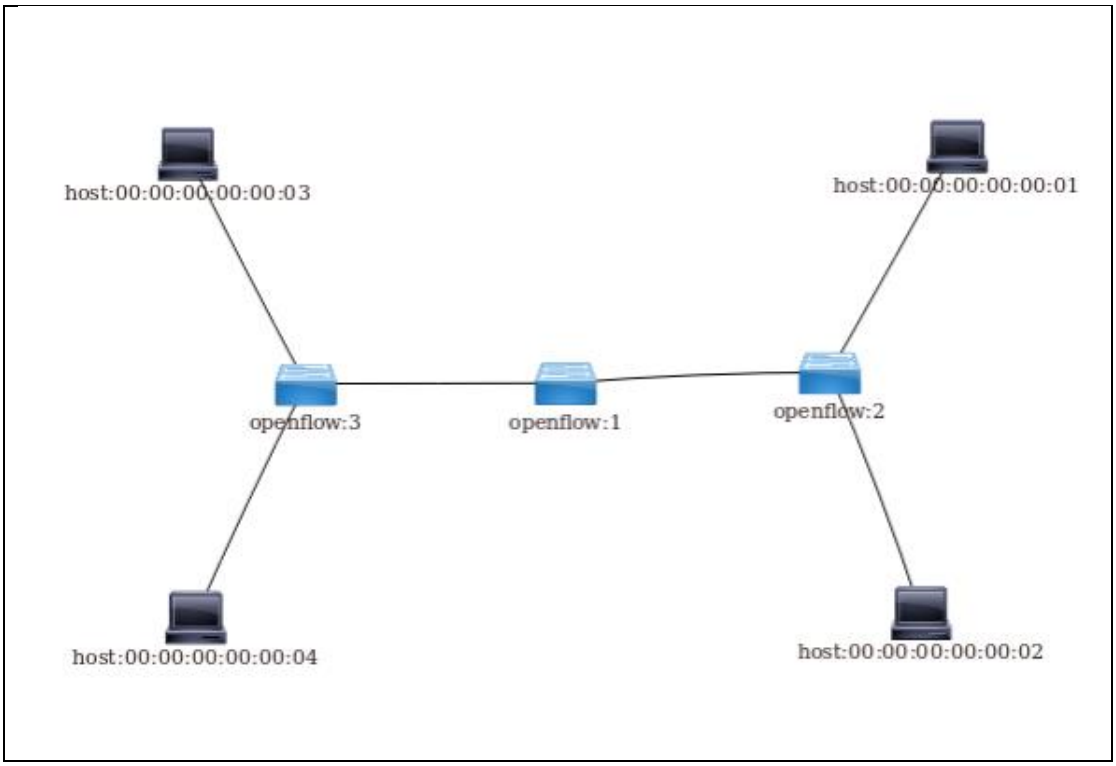


Figure 4.3 *SDN network created as viewed in ODL dlux portal*

4.3.5 DDoS traffic source design

In the PoC we utilize the hping3 tool to initiate DDoS attacks towards VM1. The attack is targeted on a specified open port 80 on VM1 by flooding the host with SYN requests. As per the design notes shared earlier this traffic is controlled will be blocked by iptables in VM1. Installation steps of hping3 tool are shared in detail in appendix section A.7. To launch the application targeting VM1 with the SDN controller the below commands was used. The flags and parameters used in the attack launch are explained below.

hping3 --flood –S –p [29]

--flood flag sends the packet as fast as possible

-S flag sets the SYN flag on in TCP mode

-p 80 sends the packet to port 80 on victim's

4.4 Iptables based Firewall implementation

The firewall is deployed by use of Linux iptables on VM1 coexisting with the ODL controller by enforcing rules that permit and block specified traffic in the iptables database. DDoS traffic from VM4 will be dropped by iptables as part of mitigation of the DDoS process. The iptables firewall module keeps stateful track of each connection passing through it and tries to anticipate future actions [41]. Traffic rules for the INPUT policy chain for iptables for all VMs and respective actions are summarized in table 4-3 below. This ingress traffic represents traffic external to the controller and viewed as source traffic by the iptables.

Table 4-3 Ingress traffic rules design guidelines

FORWARD POLICY (ingress traffic) INPUT CHAIN						
	Source IP	**Destination IP**	**Protocol**	**Port**	**Target/action**	**Comments and flags**
rule 1	10.254.0.2	10.254.0.1	OpenFlow	6633	ACCEPT	*Allow VM2 to access VM1 on OpenFlow port 6633.*
rule 2	10.254.0.2	10.254.0.1	HTTP	8181	ACCEPT	*Allow remote monitoring of DLUX application from VM2.*
rule 3	10.254.0.3	10.254.0.1	SNMP	161	ACCEPT	*Allow VM3 to access VM1 on SNMP port 161.*
rule 5	10.254.0.3	10.254.0.1	HTTP	8181	ACCEPT	*Allow remote monitoring of DLUX application from VM3.*
Rule 6	10.254.0.4	10.254.0.1	HTTP	80	DROP	*Block VM4 to access VM1 on HTTP port 80.*
Rule 7	10.254.0.4	10.254.0.1	ICMP		DROP	*Block VM4 ICMP traffic to VM1*

Packet filtering in the PoC setup is done by inspecting source address, destination address, transport level protocol (i.e. TCP, UDP, ICMP) and source or destination application port [41] as highlighted in section 3.2.1. This inspection is done for both ingress and egress traffic as determined by INPUT and OUTPUT modules in the iptables rules database. Like the INPUT chain the OUTPUT chain will have policies that filter egress traffic from the ODL controller. The iptables rules will attach ACCEPT rules to Mininet VM2 and BLOCK rules to DDoS source residing on VM4. The table below in 4.4 summaries all the egress traffic rules and respective actions applied to the traffic. Tables 4.3 and 4.4 defines the iptables firewall policies that

determines which traffic is allowed and blocked in the PoC network. The iptables database after all the rules were applied are shown in the Figure 5.17. Additional verifications of allowed or blocked traffic are shown by use of the ping tool and the screenshots are populated in the appendix section A.9.

Table 4-4 Egress traffic rules design guidelines

FORWARD POLICY (egress traffic) OUTPUT CHAIN						
	Source IP	**DEST IP**	**Protocol**	**Port**	**Target/action**	**Comments and flags**
rule 1	10.254.0.1	10.254.0.2	OpenFlow	6633	ACCEPT	Allow VM1 to access VM2 on OpenFlow port 6633.
rule 2	10.254.0.1	10.254.0.2	HTTP	8181	ACCEPT	Allow remote monitoring of DLUX from VM2.
rule 3	10.254.0.1	10.254.0.3	SNMP	161	ACCEPT	Allow VM3 to access VM1 on SNMP port 161.
rule 5	10.254.0.1	10.254.0.3	HTTP	8181	ACCEPT	Allow remote monitoring of DLUX from VM3.
rule 6	10.254.0.1	10.254.0.4	HTTP	80	DROP	Block VM4 to access VM1 on HTTP port 80.
rule 7	10.254.0.1	10.254.0.4	ICMP		DROP	Block VM4 ICMP traffic

4.4.1 Application layer call flows

The following section discusses the call flows and sequence of messages exchanged between the 4 VMs and when all applications are running. The logical flow of these messages is divided and summarized into 3 key flows explained below.

- **Flow 1**: This involves the setup of SNMP session between the NMS server and the ODL SNMP client. The NMS server will send a SNMP GET request on port 161 to the client which traverses through the iptables firewall. The traffic is already whitelisted in the iptables rules INPUT chain and tagged as legit traffic. It can communicate with the ODL server open port 161. The client will send a SNMP response which is whitelisted in the OUTPUT chain of iptables thereby completing the SNMP session with the ODL server sharing the monitored KPIs using the managed information base (MIB) files. This communication between NMS server and client is repeated with the server polling the client every 1-2 minutes to collect the latest statistics generated on the SNMP client.

- **Flow 2**: This flow type will have the traffic generating source initiating DDoS traffic flows using hping3 tool with the ODL server as the target host in the staged DDoS SYN flood attack. The attack for this experiment is done by flooding HTTP port 80 which sends a continuous stream of attack packets to the host 10.254.0.1. The iptables will have a blacklist policy in the INPUT and OUTPUT chain for blocking the traffic emanating from host 10.254.0.4 the source of the DDoS attack. This flow is induced on the ODL server for the rest of the testing to see impact on the ODL server resources.
- **Flow 3:** Involves the normal setting up of an OpenFlow channel between the Mininet host and the ODL remote controller. The flow involves a 3-way handshake between the OpenFlow switches and the ODL controller. Traffic from the VM host 1 to VM host 2 is whitelisted in the iptables firewall and the OpenFlow port, and port 80 for *dlux* application are opened. This flow constitutes the key testing of the ODL controller performance whilst under a DDoS attack and is repeated under different VM1 states.

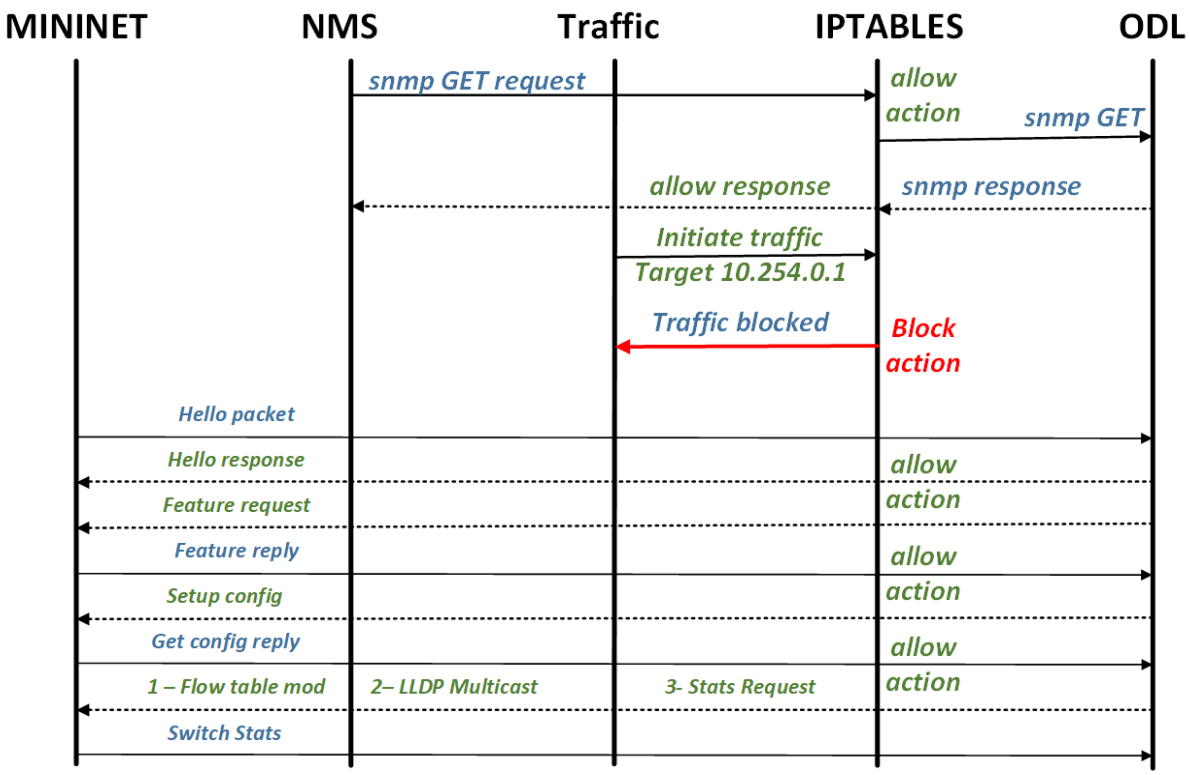


Figure 4.4 *Application call flows*

Figure 4.6 summarizes the 3 application call flows described earlier in this section and the respective VM the traffic flow emanates and terminates.

4.5 Chapter summary

The SDN controller remains the core node in the SDN architecture that is susceptible to many security attacks due to its centralization. It remains highly plagued by DDoS attacks and ever-increasing hacker attempts in the cyber digital world. Network security is relatively and highly mature area of information technology with vast number of firewalls solutions available. The continuous advancements of countermeasures to network security breaches are still being researched and developed with new traffic detecting methods being released. However, despite these advancements modern hackers still exploit the programmable and open APIs of the SDN controllers through DDoS attacks. This chapter started with a recap on the evolution of security design considerations of the SDN controllers looking at quick mitigation techniques possible in the SDN platforms. With focus of this dissertation centered on the security impact on the SDN controller various tools and technologies used in the SDN network environment are introduced. Using the possibility of virtualization techniques to create separate compute virtual machines the SDN OpenDaylight controller is proposed for the PoC controller which communicates with a Mininet based data plane network emulation instance as separate VMs. OpenFlow is the protocol that is used for communication between the ODL and the data network elements namely the OpenFlow switches. Protocol based DDoS attacks by use of HTTP and ICMP flooding are proposed as the DDoS generating traffic sources. The chapter presented the SDN ODL controller security solution to construct a DDoS mitigation solution based on Linux iptables. The proposed iptables based solution is integrated to the SDN ODL controller infrastructure. The performance measurement methodology for the SDN controller infrastructure based on SNMP protocol is reviewed and how it is integrated to the SDN architecture is also explained. Lastly in the chapter the application call flows which shows series of messages and protocols exchanges between SDN elements and data planes through the security firewall are explained in preparation for measurement of performance results in the next chapter.

5 CHAPTER FIVE

EXPERIMENTAL PERFORMANCE EVALUATION AND VALIDATION

5.1 Chapter Overview

The previous chapter explained the various tools, protocols and technologies that are required to construct the PoC model for the proposed Linux iptables based network security solution for the OpenDaylight controller. This chapter will evaluate the effectiveness and efficiency of the security solution against DDoS attacks and the impact of the security solution on ODL controller performance KPIs. The effectiveness of the security solution is measured by verifying if the mitigation of the DDoS packets from a DDoS generating host is successful. The performance KPIs evaluation was done by observing the CPU, memory, response time and SDN network creation time whilst mitigating the DDoS attacks on the controller. To achieve the above measurements and verifications, three experiments were conducted as summarized in the application flow diagram in Figure 4.6. The first experiment was done by launching a Mininet network and connecting it to a remote ODL controller with iptables firewall disabled. This experiment was a key control as it was the benchmark test for the ODL controller without any security policies. The second experiment was done by repeating experiment 1 whilst iptables was now enabled. This experiment was to provide a comparable operating state for the ODL controller with iptables running without a DDoS attack. Lastly the third experiment was performed by repeating experiment 2 whilst DDoS attack was launched on the ODL controller. This last experiment became the DDoS load testing experiment which evaluates the impact of the security solution on the ODL controller. In each experiment the SDN network was setup, communication between elements verified and the ODL performance KPIs results captured for analysis.

5.2 Experimental Evaluation Scenarios

This section presents the results of the experiments that were done as described in section 5.1. In preparation for the collection of the results from the experiments the SNMP sessions were setup between VM1 and VM3 and launched to collect the data on the metrics to be used in the evaluation. The data that was captured by the SNMP sessions in experiment 1 and 2 becomes the data for the baseline control values that are compared when the ODL controller will be induced to load conditions.

5.2.1 DDoS attack launch and verification

In this section we introduce the DDoS attack source traffic establishment and verification that was used in experiment 3 alongside the mitigation. As explained earlier in section 4.3.5 the DDoS source will be launched from VM4 using hping3 tool. Hping3 is a security penetration tool that was used to simulate DDoS attacks by flooding SYN messages towards a target [30]. Build on the similar principles as ping of death hping3 is build and was used for both TCP/IP and UDP transport protocols. The hping3 based DDoS attack was initiated by the entering the command ***sudo hping3 –flood -S -p 80 10.254.0.1*** with the (parameters for the attack

explained in section 4.3.5). The flood flag is enabled and sending of the packets to the target (ODL controller 10.254.0.1) will be done as fast as possible. In the testing the hping3 tool was configured to send the packets from a static IP addresses (10.254.0.4) to enable synchronization and verifications of the matching rules in iptables.

```
carlton@carlton-DDoS-VM4:~$ sudo hping3 --flood -S -p 80 10.254.0.1
HPING 10.254.0.1 (eth0 10.254.0.1): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

Figure 0.1 *hping3 launch commands for DDoS attack initiation*

In the DDoS attack scenario launched in Figure 5.1, the hping3 tool installed on VM4 was initiated to flood the ODL controller on IP address 10.254.0.1 with SYN messages on port 80. This was done with the Wireshark tool collecting captures of the traffic in the background in VM1. The Figure 5.2 shows the attack detail information from the source host 10.254.0.4. The source IP for VM4 and destination IP for VM1, the attack port and SYN flags details extracted from the attack are shown in the Wireshark capture which confirms the generation of the DDoS traffic as launched by the hping3 tool.

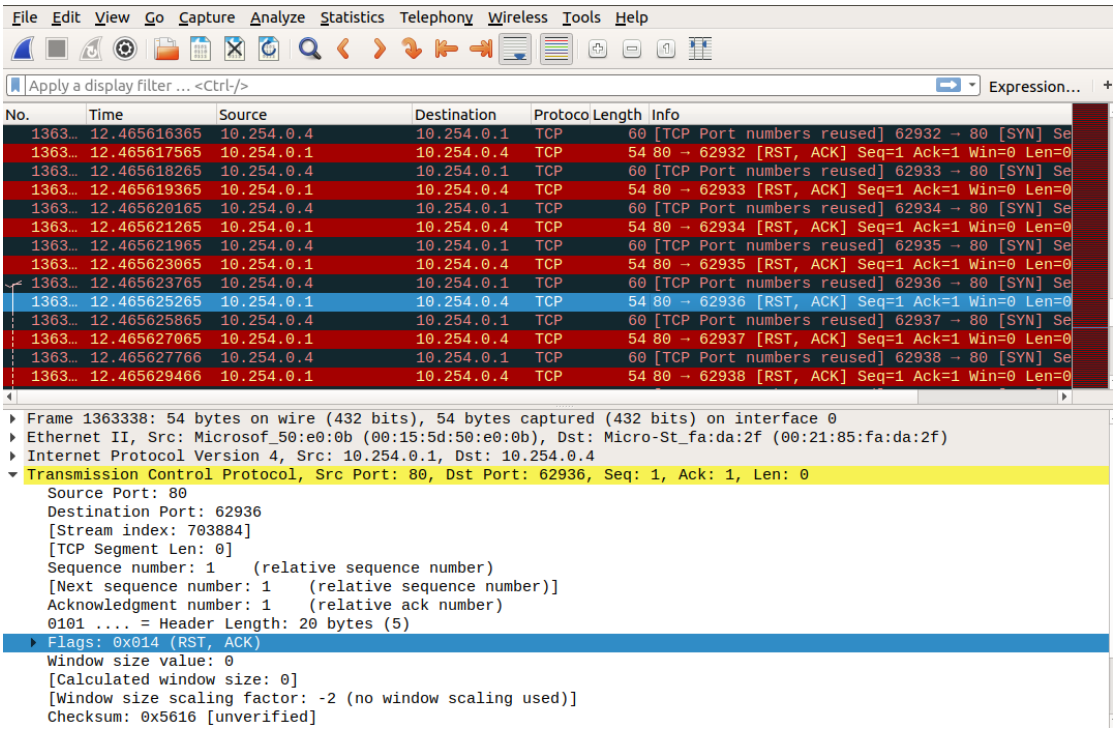


Figure 0.2 *Wireshark capture showing DDoS attack initiated from VM4 on flooding port 80 on 10.254.0.1 with SYN packets*

The generated DDoS traffic was measured on VM4 host system monitor and shows a spike in traffic of up to 4.8 MB/s throughput rate in both the receive and transmit directions as the DDoS attack launch was active. This is illustrated in Figures 5.3 and 5.4 for host VM1 and VM4 respectively. The traffic volume generated by the attack was captured on the victim's VM4

system monitor as well and corresponds to the source traffic which was measured on VM4 ethernet interface. The two graphs confirm that the DDoS traffic volume was generated at the time when hping3 based launch attack was launched.

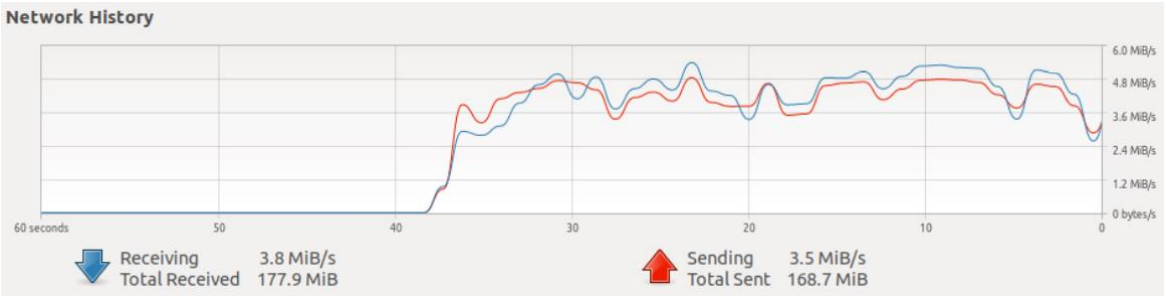


Figure 0.3 *Traffic statistics on VM1 showing DDoS traffic source on eth0*

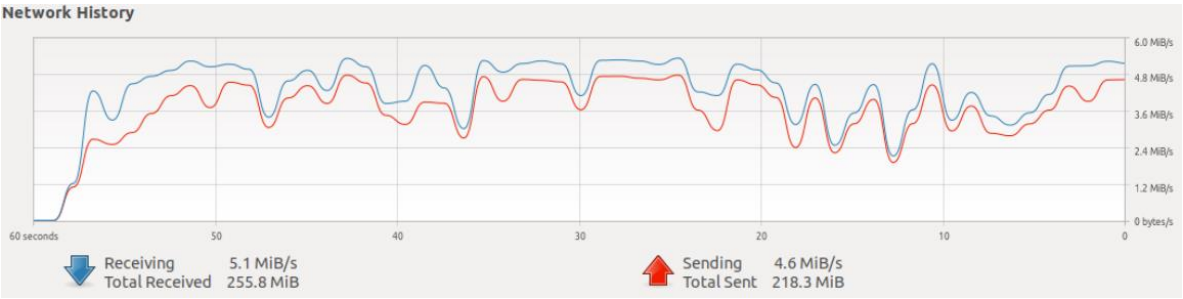


Figure 0.4 *Traffic statistics on vM4 showing traffic DDoS traffic*

5.2.2 DDoS mitigation functionality verification and measurements

In section 5.2.1 we exhibited the DDoS attack results and verified that the DDoS source attack traffic was indeed initiated from VM4 as per the design method and procedure. There was need to verify the mitigation script of the DDoS traffic that was implemented on the ODL host VM1 before the SDN based experiments were carried out. The mitigation script verification was done by enabling iptables policies on VM1, initiating the hping3 based DDoS attack presented in section 5.2.1 and verifying that the launched traffic is blocked by the firewall. The traffic generated on VM4 was also checked on the system monitor on the ethernet interface eth0. In addition, Wireshark captures were captured as shown in Figure 5.5 showing that the traffic was blocked as per firewall design rules. In the captures we see that the victim VM1 does not respond with a SYN ACK as in the scenario when iptables was not activated. This Wireshark based verification method used in this section was successfully implemented in [30] for the verification of mitigation of different types of DDoS attacks providing similar results to tests obtained in this section.

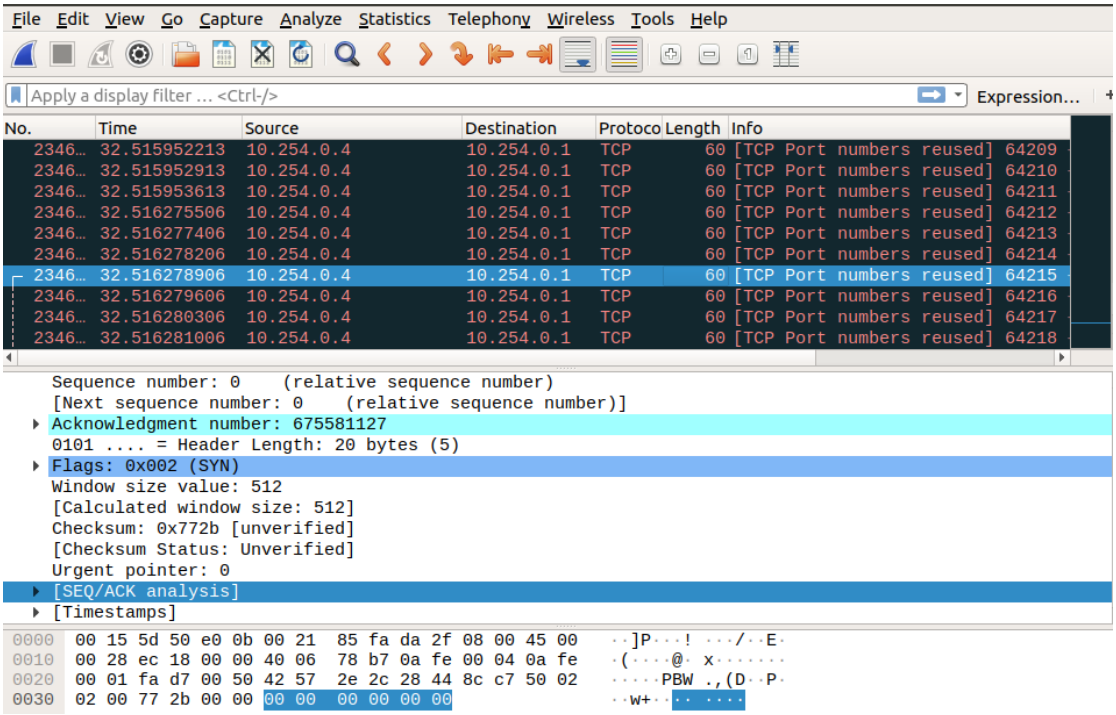


Figure 0.5 *Wireshark captures showing DDoS mitigation*

In addition to the Wireshark captures introduced earlier, traffic received on VM1 was measured and compared to traffic generated on VM4 during the attack launch times. This traffic measured on VM1 host was always comparable to traffic generated by the attack source with approximately less than 3% deviance margin. The traffic generated in the launch attack was about 4.8Mb/s as shown below.

Figure 0.6 *Traffic volumes captured on VM4 during the attack from VM1*

5.3 Iptables effect on ODL performance measurements

5.3.1 ODL performance benchmark testing

This first experiment was carried out with the firewall disabled and security layer excluded from the SDN controller network. The metrics were captured over a one-hour period and the time ***t*** when each activity was carried out was noted. At ***t°*** the SNMP session was activated

and was executed to run in the background capturing the CPU and memory KPIs. Correct operation of SNMP service was verified by the command ***service snmp status*** executed in the ODL VM1 host with results shown below in Figure 5.7 which shows that it was active. This step was crucial in validating that the SNMP server-client agent communication was active and running normally.

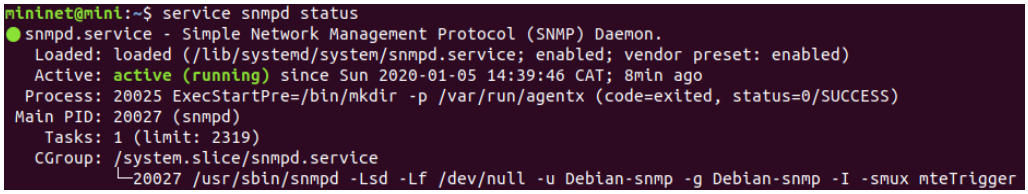


Figure 0.7 *SNMP status on VM1*

At the same time t°, the iptables firewall module was disabled by flushing all the iptables rules using ***sudo iptables -F.*** By executing the command ***sudo iptables status*** in the ODL VM1 host the iptables status is also verified. This was crucial in testing the ODL performance without any firewall rules being active. At this stage the DDoS flood mechanism from the hping3 tool is not launched as well. The firewall disabling results confirms that the firewall was inactive and no packet inspection of packets towards 10.254.0.1 was being done. At t^1 the ODL application was launched in VM1 host by activating the karaf application using the command ***sudo. /bin/karaf***. Output in Figure 5.8 below shows that the ODL controller was running and was listening on OpenFlow port for any connection attempts on the southbound API.

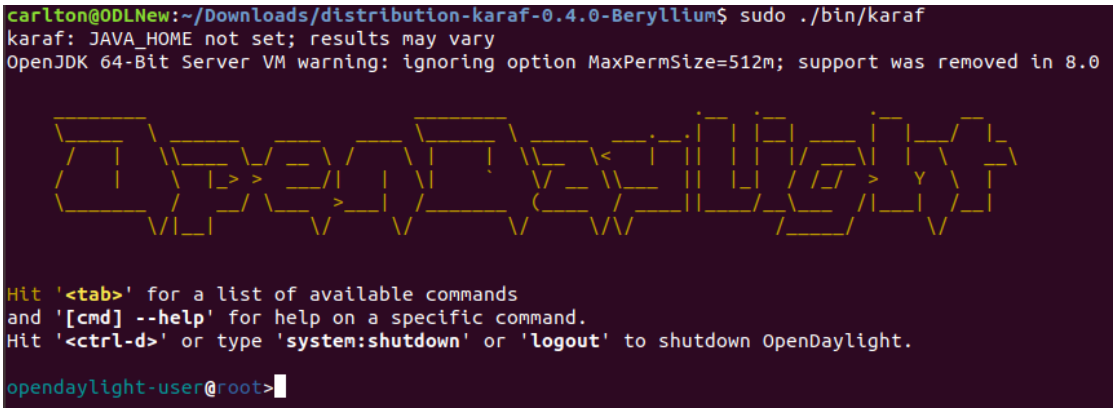


Figure 0.8 *Output from VM1 showing OpenDaylight application running*

With the iptables module disabled on VM1, at t^2 the Mininet network was launched in VM2 enabling communication between the OVS switches and remote OpenDaylight controller on VM1. The Mininet network was launched with parameters of a tree topology with 4 hosts, 3 switches connecting to a remote controller on IP address 10.254.0.1 on port 6633 as shown in Figure 5.9. The SDN network was created with the ODL controller and the topology, established links and hosts were verified in Mininet VM2.

```
mininet-VM:~/mininet/custom$ sudo mn --mac --topo=tree,2 --controller=remote,ip=10.254.0.1
*** Creating network
*** Adding controller
Connecting to remote controller at 10.254.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>
```

Figure 0.9 *Mininet launch parameters*

The verifications of the SDN network that was build was done by the Mininet commands dump, links, pingall, and net which all showed successful setup and normal operation of the SDN network. The results of these command tests are shown in Figure 5.10 ,5.11 ,5.12 and 5.13 respectively.

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=18875>
<Host h2: h2-eth0:10.0.0.2 pid=18877>
<Host h3: h3-eth0:10.0.0.3 pid=18879>
<Host h4: h4-eth0:10.0.0.4 pid=18881>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=18886>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=18889>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=18892>
<RemoteController{'ip': '10.254.0.1'} c0: 10.254.0.1:6653 pid=18869>
mininet>
```

Figure 0.10 *dump command results*

```
mininet> links
s1-eth1<->s2-eth3 (OK OK)
s1-eth2<->s3-eth3 (OK OK)
s2-eth1<->h1-eth0 (OK OK)
s2-eth2<->h2-eth0 (OK OK)
s3-eth1<->h3-eth0 (OK OK)
s3-eth2<->h4-eth0 (OK OK)
mininet>
```

Figure 0.11 *links command results*

```
mininet> net
h1 h1-eth0:s2-eth1
h2 h2-eth0:s2-eth2
h3 h3-eth0:s3-eth1
h4 h4-eth0:s3-eth2
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s3-eth3
s2 lo:  s2-eth1:h1-eth0 s2-eth2:h2-eth0 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s1-eth2
c0
mininet> links
```

Figure 0.12 *net command results*

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

Figure 0.13 *Pingall command results*

The ODL controller connected to the Mininet VM2 network and populated the topology through exchange of information through OpenFlow protocol. The SDN topology that was created by the controller was verified with ODL yang visualizer tool in Figure 4.5.

5.3.2 Benchmarking experiment results

Below are the graphs showing the CPU usage, memory utilization, response times and network setup latency respectively obtained from measurements of this experiment scenario described above in section 5.3.1. The measurements were extracted from the OpenNMS tool using the URL for the GUI on http://10.254.0.3:8980/opennms whilst the network setup latency was measured from the Mininet VM2 host command line. Below are captures of the results for the KPIs.

Figure 0.14 ***CPU usage statistics on ODL controller host for experiment 5.3.1***

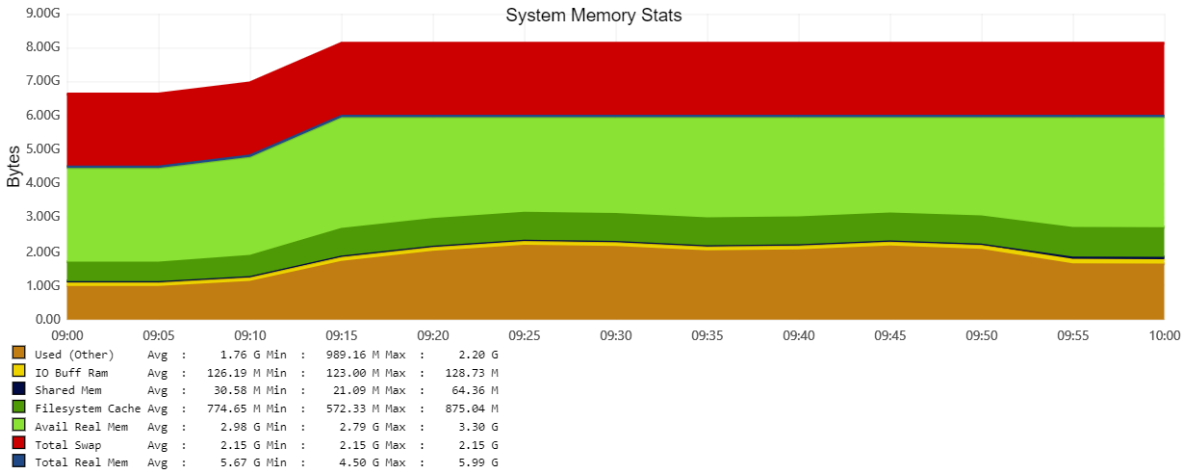


Figure 0.15 *Memory utilization of ODL host VM1 for experiment 5.3.1*

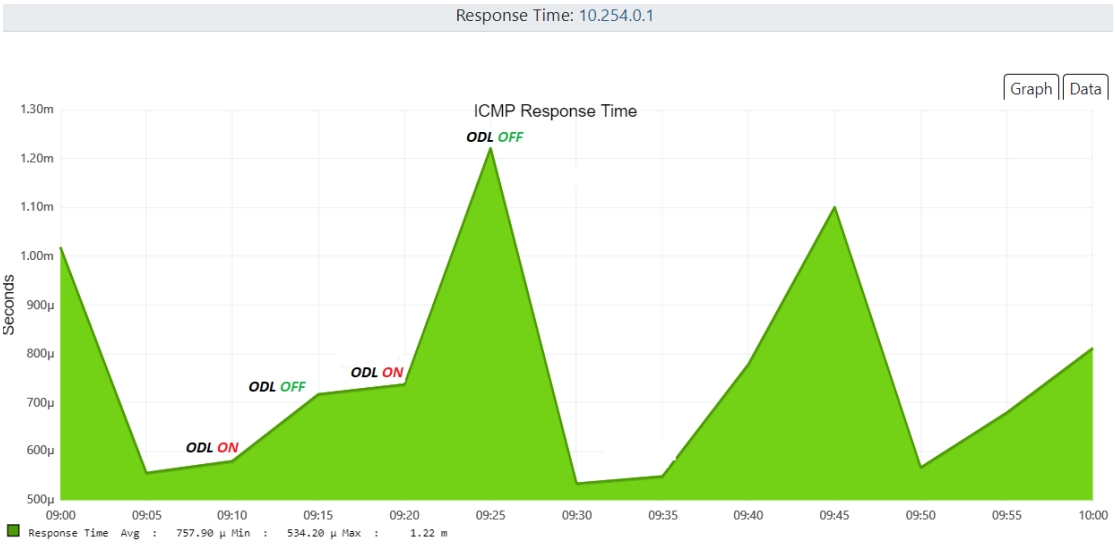


Figure 0.16 *Response times for the ODL host for experiment 5.3.1*

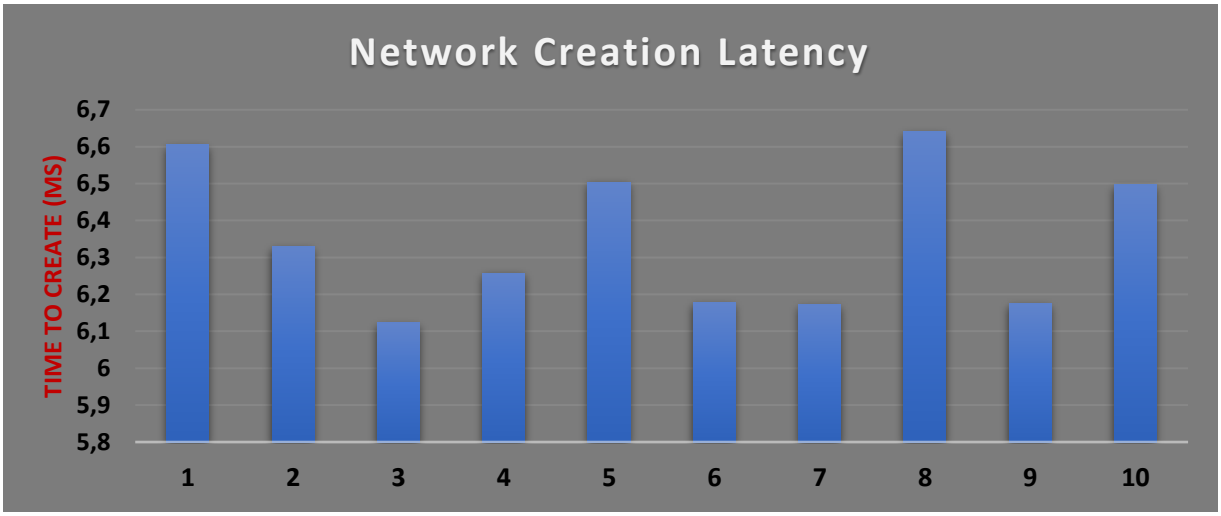


Figure 0.17 *Time in milliseconds for Mininet SDN network setup*

5.3.3 Analysis of results (Scenario 1)

In Figure 5.14 we see that the CPU usage graphs shows a spike in CPU from 30% to 50% utilization in response to the activation of the ODL application at t^1 and similar behavior on the successive ON states. This was the expected response as VM1 host responded to additional system loading introduced to the host by the activation of the ODL application in both instances. Similarly, both memory utilization and response time shown in Figure 5.15 and Figure 5.16 respectively show a spike in this parameter once the ODL application was launched due to more system loading on the host. In Figure 5.17 we see that the network creation times is between 6.1 -6.7 milliseconds which becomes the comparing value in the preceding tests when the DDoS attack is induced on the network.

5.4 Iptables effect on ODL controller measurement (No DDoS)

This experiment involved the repeating of experiment 1 whilst iptables was enabled on ODL VM1. To turn ON iptables the command ***sudo iptables-save*** was executed and the iptables rules were activated in the firewall. To confirm activation of the firewall and the contents of the rules table the command ***sudo iptables -L -v*** was issued. The figure below in 5.18 shows all the correct rules that were displayed and activated by the command.

```
carlton@ODLNew:/$ sudo iptables -L -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  eth0   any     10.254.0.2           ODLNew              tcp dpt:6633
    0     0 ACCEPT     tcp  --  eth0   any     10.254.0.2           ODLNew              tcp dpt:6653
    0     0 ACCEPT     tcp  --  eth0   any     10.254.0.2           ODLNew              tcp dpt:8181
    0     0 ACCEPT     tcp  --  eth0   any     10.254.0.3           ODLNew
    0     0 ACCEPT     udp  --  eth0   any     10.254.0.3           ODLNew
    0     0 DROP       tcp  --  eth0   any     10.254.0.4           ODLNew              tcp dpt:http

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  any    any     ODLNew               10.254.0.2
    0     0 ACCEPT     tcp  --  any    any     ODLNew               10.254.0.3
    0     0 ACCEPT     udp  --  any    any     ODLNew               10.254.0.3
    0     0 DROP       tcp  --  any    any     ODLNew               10.254.0.4          tcp dpt:http
carlton@ODLNew:/$
```

Figure 0.18 *Iptables policy rules table*

The rules that were displayed matched the whitelisting and blacklisting rules as listed in the design tables 4.3 and 4.4 introduced in chapter 4. Correct operation of rules for all the traffic classes defined in the tables 4.3 and 4.4 was verified by performing ICMP ping tests and telnet to the ODL OpenFlow port 6633 with results summarized in the Table 5.1. Steps carried out in the previous scenario in experiment 1 were carried out and measured whilst the firewall was online, and impact of the firewall on KPIs was recorded for analysis. The Mininet network creation times were tested in successive CLI based commands which involved the creation of the SDN network and the tearing it down processes. The ODL application was disabled and enabled 3 times over the one-hour period to see the response times of the host at each activation instance.

Table 0-1 **Service tests for traffic traversing the iptables firewall**

TESTING OF INPUT and OUTPUT Policies Operation							
Test No.	Source IP	DEST IP	Traffic case	Test Protocol	Port	Result	Comments
test 1	10.254.0.1	10.254.0.2	OUTPUT	Telnet/ICMP	6633	PASS	*Telnet on port 6633 successful,icmp test successful*
test 2	10.254.0.1	10.254.0.2	OUTPUT	Telnet/ICMP	8181	PASS	*Telnet on port 8181 successful,icmp test successful*
test 3	10.254.0.1	10.254.0.3	OUTPUT	Telnet/ICMP	161	PASS	*Telnet on port 161 successful,icmp test successful*
test 4	10.254.0.1	10.254.0.4	OUTPUT	Telnet/ICMP	8181	PASS	*Telnet on port 8181 successful,icmp test successful*
test 5	10.254.0.1	10.254.0.4	OUTPUT	Telnet/ICMP	80	PASS	*Telnet to 10.254.0.4 on port 80 failed*
test 6	10.254.0.1	10.254.0.4	OUTPUT	Telnet/ICMP		PASS	*Pings from host 10.254.0.1 to 10.254.0.4 not successful*
test 7	10.254.0.2	10.254.0.1	INPUT	Telnet/ICMP	6633	PASS	*Telnet on port 6633 successful,icmp test successful*
test 8	10.254.0.2	10.254.0.1	INPUT	Telnet/ICMP	8181	PASS	*Telnet on port 8181 successful,icmp test successful*
test 9	10.254.0.3	10.254.0.1	INPUT	Telnet/ICMP	161	PASS	*Telnet on port 161 successful,icmp test successful*
test 10	10.254.0.4	10.254.0.1	INPUT	Telnet/ICMP	8181	PASS	*Telnet on port 8181 not successful,icmp test fail*
test 11	10.254.0.4	10.254.0.1	INPUT	Telnet/ICMP	80	PASS	*Telnet on port 80 not successful,icmp test fail*
test 12	10.254.0.4	10.254.0.1	INPUT	Telnet/ICMP		PASS	*Pings from host 10.254.0.4 to 10.254.0.1 not successful*

5.4.1 Results of the experiment

This experimental test in this section involved the activation of the ODL controller application 3 times in an hour period at 0910hrs, 0930hrs and 0945hrs. The 4 core KPIs CPU, memory utilization and response times being investigated under the scope of the dissertation were captured. Results for this test were extracted similarly on the OpenNMS GUI using URL shared earlier. The SDN application latency was measured by executing commands in the SDN network VM2 host. Below are the captures of the results obtained in this test. In all the test cases we focus and observe the response of our KPIs from the ODL activation and DDoS effect on the targeted nodes.

.

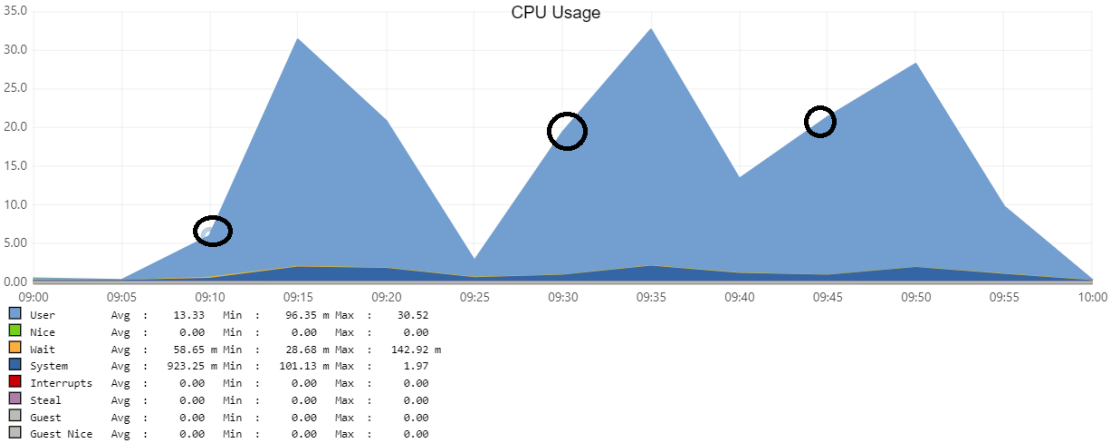


Figure 0.19 *CPU extracted with iptables turned ON experiment 2*

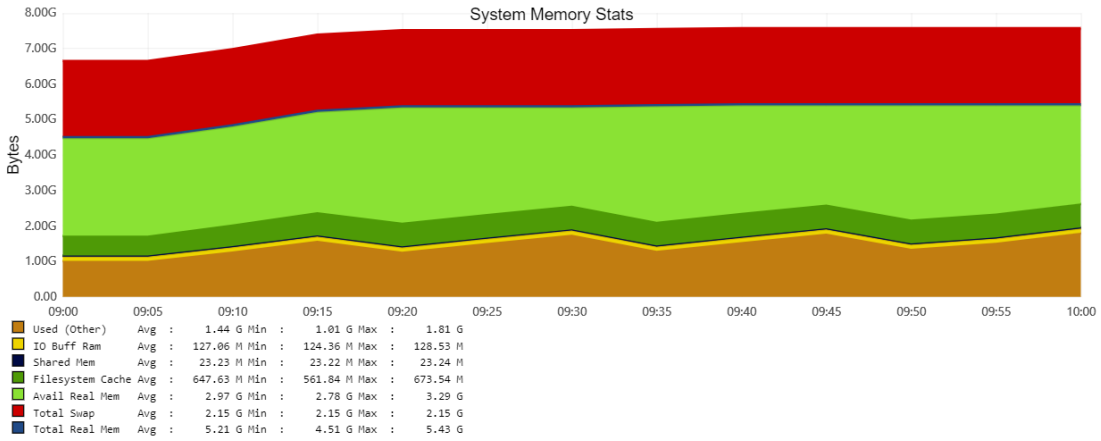


Figure 0.20 *Memory utilisation experiment 2*

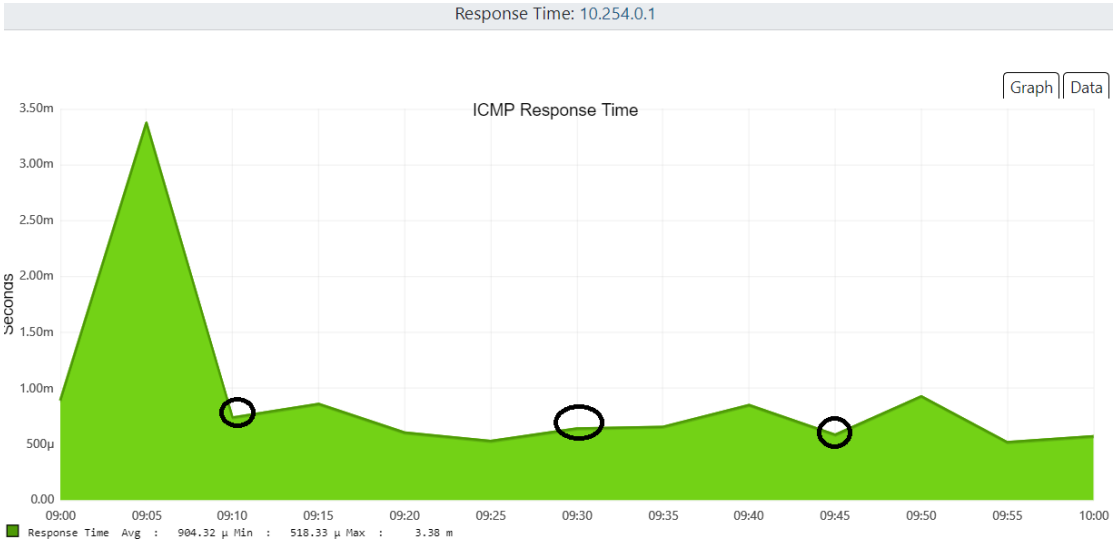


Figure 0.21 *Response times for experiment 2*

5.4.2 Analysis of results

Results from this experiment shows an increase in CPU usage, memory utilization and response times when the ODL application is launched in all the 3 times. This is the expected response as the CPU spike is triggered by the JAVA processes that the ODL application uses with heavy impact of VM1 resources. The CPU utilization increase observed in this experiment with the launch of the 3 OVS switches ranges between 40-50% as obtained by authors in [47] as shown in Figure 5.23 below. The memory rise in the experiment peaked at 7.5-7.9 G and didn't drop for the duration of the testing period which is similar to results obtained in [47] an shown in Figure 5.22 as well. Memory is only reclaimed by execution of a garbage collector process which releases the memory as was done by the authors in [47] shown by the blue arrow in Figure 5.22. For this experiment the memory reclaiming process was not initiated.

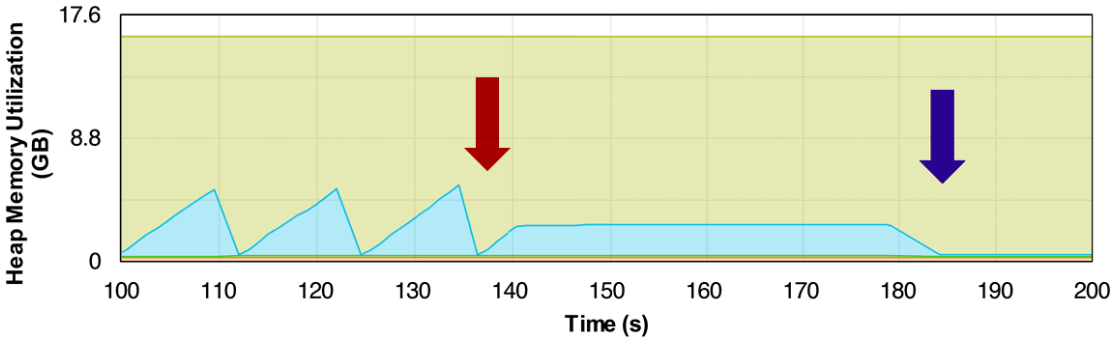


Figure 0.22 *OpenDaylight memory utilization* **[47]**

Comparison of results in this test with the preceding experiment show that the iptables activation of the firewall didn't affect the ODL CPU, memory and response times for the network topology that was launched.

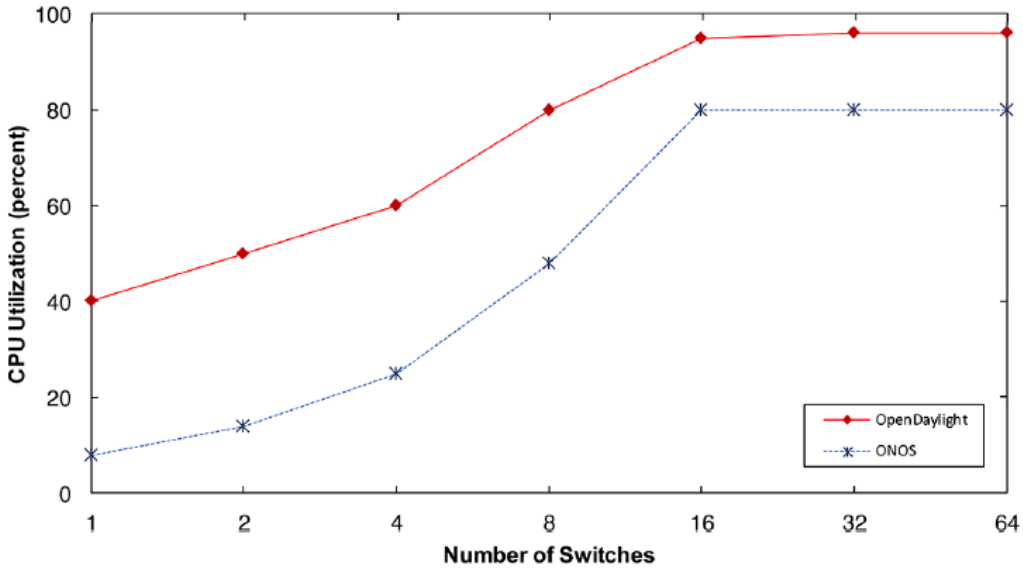
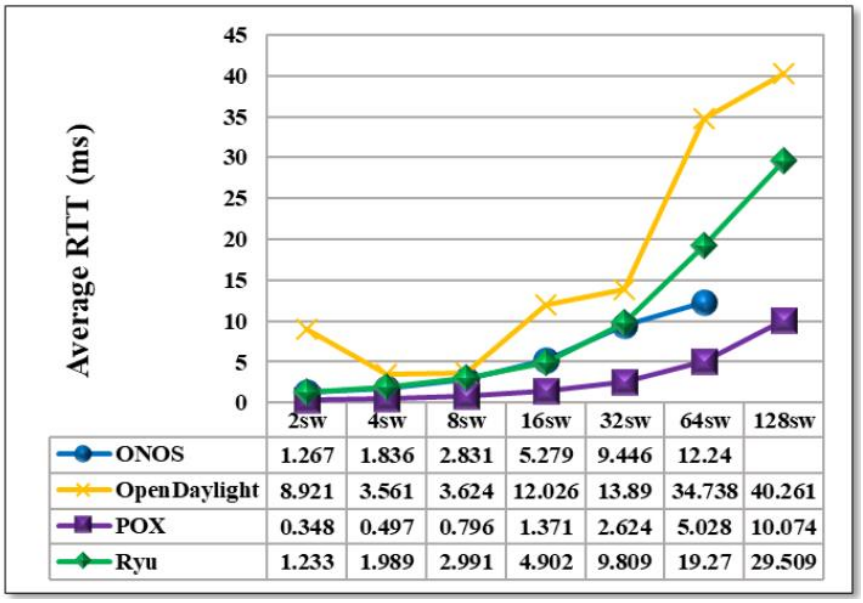


Figure 0.23 *Impact of number of switches on CPU* **[44]**

The SDN application latency is also maintained between 6-6.9ms which shows minimal deviation to results when iptables was disabled in section 5.3.2. In summary the proposed firewall has no impact on KPI performance of the SDN controller given the additional overhead of packet inspection that it introduces when the firewall is running. The OpenDaylight controller application latency of 8.9ms was obtained in experiments done in [44] as shown in Figure 5.24 which is closely comparable to the results we obtained from this for the same number of OVS switches in the topology.



	2sw	4sw	8sw	16sw	32sw	64sw	128sw
ONOS	1.267	1.836	2.831	5.279	9.446	12.24	
OpenDaylight	8.921	3.561	3.624	12.026	13.89	34.738	40.261
POX	0.348	0.497	0.796	1.371	2.624	5.028	10.074
Ryu	1.233	1.989	2.991	4.902	9.809	19.27	29.509

Figure 0.24 *Latency comparison for the same number of switches* **[26]**

5.5 DDoS mitigation on ODL performance load testing and measurement

This scenario was a final test that will determined the full impact of DDoS mitigation and determines the effectiveness of the iptables solution deployed on the ODL controller. The preconditions for this test was the enabling of the SNMP service and activation of the iptables firewall in VM1 as done in scenario 2. The test involved the DDoS attack launch on the ODL VM1 and the mitigation of the DDoS attack by iptables as tested in section 5.2.2. Whilst the DDoS attack was active on VM1 the Mininet network was launched in successive tests using the same SDN parameters used in both scenario 1 and 2 and the KPIs were measured.

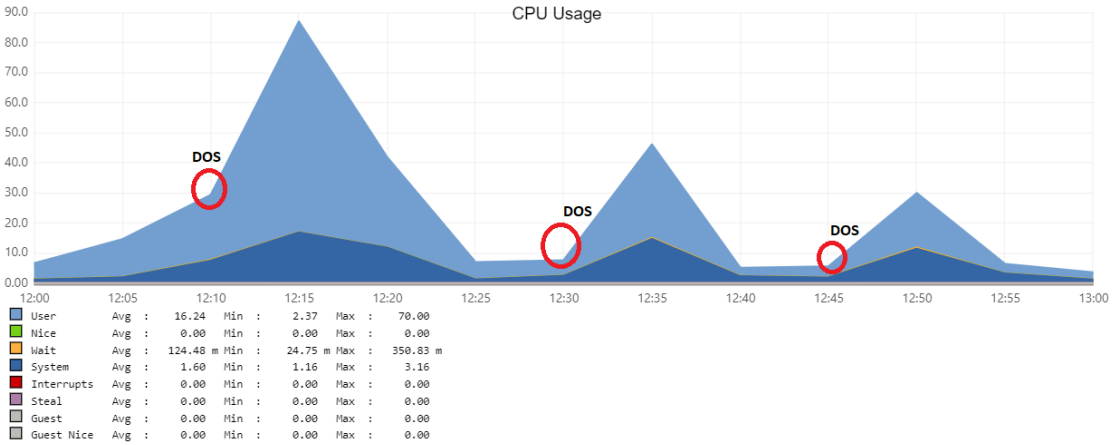


Figure 0.25 *CPU usage with DDoS active and mitigation active*

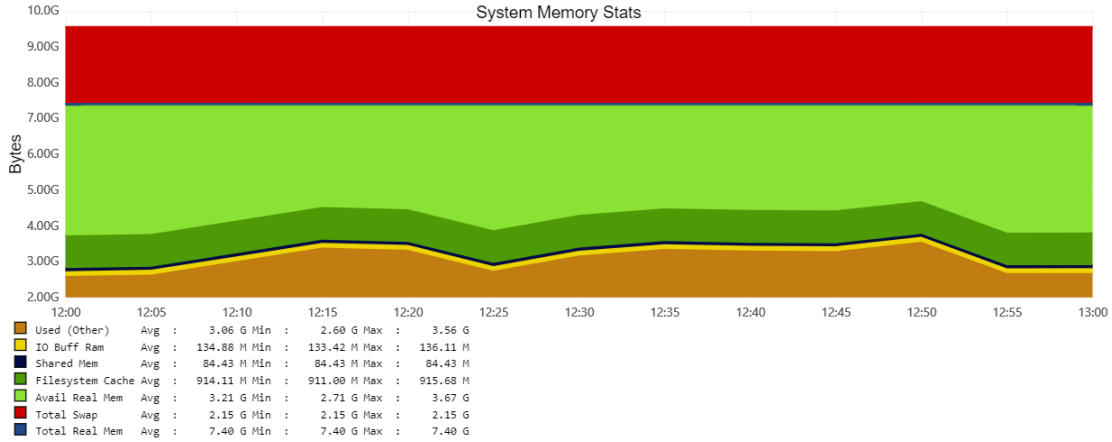


Figure 0.26 *Memory utilization with DDoS active and firewall online*

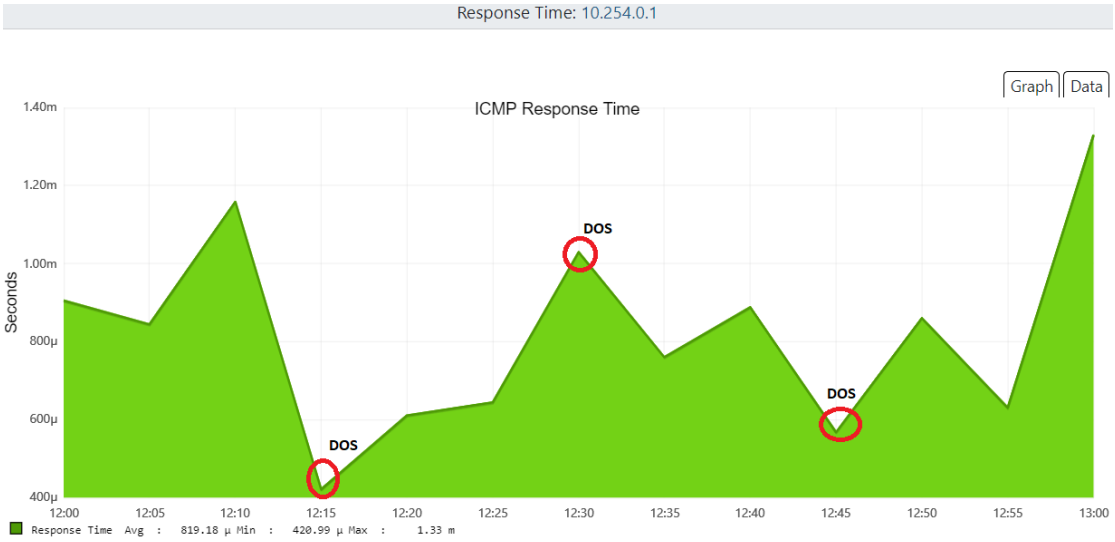


Figure 0.27 *Response times with DDoS active and firewall online*

5.5.1 Analysis of DDoS mitigation impact on ODL CPU, Memory and response rate

Figure 5.24 through to Figure 5.26 show results that were obtained from this experiment. With DDoS attack launched on VM1 and mitigation active the CPU usage spiked by an average of 300% over the testing period as shown in Figure 5.25. Memory utilization in this test remained very high like results obtained in [47] since memory reclaiming was not done. The response rate increased as well though it was under 1ms for the duration of the DDoS attack as shown on Figure 5.27. Compared to CPU loading in experiment 2 we see that all the KPIs were comparable to the results obtained in the previous scenario.

5.5.2 Impact of iptables firewall on SDN network southbound API Latency results

Latency results shows an average latency of 8.98ms as compared to 6.34ms from the previous two experiments. The increased response times is linked to the additional system loading and overhead introduced by DDoS mitigation. Additional packet inspection tasks are being performed by the iptables for all packets entering and leaving the VM1 host for whitelisting and backlisting determination. The iptables netfilter introduces an extra hop that all packets need to traverse hence the increased latency.

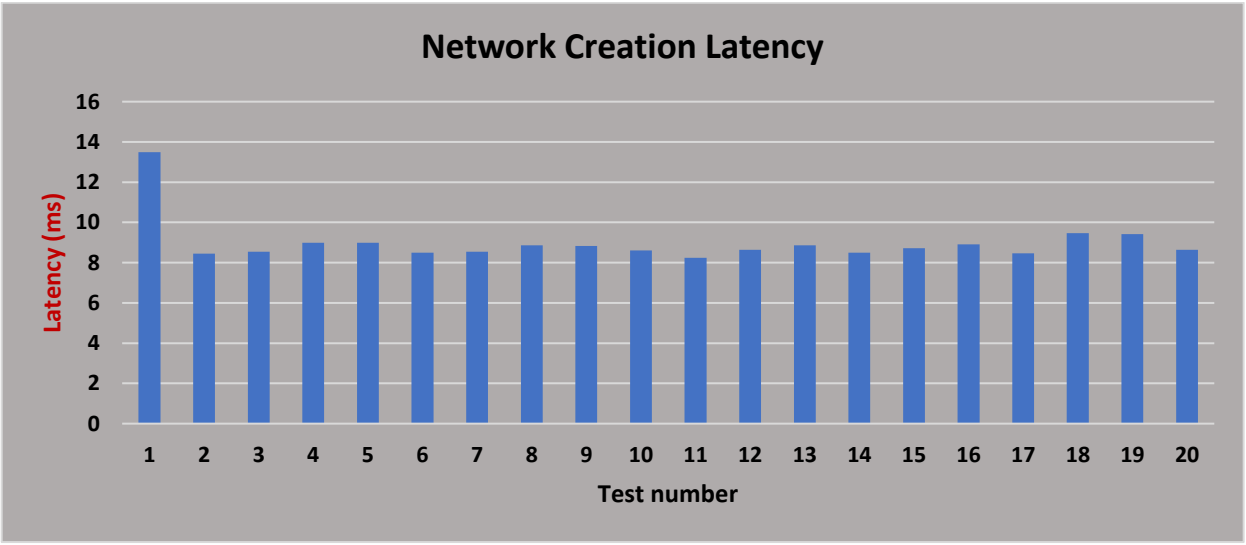


Figure 0.28 *Southbound API network creation time*

Reference was made to other packet filtering experiments done in [45] which shows that the added packet filtering functionality introduced by the firewalls increases latencies between applications. Figure 5.29 below extracted from [47] shows the same phenomenon as the HTTP response times are affected by packet filtering activities in firewall. This behavior is similar to the results obtained in our experiment with increased latency but no service degradation to all SDN based services.

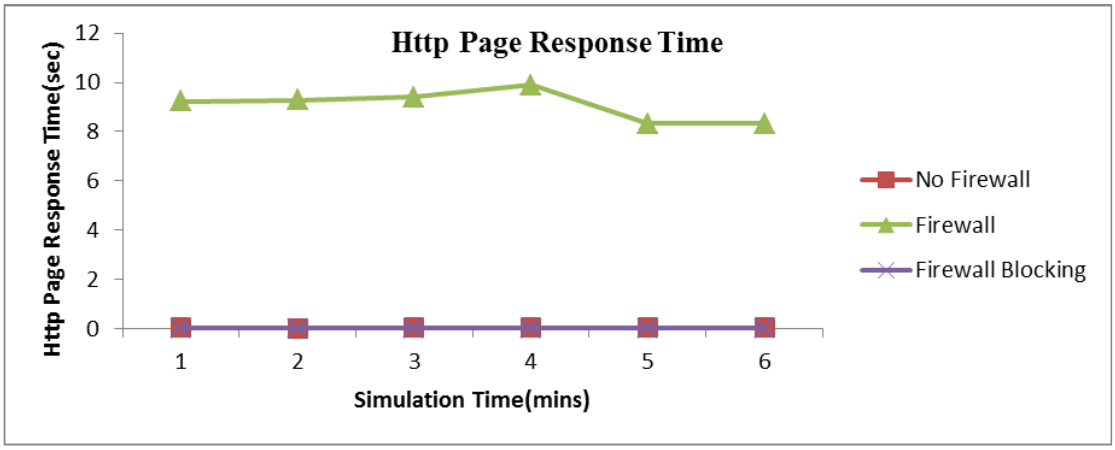


Figure 0.29 *Firewall impact on latency assessment* **[26]**

In addition to the 4 core KPIs 2 additional KPIs namely average load and TCP errors or failures were captured to support findings from results provided in 5.5.1. Analyzing these two additional KPIs we see that the average system load for the VM1 host increased with activation of the ODL application and mitigation of the DDoS traffic. The system load increased by an average of at least 100% percent showing the negative impact of the DDoS mitigation process on the host resources. Given that the iptables firewall works by sending TCP reset of the connections to the DDoS attack host we see in Figure 5.31 that the established TCP resets and resets sent counters increased proportionally with all other KPIs due to DDoS blocking.

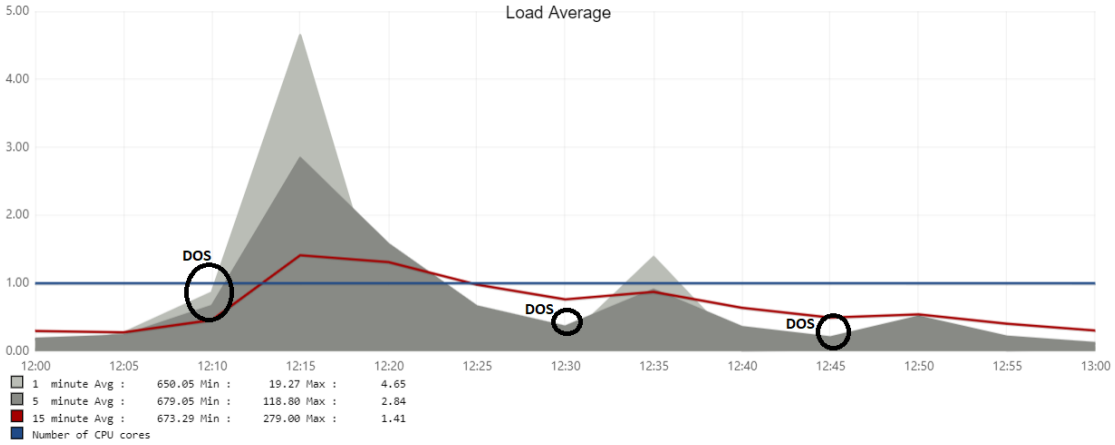


Figure 0.30 *Average system load KPI for the SDN controller*

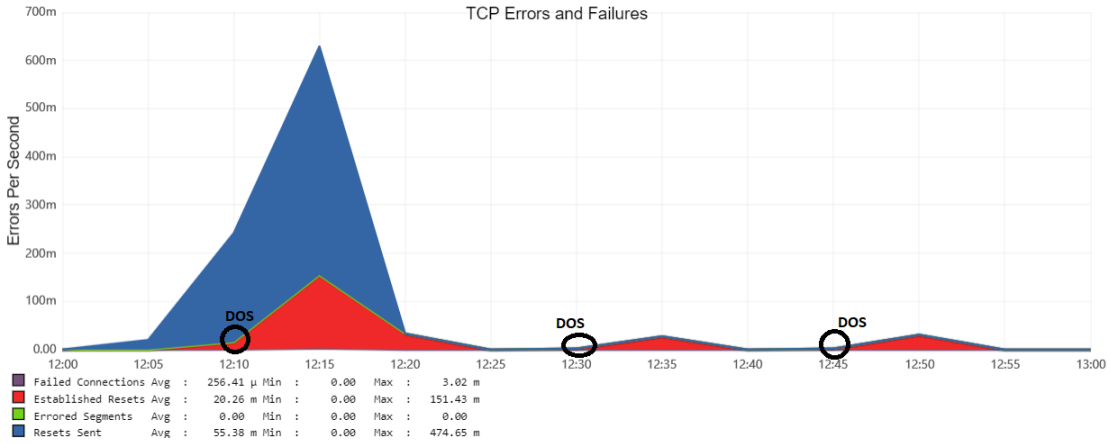


Figure 0.31 *TCP errors and failures during mitigation*

5.6 Chapter Summary

This chapter presented the experimental results obtained from the iptables firewall functionality verification, DDoS attack mitigation verification and impact of the DDoS attack on the SDN controller performance KPIs.

The iptables firewall functionality was tested by use of the ping tool which verified the TCP/IP communication between the VMs as defined in the iptables rules-set database. All ping tests were successful as per the allow or block policies defined in the firewall. The egress and ingress traffic tests matched the policies which confirmed the correct operation of the firewall. With the iptables rules tested successfully the DDoS mitigation was tested. The DDoS attack traffic launched from VM4 was verified by Wireshark traces and ethernet interface packet monitoring from the generating host. Analysis of the Wireshark captures was done and as expected they contained the correct DDoS attack parameters with the attack exploiting TCP/IP SYN packets on port 80 on the ODL VM1.

To confirm DDoS mitigation by the iptables rules the Wireshark tool was used to analyze the packets between VM1 and VM4 which showed that VM1 was sending TCP reset flags to the DDoS source when blocking the traffic. This was the expected result as per the TCP/IP 3-way handshake principles with reset replies from VM1 host closing the all sessions that VM4 attempts to open with it.

The impact of DDoS mitigation on SDN controller performance was investigated and measured by use of Opensource SNMP protocol-based software. The performance of the SDN controller was benchmarked by measuring the CPU utilization, memory utilization, response times and SDN application latency from the Mininet host without external loading. Results show that the CPU utilization and memory utilization was mostly affected by launching of ODL application on VM1 by at least 200%. This was followed by a corresponding increase in response times of the SDN controller host as it was exposed to additional load. These results obtained were similar to other benchmarking tests done using cbench tool in [44] showing a direct

relationship between CPU, memory utilization and response times. The SDN application latency was measured without the DDoS attack as a benchmark with a tree topology network with 3 OVS switches and 4 hosts. For this topology it was determined that the average latency for communications with the remote controller was 6.34ms which was comparable to tests done in Mininet with the reference controller. The results showed that the activation of the iptables firewall had no impact on the KPIs above without the DDoS attack present. With DDoS attack induced on the controller the first 3 KPIs followed results of the first two experiments and in summary they showed a direct relationship between CPU level, memory utilization and response times. The SDN latency was different as it increased to 8.98ms as the iptables was mitigating the DDoS attack. This observation is expected as the firewall introduces extra overheads by performing packet filtering tasks.

In summary the DDoS attack mitigation by iptables is possible and we see a direct proportional increase in controller CPU, memory utilization and response rate when mitigation is active. The increase in the above KPIs didn't cause severe service degradation of the SDN applications. The SDN latency was however affected and increased by the additional segment introduced by the firewall. Due to resourcing constraints the network setup could not be scaled up with more switches and hosts to determine the behavior of the KPIs in large SDN network deployments.

6 CHAPTER 6

CONCLUSION

6.1 Chapter Overview

In this chapter, a summary of the research work done in the first five chapters is presented and the findings of the experimental evaluations and results are discussed. The chapter will suggest areas of further research on iptables based security design considerations for the SDN controller as future work.

6.2 Conclusion

Software Defined Networking is emerging as a new networking technology offering Telecoms' operators many new features and capabilities with benefits emanating from having a programmable and a dynamically configurable network [27]. The technology involves the decoupling of the control function from packet forwarding module which makes the packet forwarding of packets more efficient and faster whilst enabling the centralization of infrastructure and equipment management [1]. This evolution in the networking field was inevitable as new use cases in 5G networks and network functions virtualization technology demands a programmable and highly dynamic network platform which the SDN platforms supports.

However, with the many advantages that the SDN technology provides the centralized nature of the SDN architecture and its programmability exposes it to many security threats. Security attacks are subjected on both the control and data plane of the SDN architecture, but many attacks are targeted towards the centralized SDN controller with the distributed denial of service attack (DDoS) being the most exploited attack method. This dissertation shows the possibility of using Linux netfilter firewall with iptables for DDoS mitigation attacks on the SDN controller. With the iptables firewall coexisting with the SDN controller infrastructure the dissertation evaluates the impact of the extra functionality on the SDN controller core functions for input to secure SDN controller and system sizing design considerations.

The dissertation firstly explores the SDN architecture components with focus on the SDN controller southbound interface, the OpenFlow protocol and the related message exchanges that occur in the setting up of the SDN network. This exploration was then followed by a review of the security threat vectors affecting the control plane, data plane network and application plane of the SDN architecture. With focus on the SDN controller the attack types and sources were analyzed, and more focus was put on DDoS based attacks which became the attack methodology used in the proof of concept implementation for the firewall security design. A review of SDN security solutions for the SDN controller was done with the requirements and limitations of each method analyzed. The summary of the analysis of the current SDN security solutions proposed by many authors required an external intrusion

detection system integration or distributed SDN controller deployments which are costly to Telecoms operators due to added duplicated hardware and dedicated security appliances.

In the dissertation a Linux netfilter based firewall is proposed with it coexisting with the SDN controller application to mitigate DDoS based attacks on the southbound interface. The proposed security solution considered the possible cost reduction or elimination of an external mitigation appliance and the eradication of proprietary reliance of the current SDN solutions. A DDoS mitigation requirement analysis is done, and deployment framework is proposed with the block rules and mitigation algorithm for the attacks. These algorithms are constructed by blacklisting SYN attacks on port 80 from the attack source host whilst whitelisting OpenFlow protocol port 6633 for normal SDN network establishment. The iptables rules database comprises of other whitelisted rules supporting functionality of SNMP based SDN controller performance monitoring tool and the SDN controller GUI interface.

A proof of concept testbed is setup to test the proposed Linux based firewall solution. The testbed comprises of 4 integrated modules namely the OpenDaylight SDN controller with the iptables based firewall, the Mininet module for packet forwarding network, the performance tracking module (NMS) using OpenNMS software and the DDoS traffic source traffic using the hping3 tool. The modules are created as independent virtual machines and interconnected by use of a virtual LAN switch for TCP/IP communication between the hosts. The Mininet OVS switches communicates with the OpenDaylight remote controller using OpenFlow on the southbound interface. The NMS periodically polls the virtual machine hosting the controller using SNMP protocol. The 4 modules are all deployed by use of free opensource tools and software using Linux operating system by use of Microsoft Hyper-V virtualization tool. The opensource tools are setup and a series of experimental tests are done. The experiments target findings on effectiveness of iptables proposed solution in DDoS mitigation and the impact of the mitigation on the SDN controller performance. The CPU, memory utilization and response times KPIs are measured by the OpenNMS tool and resource graphs are setup for the traffic counters and captures. The southbound interface latency metrics are captured by simulating SDN network creation through commands launched from the Mininet virtual machine that communicates with the external remote controller. In the experiments we see that SYN flooding can be mitigated by iptables with assumption that the attack source IP addresses are static. Analysis of KPI results show that the CPU utilization, memory utilization and response times are all affected negatively by the ODL application and the DDoS mitigation by iptables firewall. CPU utilization and system load counters always show an average of at least 200% increase when mitigating rules is affected. The SDN network latency was also affected marginally with a 2-millisecond extra delay introduced in comparison to the benchmark test with no firewall mitigation running.

Although the KPIs are negatively affected no service and application degradation was experienced from the tests that were done on the virtual machine hosting the ODL application which furthers supports the effectiveness of the iptables based firewall solution.

6.3 Future work

6.3.1 DDoS Detection algorithms

The DDoS traffic detection and design in iptables before the blocking was not covered in this dissertation. The DDoS mitigation demonstrated in the research was done with the attack source IP addresses already known by iptables firewall. As part of future work DDoS detection algorithms and policies in iptables can be implemented with attack source IP masquerading as the compromised host's IP in forms of Smurf or man in the middle attack. With this algorithm present in the iptables firewall the DDoS attacks can be predicted without need for an external security appliance which reduces deployment costs. The DDoS attack source which was used in the experiments was based on hping3 kali free software tool. Other DDoS source generating tools like LOIC and HULK can be tested with this solution and verify if similar results are obtained. Analyzing the effect of increasing the number of policies on the controller to overall network throughput and device processing capability is proposed as part of this scope on DDoS detection algorithms as it could not be tested within this research.

6.3.2 Packet forwarder Network size

Due to limited computing resources, RAM and CPU cores for the laptops performing the experiments and simulations the network sizes for Mininet was restricted to 3 OVS switches and 4 hosts only. As part of future work the network size and topology's impact on latency introduced by the iptables solution on southbound API can be investigated. This test of a large network will suffice as a decent benchmarking process as increased number of OVS switches will approach near real-life Telecoms operators network sizes and designs.

6.4 Chapter Summary

This chapter has given a summary overview of the scope of work and what was covered in the dissertation. It presented the benefits of the SDN technology architecture, the security threat vectors targeting the architecture and proposed a security solution for mitigating the DDoS, the prevalent attack. Future work in DDoS attack detection mechanisms and packet forwarder network sizing is suggested as areas requiring more research [27] as the security designs for SDN based networks keeps evolving.

7 References

[1] F. Prof. Dr. T. Magedanz, "NFV Start up - Network Technology Evolution," Institute FOKUS/TU Berlin, Germany, pp. 27-29, 2017.

[2] A. L. Vice-Chair, "Security Risks in SDN and Other New Software Issues," Frost & Sullivan, no. Application Security, isc2.org Sr Cybersecurity Advisor, pp. 654-656, 2017.

[3] P. Kampanakis, "SDN-based solutions for Moving Target Defense network protection," Security Research & Operations, Cisco Systems, USA, Harry Perros-Computer Science Department, NC State University,Tsegereda Beyene -SP Infrastructure & Wireless, pp. 1-2, 2014.

[4] E. N. Hamza Shaban, "The Switch: Pharmaceutical giant rocked by ransomware attack," June 2017. [Online]. Available: https://www.washingtonpost.com/news/the-switch/wp/2017/06/27/pharmaceutical-giant-rocked-by-ransomware-attack/. [Accessed 2018].

[5] Christian Fuchs, "Implications of deep packet inspection (DPI) internet surveillance for society," Department of Informatics and Media, Uppsala University, 2012.

[6] G. X. Dimitra Sakellaropoulou, "MSc. Thesis: A Qualitative Study of SDN Controllers," vol. Athens, pp. 49-52, 2017.

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

[8] LoritPro v5, "The SNMP concepts," https://www.loriotpro.com/Products/On-line_Documentation_V5/LoriotProDoc_EN/C3-Introduction_to_Network_Supervision/C3-B2_SNMP_Concepts_EN.htm.[Accessed 2019]

[9] Fernando M.V. Ramos , Diego Kreutz, Paulo Verissimo, "Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking" pp. 55–60, August 2013.

[10] F. H. M. I. M. H. D. M. W.] Lundkwhar Mardan, "A Simple Security Policy Enforcement System for an Institution Using SDN Controller Department of Computer and Information Sciences," Pakistan Institute of Engineering and Applied Sciences, pp. 1-5, 2018.

[11] O. Salman, I. H. Elhajj, A. Kayssi and A. Chehab, "SDN controllers: A comparative study," 2016 18th Mediterranean Electrotechnical Conference (MELECON), Lemesos, 2016, pp. 1-6.

[12] K. Kalkan and S. Zeadally, "Securing Internet of Things with Software Defined Networking," in IEEE Communications Magazine, vol. 56, no. 9, pp. 186-192, Sept. 2018

[13] T. O. N. Foundation, "OpenFlow Switch Specification Version 1.4.0 (Wire Protocol 0x05)," 03 October 2014. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf. [Accessed 30 September 2019].

[14] D. Bombal, "GNS3, SDN and OpenFlow," Open Network Foundation, 15 February 2017. [Online]. Available: https://overlaid.net/2017/02/15/openflow-basic-concepts-and-theory/. [Accessed 11 August 2019].

[15] R. Toghraee, "Learning Open Daylight, the art of deploying successful networks," in Mumbai, India, Birmingham, UK : Packt Publishing, 2017.

[16] O. Project, "Virtual Tenant Network (VTN)," docs.opendaylight.org, 2016-2018. https://docs.opendaylight.org/en/stable-fluorine/user-guide/virtual-tenant-network-(vtn).html. [Accessed May 2019].

[17] M. Dabbagh, B. Hamdaoui, M. Guizani and A. Rayes, "Software-defined networking security: pros and cons," in IEEE Communications Magazine, vol. 53, no. 6, pp. 73-79, June 2015.

[18] Chao Qi et al., "An intensive security architecture with multi-controller for SDN," 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), San Francisco, CA, 2016, pp. 401-402.

[19] Fan Jiang, Chen Song, Hao Xun, Zhen Xu, "Combat-Sniff: A Comprehensive Countermeasure to Resist Data Plane Eavesdropping in Software-Defined Networks" 2016 American Journal of Networks and Communications, pp. 27-34, April 2016.

[20] A. Imran, "SDN Controllers Security Issues," University of Jyväskylä, Department of Mathematical Information Technology, MS Thesis: Web Intelligence and Service, pp. 33-46, 2017.

[21] C. Bouras, A. Kollia and A. Papazois, "Teaching network security in mobile 5G using ONOS SDN controller," 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN), Milan, 2017, pp. 465-470.

[22] Curtis Yu, Cristian Lumezanu, Abhishek Sharma, Qiang Xu, Guofei Jiang, Harsha V. Madhyastha, "Software-defined Latency Monitoring in Data Center Networks," University of California, Riverside, NEC Labs America, pp. 1-12, 2015.

[23] Mahmood Z. Abdullah, Nasir A. Al-awad, Fatima W. Hussein, "Evaluating and Comparing the Performance of Using Multiple Controllers in Software Defined Networks," Al-Mustansiriyah University/College of Engineering/ Computer Engineering Department, Baghdad, pp. 45-50 , 2017.

[24] Liehuang Zhu, Md Monjurul Karim, Kashif Sharif, Fan Li, Xiaojiang Du, Mohsen Guizani, "SDN Controllers: Benchmarking & Performance Evaluation," arxiv.org, pp. 1-12, 2019.

[25] Spooner, Jakob Zhu, Shao Ying, "A Review of Solutions for SDN-Exclusive Security Issues," Department of Computing and Mathematics, University of Derby, England, pp. 113-120, 2016.

[26] Francis Kwadzo Agbenyegah, Michael Asante, "Impact of Firewall on Network Performance," International Journal of Scientific and Technology Research , pp. 32-36, 2017.

[27] L. Dridi and M. F. Zhani, "SDN-Guard: DoS Attacks Mitigation in SDN Networks," 2016 5th IEEE International Conference on Cloud Networking (Cloudnet), Pisa, 2016, pp. 212-217

[28] G. Pandya, "Preparing to withstand a DDoS Attack," Information Security Reading Room ,SANS Institute, pp. 3-5, 2015.

[29] Hassan Ahmed Khan, Sufian Hameed, "SDN Based Collaborative Scheme for Mitigation of DDoS Attacks," IT Security Labs, National University of Computer and Emerging Sciences, pp. 2-11, 2017.

[30] Bahaa Qasim, M. Al-musawi, "Mitigating DDoS Attacks using Iptables," College of Engineering University of Kufa, An Najaf, Iraq, pp. 101-108, 2012.

[31] C. Mellon, "CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks," University Software Engineering Institute, pp. 160-166, 2013.

[32] S. Scott-Hayward, "Trailing the Snail: SDN Controller Security Evolution," Centre for Secure Information Technologies (CSIT), Queen's University Belfast, Belfast, BT3 9DT, N. Ireland, pp. 1-6, 2017. .

[33] N. Chakraborty, "Intrusion detection system and intrusion prevention system," School of Future Studies & Planning, Devi Ahilya University, Indore, India, no. A comparative study, pp. 2-4, 2013.

[34] S. Sezer et al., "Are we ready for SDN? Implementation challenges for software-defined networks," in IEEE Communications Magazine, vol. 51, no. 7, pp. 36-43, July 2013.

[35] M. Dabbagh, B. Hamdaoui, M. Guizani and A. Rayes, "Software-defined networking security: pros and cons," in IEEE Communications Magazine, vol. 53, no. 6, pp. 73-79, June 2015.

[36] Sandra Scott-Hayward, "SDN Security: A Survey," Centre for Secure Information Technology (CSIT) Queen's University Belfast Belfast, BT3 9DT, Northern Ireland, pp. 36-43, 2013.

[37] O. foundation, "Defense4All:Tutorial," [Online]. Available: https://wiki.opendaylight.org/view/Defense4All:Tutorial#Introduction. [Accessed May 2018].

[38] S. Scott-Hayward, "Design and deployment of secure, robust, and resilient SDN Controllers," Centre for Secure Information Technology (CSIT), Queen's University Belfast, Belfast, BT3 9DT, N. Ireland, pp. 2-4, 2015.

[39] C. Liang, R. Kawashima and H. Matsuo, "Scalable and Crash-Tolerant Load Balancing Based on Switch Migration for Multiple Open Flow Controllers," 2014 Second International Symposium on Computing and Networking, Shizuoka, pp. 171-177, 2014.

[40] O. O. Plugin, "OpenFlow plugin," ODL OpenFlow Plugin ,Release master, [Online]. Available: https://media.readthedocs.org/pdf/odl-openflowplugin/latest/odl-openflowplugin.pdf. [Accessed 21 Jan 2020].

[41] K. I. Stephanie, "Network Firewalls," University of New Mexico, Santa Fe Institute, pp. 7-14, 2011.

[42] E. Romanofski, "A Comparison of Packet Filtering Vs Application Level Firewall Technology," no. Global Information Assurance Certification Paper, pp. 2-8, 2002.

[43] Luca Petrucci, Marco Bonola, Salvatore Pontarelli,Giuseppe Bianchi,Roberto Bifulco, "Implementing iptables using a programmable stateful data plane abstraction," Bifulco,CNIT/University of Rome Tor Vergata, pp. 1-2, 2018.

[44] J. Tönsing, "Framework for SDN: Scope and Requirements Open Networking Foundation," pp. 30-32 , June 2015.

[45] Z. Ivy Guo, "Security Foundation Requirements for SDN Controllers, Open Networking Foundation," pp. 5-18, July 2016.

[46] S. Hogg, "SDN Security Attack Vectors and SDN Hardening, Core Networking," 28 October 2014. https://www.networkworld.com/article/2840273/sdn/sdn-security-attack-vectors-and-sdn-hardening.html.

[47] M. Darianian, C. Williamson and I. Haque, "Experimental evaluation of two OpenFlow controllers," 2017 IEEE 25th International Conference on Network Protocols (ICNP), Toronto, ON, 2017, pp. 1-6.

[48] Manijeh Keshtgari , Vajihe Abdi ,Shiva Rowshanrad, "Performance evaluation of SDN controllers: Floodlight and OpenDaylight," Department of Computer and Information Technology, Shiraz University of Technology, Shiraz, Iran, pp. 47-52, 2016.

[49] M. Boye, "Netfilter Connection Tracking and NAT Implementation," Aalto University Otakaari 5 Espoo, Finland, 2018.

[50] Casimer DeCusatis, PhD, Aparicio Carranza, PhD ,jean Delgado-Caceres, "Modeling Software Defined Networks using Mininet," Marist College, Poughkeepsie, NY USA New York City, College of Technology of The City University of New York (CUNY) Brooklyn, USA, pp. 133-1 - 133-5, 2016.

[51] "Introduction to Hyper-V on Windows 10," 25 June 2018. [Online]. Available: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/.

[52] Marco Tiloca, Alexandra Stagkopoulou, Gianluca Dini, "Performance and Security Evaluation of SDN Networks in OMNeT++/INET," KTH Royal Institute of Technology, Isafjordsgatan 22, Kista (Sweden), University of Pisa argo Lazzarino 1, Pisa (Italy), pp. 2-4, 2016.

[53] O. Project, "OpenFlow Plugin Operation," OpenDaylight foundation, [Online]. Available: https://docs.opendaylight.org/projects/openflowplugin/en/latest/users/operation.html. [Accessed June 2019].

8 Appendix A: Experiment

A1. Set up & Configuration

This section describes and gives more details on the specifications and customisation settings of the tools that were utilised in the experiments in the dissertation. The details will provide the installation steps, the method of setup and configurations that were done to deliver the proposed solution as guidance to the reader.

A.2 System Requirements for Software applications

Using the Hyper-V virtualisation tool, the 4 VMs that host the experimental models are created with below specifications in table 7.1 as recommended for the applications they will be running. The ODL host requires the most resources when compared to all other VMs due to the Java based karaf application.

Table 8-1 Hardware specifications used in the experiments

	Random Access Memory (RAM)	CPU and HDD	Operating Software
OpenDaylight VM1	6 GB	4 cores, 60 GB	Ubuntu 16.04
Mininet VM2	2 GB	1 core ,40 GB	Ubuntu 16.04
OpenNMS VM3	2 GB	1 core ,40 GB	Ubuntu 18.04
DDoS Source VM4	2 GB	1 core ,40 GB	Ubuntu 18.04

A.3 SDN controller Setup using OpenDaylight

OpenDaylight controller version that was installed for the PoC was Beryllium 0.4.0. As a prerequisite the Java was installed on the host VM1 to support the ODL karaf application. The major steps for ODL installation in sequential are shown below.

Java software packages installation:

```
#sudo apt-get -y install openjdk-8-jre
#export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
```

OpenDaylight installation:

The ODL software is downloaded from the OpenDaylight website for the specific version of Beryllium. The Figure 7.1 below show the process in downloading the software from the OpenDaylight website (URL is *https://docs.opendaylight.org/en/latest/downloads.html).* Once the download was completed to the VM1 desktop the following commands are used to activate the controller.

```
#sudo unzip distribution-karaf-0.4.0-Beryllium.zip
#sudo distribution-karaf-0.4.0-Beryllium
#sudo. /bin/karaf
```

After launching the application using *sudo. /bin/karaf,* the 3 key features required to run a SDN network need to be installed as shown below. The features that enable the ODL to communicate with the Mininet software and allow GUI access are shown below.

```
opendaylight-user@root>feature:install odl-restconf
opendaylight-user@root>feature:install odl-l2switch-switch
opendaylight-user@root>feature:install odl-dlux-core
```

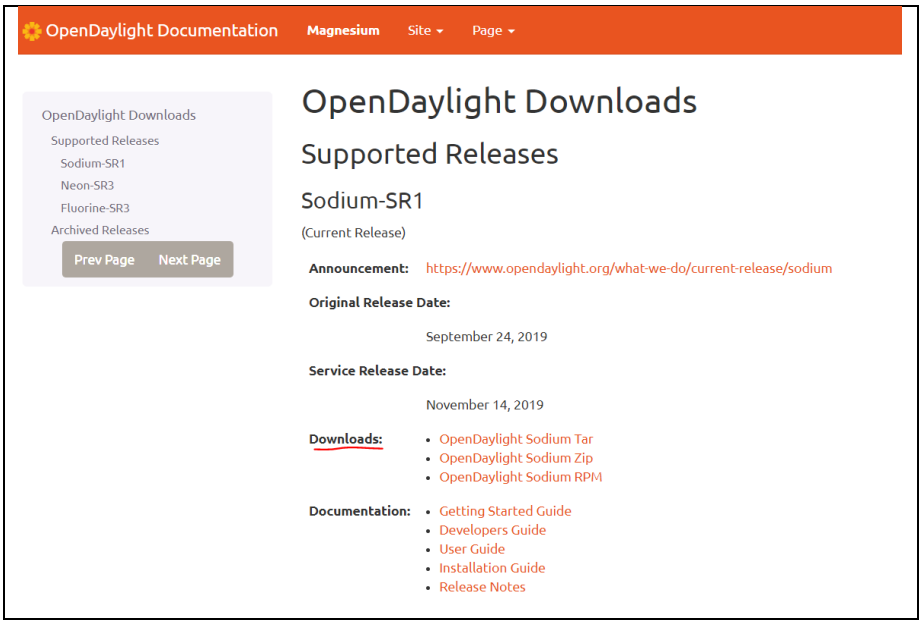


Figure 8.1 *Installation path for ODL software*

Installation of OpenDaylight dlux feature enabled the GUI for accessing the controller remotely within the 10.254.0.1/29. For the experiment the ODL URL is accessible on *http://10.254.0.1:8181/*. The resulting network that was created is shown in Figure 7.2 below.

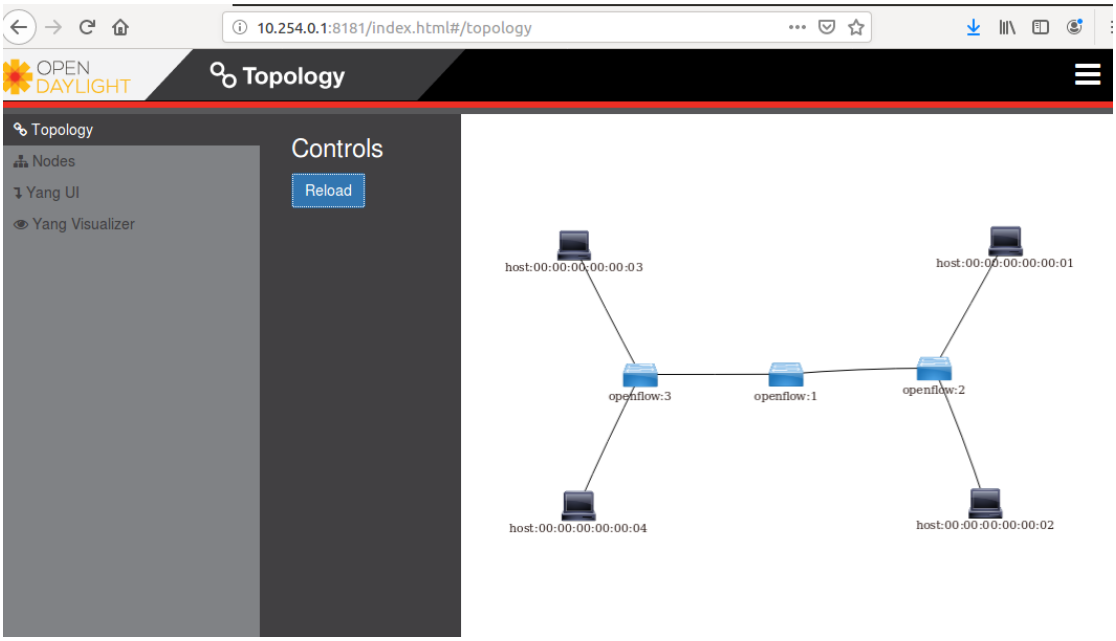


Figure 8.2 *ODL GUI topology view*

A.4 Mininet

Mininet the packet forwarder emulation software is installed on VM2. The installation steps for the software is as shown below from Mininet website. Source of the packages is extracted from ***http://mininet.org/download/*.** The installation option that was used is the native installation from source option.

```
#git clone git://github.com/mininet/mininet
#git checkout -b 2.2.1 2.2.1
#cd ..
#cd mininet
#sudo mininet/util/install.sh
```

A.5 Iptables Firewall

Iptables firewall is installed on Ubuntu operating systems by default and requires definitions of rules for activation of the firewall. Proposed rules in the database are as per design in Figure 5.17 and below in Figure 7.3 are the commands for activating the rules.

```
carlton@ODLNew:~$ sudo iptables -A INPUT -i eth0 -p tcp --dport 6633 -s 10.254.0.2 -d 10.254.0.1 -j ACCEPT
carlton@ODLNew:~$ sudo iptables -A INPUT -i eth0 -p tcp --dport 6653 -s 10.254.0.2 -d 10.254.0.1 -j ACCEPT
carlton@ODLNew:~$ sudo iptables -A INPUT -i eth0 -p tcp --dport 8181 -s 10.254.0.2 -d 10.254.0.1 -j ACCEPT
carlton@ODLNew:~$ sudo iptables -A INPUT -i eth0 -p tcp -s 10.254.0.3 -d 10.254.0.1 -j ACCEPT
carlton@ODLNew:~$ sudo iptables -A INPUT -i eth0 -p udp -s 10.254.0.3 -d 10.254.0.1 -j ACCEPT
carlton@ODLNew:~$ sudo iptables -A INPUT -i eth0 -p tcp --dport 80 -s 10.254.0.4 -d 10.254.0.1 -j DROP
carlton@ODLNew:~$ sudo iptables -A INPUT -i eth0 -p tcp --dport 80 -s 10.254.0.10 -d 10.254.0.1 -j DROP
carlton@ODLNew:~$
carlton@ODLNew:~$ sudo iptables -A OUTPUT -p tcp -s 10.254.0.1 -d 10.254.0.2 -j ACCEPT
carlton@ODLNew:~$ sudo iptables -A OUTPUT -p tcp -s 10.254.0.1 -d 10.254.0.3 -j ACCEPT
carlton@ODLNew:~$ sudo iptables -A OUTPUT -p udp -s 10.254.0.1 -d 10.254.0.3 -j ACCEPT
carlton@ODLNew:~$ sudo iptables -A OUTPUT -p tcp --dport 80 -s 10.254.0.1 -d 10.254.0.4 -j DROP
carlton@ODLNew:~$ sudo iptables -A OUTPUT -p tcp --dport 80 -s 10.254.0.1 -d 10.254.0.10 -j DROP
carlton@ODLNew:~$ sudo iptables -A OUTPUT -s 10.254.0.1 -d 10.254.0.10 -j DROP
```

Figure 8.3 *Commands for activating iptables rules*

Once the rules are executed in the VM1 CLI the rules require to be activated. The activation is done by the command *sudo iptables-save* shown below in Figure 7.4. This will trigger the traffic inspection and enforcing for both the INPUT and OUTPUT chains of the firewall.

```
carlton@ODLNew:~$ sudo iptables-save
[sudo] password for carlton:
# Generated by iptables-save v1.6.1 on Mon Jan 20 17:43:52 2020
*filter
:INPUT ACCEPT [552:39736]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [552:39736]
-A INPUT -s 10.254.0.2/32 -d 10.254.0.1/32 -i eth0 -p tcp -m tcp --dport 6633 -j ACCEPT
-A INPUT -s 10.254.0.2/32 -d 10.254.0.1/32 -i eth0 -p tcp -m tcp --dport 6653 -j ACCEPT
-A INPUT -s 10.254.0.2/32 -d 10.254.0.1/32 -i eth0 -p tcp -m tcp --dport 8181 -j ACCEPT
-A INPUT -s 10.254.0.3/32 -d 10.254.0.1/32 -i eth0 -p tcp -j ACCEPT
-A INPUT -s 10.254.0.3/32 -d 10.254.0.1/32 -i eth0 -p udp -j ACCEPT
-A INPUT -s 10.254.0.4/32 -d 10.254.0.1/32 -i eth0 -p tcp -m tcp --dport 80 -j DROP
-A INPUT -s 10.254.0.10/32 -d 10.254.0.1/32 -i eth0 -p tcp -m tcp --dport 80 -j DROP
-A OUTPUT -s 10.254.0.1/32 -d 10.254.0.2/32 -p tcp -j ACCEPT
-A OUTPUT -s 10.254.0.1/32 -d 10.254.0.3/32 -p tcp -j ACCEPT
-A OUTPUT -s 10.254.0.1/32 -d 10.254.0.3/32 -p udp -j ACCEPT
-A OUTPUT -s 10.254.0.1/32 -d 10.254.0.4/32 -p tcp -m tcp --dport 80 -j DROP
-A OUTPUT -s 10.254.0.1/32 -d 10.254.0.10/32 -p tcp -m tcp --dport 80 -j DROP
-A OUTPUT -s 10.254.0.1/32 -d 10.254.0.10/32 -j DROP
COMMIT
# Completed on Mon Jan 20 17:43:52 2020
carlton@ODLNew:~$
```

Figure 8.4 *Iptables rules activation commands*

A.6 OpenNMS

The OpenNMS software was used for setting up the network monitoring system for capturing the performance KPIs. The Opensource software installation steps was extracted from https://linoxide.com/monitoring-2/install-configure-opennms-ubuntu/. Below are the key steps in the installation procedure in setting up the SNMP server.

```
# sudo wget https://github.com/opennms-forge/opennms-install/archive/master.zip
# sudo unzip master.zip
# sudo bash bootstrap-debian.sh
# sudo /usr/share/opennms/bin/install
# sudo /usr/share/opennms/bin/install -dis
```

The NMS portal that was activated after the successful installation was accessible on http://10.254.0.3:8980/ as shown in Figure 7.5 for statistics extraction. With the server online, it will listen to SNMP port 161 for any agent communication using the default community strings.

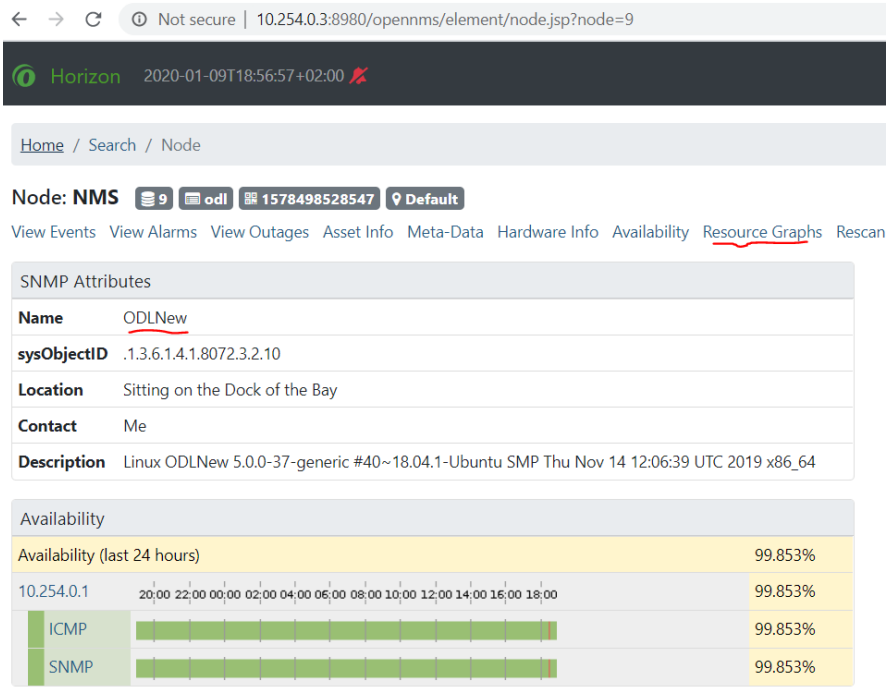


Figure 8.5 *OpenNMS portal available on http://10.254.0.3:8980/*

Once the OpenNMS server is installed we need to install the SNMP client on the host machines to be monitored. The key parameters to be set is the community string, agent address and access-lists permitting polling of the agent by the server. Below are captures of the set parameters for VM1 in Figure 7.6 and 7.7 which enabled the collection of statistics for the VM1 host.

```
carlton@ODLNew:~$ sudo cat /etc/snmp/snmpd.conf
###############################################################################
#
# EXAMPLE.conf:
#   An example configuration file for configuring the Net-SNMP agent ('snmpd')
#   See the 'snmpd.conf(5)' man page for details
#
#  Some entries are deliberately commented out, and will need to be explicitly activated
#
###############################################################################
#
#  AGENT BEHAVIOUR
#

#  Listen for connections from the local system only
#agentAddress  udp:127.0.0.1:161
#  Listen for connections on all interfaces (both IPv4 *and* IPv6)
agentAddress udp:161,udp6:[::1]:161
```

Figure 8.6 *SNMP agent configuration in VM1*

The agent configuration above in Figure 7.4 allows SNMP requests on udp port 161 for polling and statistics collection by the server. Additionally, the access-lists are defined as in Figure 7.5 allowing network 10.254.0.0/24 to access VM1 using SNMP community string *odl* which is shared between the SNMP server and client.

```
#
#  ACCESS CONTROL
#

                                          #  system + hrSystem groups only
view   systemonly  included   .1.3.6.1.2.1.1
view   systemonly  included   .1.3.6.1.2.1.25.1

                                          #  Full access from the local host
 rocommunity odl  10.254.0.0/24
                                          #  Default access to basic system info
 rocommunity odl  default    -V All
                                          #  rocommunity6 is for IPv6
 rocommunity6 odl  default   -V All


                                          #  Full access from an example network
                                          #     Adjust this network address to match your local
                                          #     settings, change the community string,
                                          #     and check the 'agentAddress' setting above
#rocommunity secret  10.0.0.0/16

                                          #  Full read-only access for SNMPv3
 rouser   authOnlyUser
                                          #  Full write access for encrypted requests
                                          #     Remember to activate the 'createUser' lines above
#rwuser   authPrivUser   priv
```

Figure 8.7 *Access control-list for SNMP server authorisation*

A.7 Hping3

It is the tool used to launch the DDoS attack installed on VM4. The tool was downloaded from Linux online packages and setup by executing the commands shown below.

```
sudo apt-get update
sudo apt-get install hping3
```

The hping3 tool is launched to VM1 by executing the command shown in Figure 5.1 in chapter 5.

A.8 Hyper-V

To separate the logical functions of the modules used in the experiments the Microsoft Hyper-V virtualisation tool was used. The tool enabled the creation of the OpenDaylight controller , Mininet VM , OpenNMS server and DDoS attack VM4. The 4 VMs are configured in a private IP network shared in Figure 4.3 and interconnected by use of a virtual switch created in Hyper-V shown in Figure 7.8 below.

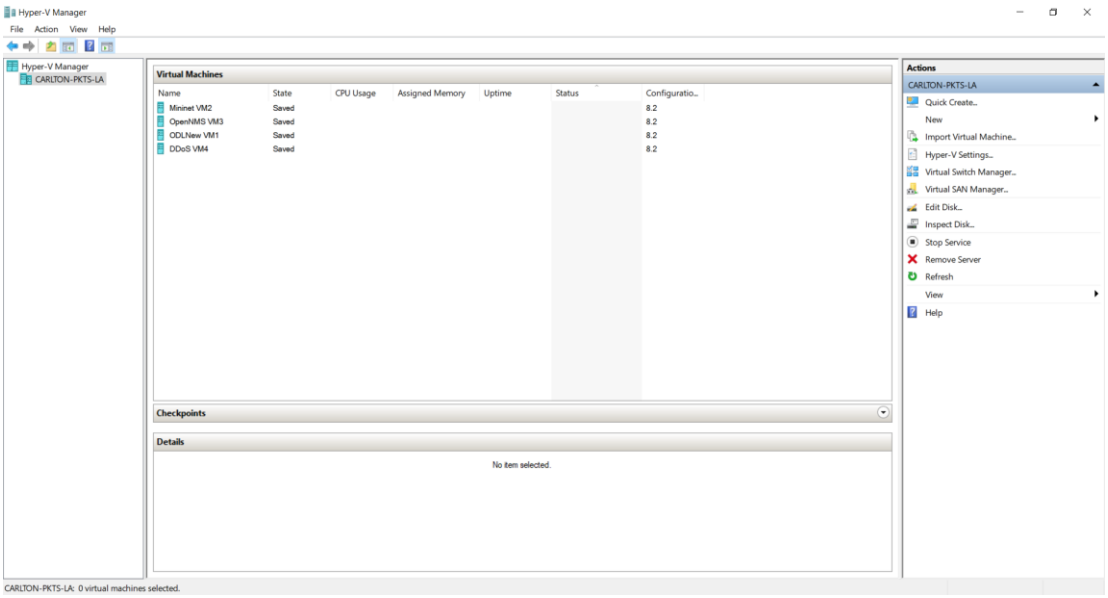


Figure 8.8 *Hyper-V host configuration portal*

The virtual switch enables the communication on the VMs in one logical network. The virtual switch that was created is named *virtual switch 1*. The main laptop host network interface card is bridged to enable sharing of the media by all the VMs. The virtual switch created is shown below in Figure 7.9.

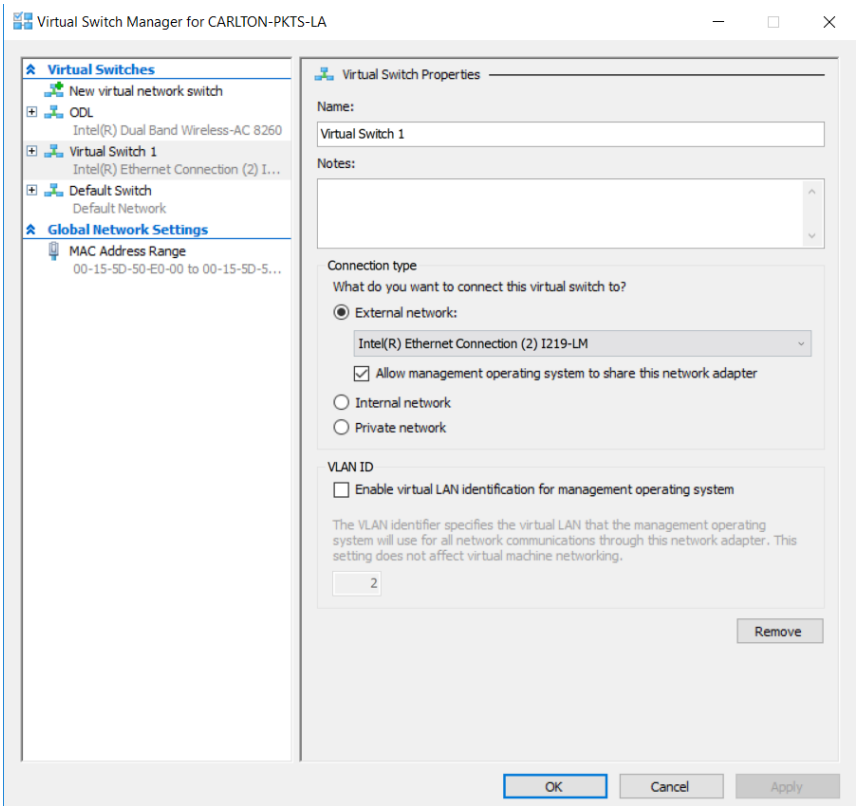


Figure 8.9 *Hyper-V virtual switch settings*

A.9 IP connectivity Tests between VMs

This section displays the IP connectivity results tested between VM1 and the other VMs it communicates using different protocols. The tests were done by use of the ping ICMP tool executed from VM1 and results are displayed in Figure 7.10 ,7.11 and 7.12. All ICMP tests between the VMs were successful with averages latencies of less than 1ms which is accepted for directly connected LAN devices in this environment.

```
carlton@ODLNew:~$ ping 10.254.0.2
PING 10.254.0.2 (10.254.0.2) 56(84) bytes of data.
64 bytes from 10.254.0.2: icmp_seq=1 ttl=64 time=0.781 ms
64 bytes from 10.254.0.2: icmp_seq=2 ttl=64 time=0.654 ms
64 bytes from 10.254.0.2: icmp_seq=3 ttl=64 time=1.39 ms
64 bytes from 10.254.0.2: icmp_seq=4 ttl=64 time=0.475 ms
64 bytes from 10.254.0.2: icmp_seq=5 ttl=64 time=0.257 ms
64 bytes from 10.254.0.2: icmp_seq=6 ttl=64 time=0.613 ms
64 bytes from 10.254.0.2: icmp_seq=7 ttl=64 time=0.787 ms
64 bytes from 10.254.0.2: icmp_seq=8 ttl=64 time=0.227 ms
64 bytes from 10.254.0.2: icmp_seq=9 ttl=64 time=0.294 ms
^C
--- 10.254.0.2 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8108ms
rtt min/avg/max/mdev = 0.227/0.609/1.395/0.344 ms
```

Figure 8.10 *Test between VM1 and VM2*

```
carlton@ODLNew:~$ ping 10.254.0.3
PING 10.254.0.3 (10.254.0.3) 56(84) bytes of data.
64 bytes from 10.254.0.3: icmp_seq=1 ttl=64 time=1.12 ms
64 bytes from 10.254.0.3: icmp_seq=2 ttl=64 time=0.888 ms
64 bytes from 10.254.0.3: icmp_seq=3 ttl=64 time=0.910 ms
64 bytes from 10.254.0.3: icmp_seq=4 ttl=64 time=0.297 ms
64 bytes from 10.254.0.3: icmp_seq=5 ttl=64 time=0.939 ms
64 bytes from 10.254.0.3: icmp_seq=6 ttl=64 time=0.856 ms
64 bytes from 10.254.0.3: icmp_seq=7 ttl=64 time=0.873 ms
64 bytes from 10.254.0.3: icmp_seq=8 ttl=64 time=0.825 ms
64 bytes from 10.254.0.3: icmp_seq=9 ttl=64 time=1.24 ms
64 bytes from 10.254.0.3: icmp_seq=10 ttl=64 time=1.15 ms
64 bytes from 10.254.0.3: icmp_seq=11 ttl=64 time=0.994 ms
^C
--- 10.254.0.3 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10099ms
rtt min/avg/max/mdev = 0.297/0.918/1.249/0.239 ms
```

Figure 8.11 *Test between VM1 and VM3*

```
carlton@ODLNew:~$ ping 10.254.0.4
PING 10.254.0.4 (10.254.0.4) 56(84) bytes of data.
64 bytes from 10.254.0.4: icmp_seq=1 ttl=64 time=2.73 ms
64 bytes from 10.254.0.4: icmp_seq=2 ttl=64 time=0.654 ms
64 bytes from 10.254.0.4: icmp_seq=3 ttl=64 time=0.839 ms
64 bytes from 10.254.0.4: icmp_seq=4 ttl=64 time=1.46 ms
64 bytes from 10.254.0.4: icmp_seq=5 ttl=64 time=0.882 ms
64 bytes from 10.254.0.4: icmp_seq=6 ttl=64 time=1.41 ms
64 bytes from 10.254.0.4: icmp_seq=7 ttl=64 time=1.07 ms
64 bytes from 10.254.0.4: icmp_seq=8 ttl=64 time=0.836 ms
64 bytes from 10.254.0.4: icmp_seq=9 ttl=64 time=0.834 ms
^C
--- 10.254.0.4 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8051ms
rtt min/avg/max/mdev = 0.654/1.192/2.737/0.606 ms
```

Figure 8.12 *Test between VM1 and VM4*