

**SAKARYA UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY**

**A TESTBED DESIGN FOR INTRUSION DETECTION
AND MITIGATION IN SDN ARCHITECTURE BY
USING DPI**

M.Sc. THESIS

Ahmed DIRIE

Department : COMPUTER AND INFO. ENGINEERING

Field of Science : COMPUTER AND INFO. ENGINEERING

Supervisor : Prof. Dr. Celal ÇEKEN

November 2017

SAKARYA UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY

**A TESTBED DESIGN FOR INTRUSION DETECTION
AND MITIGATION IN SDN ARCHITECTURE BY
USING DPI**

M.Sc. THESIS

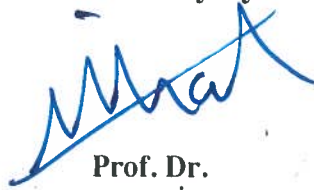
Ahmed DIRIE

Department : COMPUTER AND INFO. ENGINEERING
Field of Science : COMPUTER AND INFO. ENGINEERING
Supervisor : Prof. Dr. Celal ÇEKEN

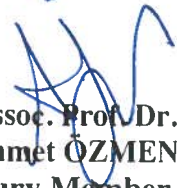
**This thesis has been accepted unanimously by the examination committee on
08.11.2017**



**Prof. Dr.
Celal ÇEKEN
Head of Jury**



**Prof. Dr.
M. Melih İNAL
Jury Member**



**Assoc. Prof. Dr.
Ahmet ÖZMEN
Jury Member**

DECLARATION

I declare that all the data in this thesis was obtained by myself in academic rules, all visual and written information and results were presented in accordance with academic and ethical rules, there is no distortion in the presented data, in case of utilizing other people's works they were refereed properly to scientific norms, the data presented in this thesis has not been used in any other thesis in this university or in any other university.

Ahmed DIRIE

11.11.2017

ACKNOWLEDGEMENT

Firstly, I would like to precise my thankfulness to “ALLAH” for giving me ability to complete my thesis successfully. Without his wishes, I could not reach this milestone.

With deep appreciation and respect, I hope to give my exclusive credits to my research and thesis supervisor Prof. Dr. Celal ÇEKEN for his valuable efforts, great support and advice that he gives me since the first day I joined Department of Computer and Information Engineering in different ways and for giving me the opportunity to be a member of his research team. His productive and positive comments have been inspiring and created a pleasant working atmosphere. He really inspired me to become an outstanding independent researcher and helped me to build critical thinking and reasoning toward solving the difficult research problems. Indeed, his aspirations and determination toward achieving big goals influenced me to complete this thesis successfully.

Furthermore, I would like to express my special thanks to all my lecturers in the Department of Computer and Information Engineering at Sakarya University for their remarkable contribution to my educational career. I also owe a great debt to Soumaine BOUBA for proofreading my thesis and paper. Last but not least, I want to thank my colleges and collaborators in the Internet of Things IoT-LAB team for providing an environment that was enjoyable to be a part of.

Finally, I deeply thank my beautiful family especially my lovely mother and father for their unconditional love, valuable advice, encouragement, trust, prayers, and endless patience. Without them I could not complete this thesis successfully.

TABLE OF CONTENTS

DECLARATION	i
ACKNOWLEDGEMENT	i
TABLE OF CONTENTS	ii
LIST OF SYMBOLS AND ABBREVIATIONS	v
LIST OF FIGURES	vi
SUMMARY	vii
ÖZET	viii
CHAPTER 1.	
INTRODUCTION	1
1.1. Problem Statement and Motivation	3
1.2. Structure of The Thesis	4
CHAPTER 2.	
SOFTWARE DEFINED NETWORKING	5
2.1. Software Defined Networking Definition	7
2.2. Architecture and Concept of SDN	8
2.3. Northbound Interface	9
2.4. Southbound Interface	10
2.5. OpenFlow Protocol	10
2.5.1. Switch components	11
2.5.1.1. Flow table	12
2.5.1.2. Secure channel	12
2.6. SDN Controller	13
2.6.1. Floodlight SDN controller	14

CHAPTER 3.	
ANOMALY DETECTION AND DOS ATTACKS	16
3.1. Methodological Overview	17
3.2. Using SDN For Anomaly Detection in User Traffic	17
3.3. Deep Packet Inspection	18
3.4. DoS Attack and Defense Methods	18
3.5. DoS and DDoS Attacks	19
3.5.1. Attack classification	20
3.5.1.1. Protocol attacks	20
3.5.1.2. Bandwidth attacks	20
3.5.1.3. Logic attacks	21
3.5.2. Defense classification	21
3.5.2.1. Attack prevention	22
3.5.2.2. Attack detection	22
3.5.2.3. Attack source identificaton	23
3.5.2.4. Attack reaction	24
CHAPTER 4.	
EXPERIMENTAL IMPLEMENTATION.....	25
4.1. Tools Used for The Testbed	27
4.1.1. Mininet	27
4.1.2. sFlow	28
4.2. Environment	28
4.2.1. Topology	29
4.2.2. Attack scenario	36
4.3. Results	38
CHAPTER 5.	
CONCLUSION AND FUTURE WORK	39

REFERENCES	41
RESUME	45



LIST OF SYMBOLS AND ABBREVIATIONS

API	: Application Programming Interface
CPU	: Central Processing Unit
DLP	: Data Loss Prevention
DDoS	: Distributed Denial of Service
DoS	: Denial of Service
DPI	: Deep Packet Inspection
FTP	: File Transfer Protocol
ICMP	: Internet Control Message Protocol
IoT	: Internet of Things
IPS	: Intrusion Prevention System
IP	: Internet Protocol
JSON	: JavaScript Object Notation
MAC	: Media Access Control
ONF	: Open Networking Foundation
UDP	: User Datagram Protocol
PCAP	: Packet Capture
RAM	: Random Access Memory
SDN	: Software Defined Networking
SPI	: Stochastic Packet Inspection
SYN	: Synchronize
TCP	: Transmission Control Protocol
TLS	: Transport Layer Security
VLAN	: Virtual Local Area Network

LIST OF FIGURES

Figure 1.1. SDN topology used in this thesis	3
Figure 2.1. Traditional IP network device	6
Figure 2.2. SDN infrascture network device	7
Figure 2.3. SDN architecture in detail	9
Figure 2.4. OpenFlow algorithm	11
Figure 2.5. OpenFlow switch component	12
Figure 2.6. Floodlight SDN controller	15
Figure 3.1. DDoS attack structure	19
Figure 4.1. Anomaly detection system login screen	25
Figure 4.2. Firewall module settings	26
Figure 4.3. Deep packet inspection module	27
Figure 4.4. Launching floodlight SDN controller	30
Figure 4.5. Floodlight built-in topology	31
Figure 4.6. Starting sflow monitor services	31
Figure 4.7. Topology created by python script	34
Figure 4.8. Launching node.js server	35
Figure 4.9. Mapping between mininet and sflow	36
Figure 4.10. Sequence diagram	37
Figure 4.11. Full packet capture in wireshark	37
Figure 4.12. Anomaly detection in SDN	38
Figure 4.13. Mitigation of dos attack	38

SUMMARY

Keywords: Software Defined Networking, Anomaly Detection, Deep Packet Inspection, Testbed

Over the last few decades, computer technologies which are used to design and build networks have remained unchanged. In the meantime, the number of connected networking devices has raised exponentially, thereby increasing the size of computer networks. Accordingly, the existing networks in data centres and companies have become much more difficult and harder to administrate.

Software Defined Networking's (SDN) idea brings the fact of separating the control plane from the data plane which were previously tighten together in the same device, and thus allows the network to be programmed from a logically centralized place called the SDN controller. The data plane in this structure consists of dump devices which are only capable of forwarding the data as instructed by the SDN controller. OpenFlow is the well-known protocol used to take the communication between the SDN controller and the forwarding devices.

In this study, a new testbed has been implemented for anomaly detection in SDN. The testbed formed has several components such as a web based application, an anomaly detection sub-system, an SDN structure with floodlight controller and sFlow protocol. The system developed examines the payload of the packets in order to find any threats in ongoing traffic. In order to investigate the performance of the testbed developed, DoS attack has been considered. The results show that experiments related to security aspects of the SDN systems can be realized by the testbed, easily.

YAZILIM TANIMLI AĐ MİMARİSİNDE DERİN PAKET ANALİZİ KULLANARAK SALDIRI TESPİTİ VE ÖNLEME İÇİN DENEY DÜZENEGİ TASARIMI

ÖZET

Anahtar kelimeler: Yazılım Tanımlı Ađ, Anomali Tespiti, Derin Paket Analizi, Test Düzenegi

Son on yılda, ađları tasarlamak ve geliřtirmek için kullanılan teknolojiler konusunda köklü deđişiklikler yařanmamıřtır. Bu süre zarfında, ađa bađlı cihazlarının sayısı üstel olarak artarak bilgisayar ađlarının toplamı ve boyutunun artmasına yol açtı. Bu ise, veri merkezlerinde ve řirketlerde mevcut ađ yapılarının yönetimini daha da zorlařtırdı.

Yazılım Tanımlı Ađ fikri, daha önce aynı cihazda sıkıřtırılmıř olan veri düzlemi ile denetim düzlemini birbirinden ayırmayı getirir ve böylece tüm ađ yapısının SDN denetleyici adı verilen merkezi bir yerden programlanmasına imkan verir. Bu yapı içerisindeki very düzlemi, kendisine gelen verileri SDN denetleyici tarafından belirlendiđi řekilde bir sonraki düđüme ileten aptal cihazlardan oluşur. OpenFlow, SDN denetleyici ile very düzelmi cihazları arasındaki bađlantıyı sađlamak üzere yaygın olarak kullanılan haberleřme protokolüdür.

Oluřturulan test düzenegi web uygulaması, anormal durum tespiti alt sistemi, floodlight denetleyiciye sahip SDN yapısı ve sFlow protokolü gibi çok sayıda bileřene sahiptir. Geliřtirilen system, akan trafik üzerindeki tehditleri bulabilmek için paketlerin yük kısımlarını incelemektedir. Geliřtirilen test düzeneginin başarımını sorgulamak için DoS saldırısı göz önüne alınımıřtır. Elde edilen sonuçlar SDN sistemlerin güvenliđiyle ilgili deneylerin oluřturulan bu test düzenegi ile kolayca gerçekleştirilebileceđini göstermektedir.

CHAPTER 1. INTRODUCTION

Over the last few decades, computer technologies which are used to design and build networks have remained unchanged. In the meantime, the number of connected networking devices has raised exponentially, thereby increasing the size of computer networks. Accordingly, the existing networks in data centres and companies have become much more difficult and harder to administrate.

Though the quantity of Information Technology services is becoming quickly, organizations are taking them out from self-guided infrastructure. Additionally, the utilisation of cloud computing has turned out to be increasingly well known and the advances in the Internet of Things (IoT) made each single device's connection to the internet a must. Those things are passing on the security to fundamental bit of frameworks organization and especially in cloud and server farm, the security is remaining an important factor.

The following vast development in Software Defined Networking is to move conventional and fixed Ethernet systems to considerably unique and effortlessly sensible ones. Software Defined Networking (SDN), changes the Ethernet layer 2 to centrally directed layer 2 clouds, where the system's activity can be managed utilizing the advantage of programming languages. To state the preferred high-level network policies, network administrators need to configure individual network devices independently using low level and frequently vendor specific commands. Furthermore, to the complexity of the configuration, network environments have to suffer the dynamics of faults and adapt load changes. Automatic (re)configuration and response methods are practically absent in traditional IP networks. Applying the required policies in such a dynamic environment is therefore extremely challenging. that gives

us to fabricate more effective and further secure networks, in Figure 1.1. an example of an SDN topology utilized as a part of this thesis is shown.

In this study, we are investigating and providing a testbed of SDN-based networks by doing anomaly detection for the continuous packets, which will give better system security. We are likewise discovering presented Software Defined Networking applications constructed uniquely for Anomaly Detection and Deep Packet Inspection. The motivation behind this study is to discover current situation with SDN idea and assess its development for more extensive use on research and production environments.

This thesis deals with creating a testbed to do anomaly detection in the new era of networking called Software Defined Networking with an attack scenario to validate the testbed, in order to do definite anomaly detection by using a static threshold value. If the ping attack reaches beyond the threshold value, the created system will automatically drop the packet from the machine originating the attack. Software Defined Networking (SDN) provides an abstraction layer for the physical network and separates what is called the or the so-called control plane which in human analogy is the brain of the body from forwarding plane. Furthermore, SDN hype is growing rapidly in big data centers like Google, Cisco and Facebook. Which the idea of SDN is mainly aimed to reduce the complexity of designing the network configurations and costs. Security is as always, a big concern in the business continuity and the issue of security remains the same with traditional networks. A lot of study must be done and unfortunately there is no such a system that is 100 percentage secure from anything, every device which is connected to the Internet is vulnerable to certain attack must well-known attacks are Denial-of-Service attack.

We will simulate an attack to a virtual server which resides in our SDN-based network which will overwhelm the virtual server and cause unavailability to legitimate requests from clients. This form of an attack is called Denial-of-Services (DoS) attack. There are many techniques to protect networks from attackers, i.e. using middle boxes alike devices Intrusion Prevention System (IPS) and/or Firewall. Although they cannot

mitigate DoS attack easily. This research will enhance the operation of DoS mitigation. We apply another method by using SDN technology such as sFlow and OpenFlow.

The procedure of sFlow is to detect the attacker by taking some accumulative traffic from agent(s) to send to the sFlow collector to analyze it. When sFlow collector discovers some traffics as attacker, it will send to the Floodlight controller then will modify the rule in OpenFlow-enabled switch table to mitigate attacks by preventing attack(s) traffic. Thus, by mixing some cumulative traffic using sFlow and preventing traffic using OpenFlow, we can detect and mitigate ping flood attack rapidly. In addition to mitigating attacks, the testbed platform can also perform full packet capture in SDN. These packet captures can be used later in Deep Packet Inspection (DPI) and Data Loss Prevention (DLP).

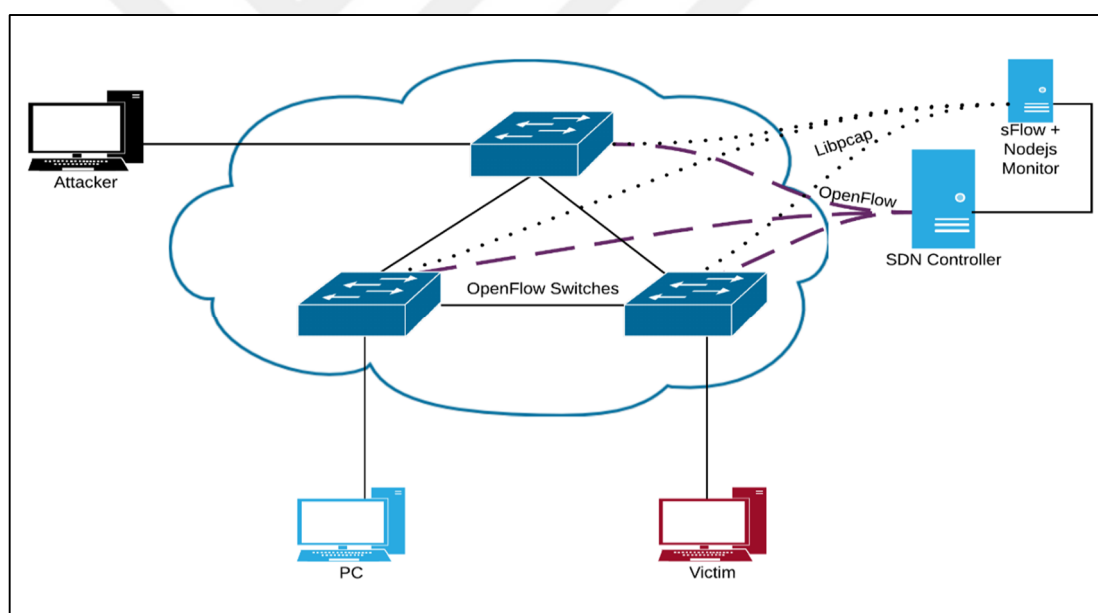


Figure 1.1. SDN topology used in this study

1.1. Problem Statement and Motivation

The architecture idea of Software Defined Networking (SDN) is a novel and a new way of programming the network. In SDN enabled networks, switches and routers do not process the incoming packets. They consider their forwarding tables for a match of incoming packet and if there is nothing matching, it will be sent to the controller for

handling. In SDN, the controller is the operating system. It handles the packets and decides whether the packet will be sent in the switch or will be dropped. By applying this method, SDN separate the data plane from control plane.

One of the potentials possibilities that can cause the emulated hosts and the controller inside the SDN enabled networks to be unreachable is DoS attack. The main goal of this research is to build a testbed and do anomaly detection in SDN enabled networks, and mitigate the DoS attack towards the emulated hosts in our SDN networks.

1.2. Structure of The Thesis

This thesis is structured as follows; in Chapter 2 we will talk in detail about Software Defined Networking, what is it? The definition of SDN, the difference between SDN and traditional networks, and OpenFlow protocol, the most well-known southbound API in SDN architecture. In Chapter 3, we will talk about Anomaly Detection in the context of SDN, what is called Anomaly Detection? Furthermore, how it can be integrated into SDN, and Denial of Service attacks, classification of attacks and defense, and briefly what is Deep Packet Inspection and how to look deeply in the payload of the packets. In the fourth Chapter, we will experiment the testbed and we will discuss the tools that we have used. We will also discuss the web-based system that we created. Finally, in Chapter 5 we will conclude the thesis and we will discuss the future improvements that are possible to enhance this work.

CHAPTER 2. SOFTWARE DEFINED NETWORKING

SDN is a new model in how we create and manage networking devices nowadays, it is the norm in networking and it enables programmable, dynamic and flexible network architectures. Software Defined Networking idea can dramatically reduce operational costs in large data centers and increase the flexibility in design and implementation in the network. Thus the main concept is taking off the control plane from the data plane [1]. In other words, decoupling the data plane and the control plane from each other, in this regard the traditional network devices, let say for example routers or switches, were having the control plane and the data plane in the same device. In Figures 2.1. and 2.2. We will present the differences between the traditional network infrastructure and the SDN based infrastructure. In Figure 2.1. we can see that the control plane (which decides how to forward packets) and the data plane (which is responsible to forward the packet) are tighten together in the same hardware device. On the other hand, in Figure 2.2. we can see that the control plane is decoupled from the data plane into a place where is it logically centralized.

The logical processing of network packets is taken over by a logically centralized instance, called the SDN controller (i.e. Floodlight). Thus, SDN exploits the fact that software is more flexible than hardware in terms of design, implement and improve. In large networks, the configuration of network elements is error-prone, SDN provides the advantage that configurations are centralized in one component. Furthermore, SDN bears the potential for network virtualization.

Several concepts such as Virtual Local Area Networks (VLAN) were developed in the past, but the consequence was an increased complexity, especially in wide network infrastructures. Existing SDN software solutions allow network virtualization based on flexible criteria, such as network packet header fields. This provides a network

abstraction layer which allows multiple tenants to share a single physical network. Additionally, SDN controllers provide APIs (Application Programming Interface) for third-party applications, allowing customizable and manageable networks. However, the centralization of the management plane entails risks: the controller is a critical component, and, if compromised, affects the availability of network services.

Network security has been widely addressed in researches, and several tools and techniques exist for traditional IP network security. Network-based firewalls is a device that monitors incoming and outgoing network traffic and decides whether to allow or block specific traffic based on a defined set of security rules. Intrusion Prevention System (IPS) inspects traffic flowing through a network and is capable of blocking or otherwise remediating flows that it determines are malicious. Usually uses a combination of traffic and file signatures and heuristic analysis of flows. Intrusion Detection System (IDS) similar to IPS but does not affect flows in any way, only logs or alters on mmalicious traffic. But in SDN architecture there is no such devices yet, and widely deployed to mainstream networks. For the case of anomaly detection and deep packet inspection, researchers are still looking for developing such a device.

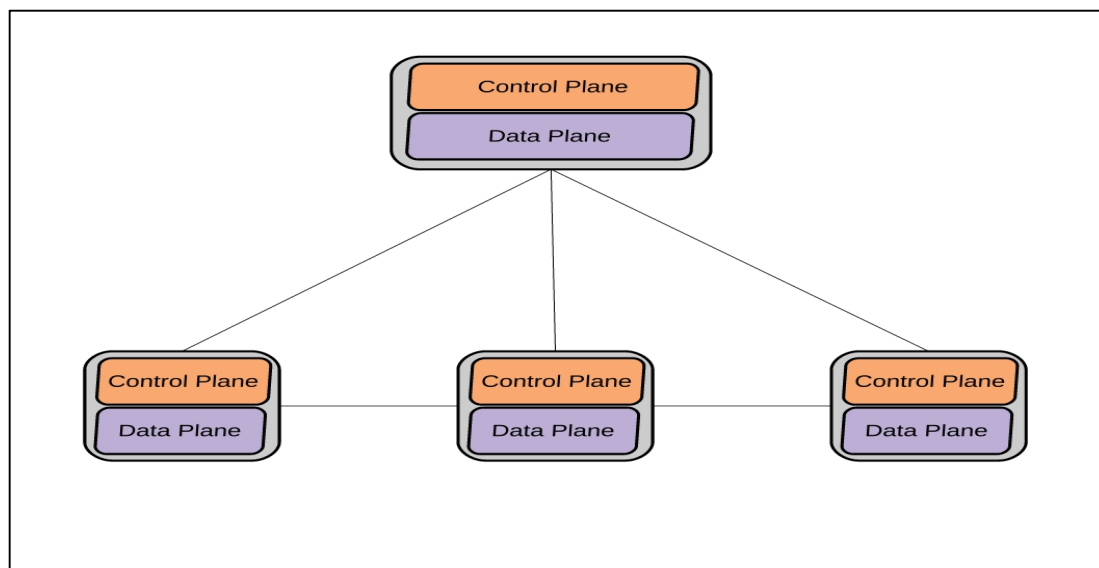


Figure 2.1. Traditional IP networks device

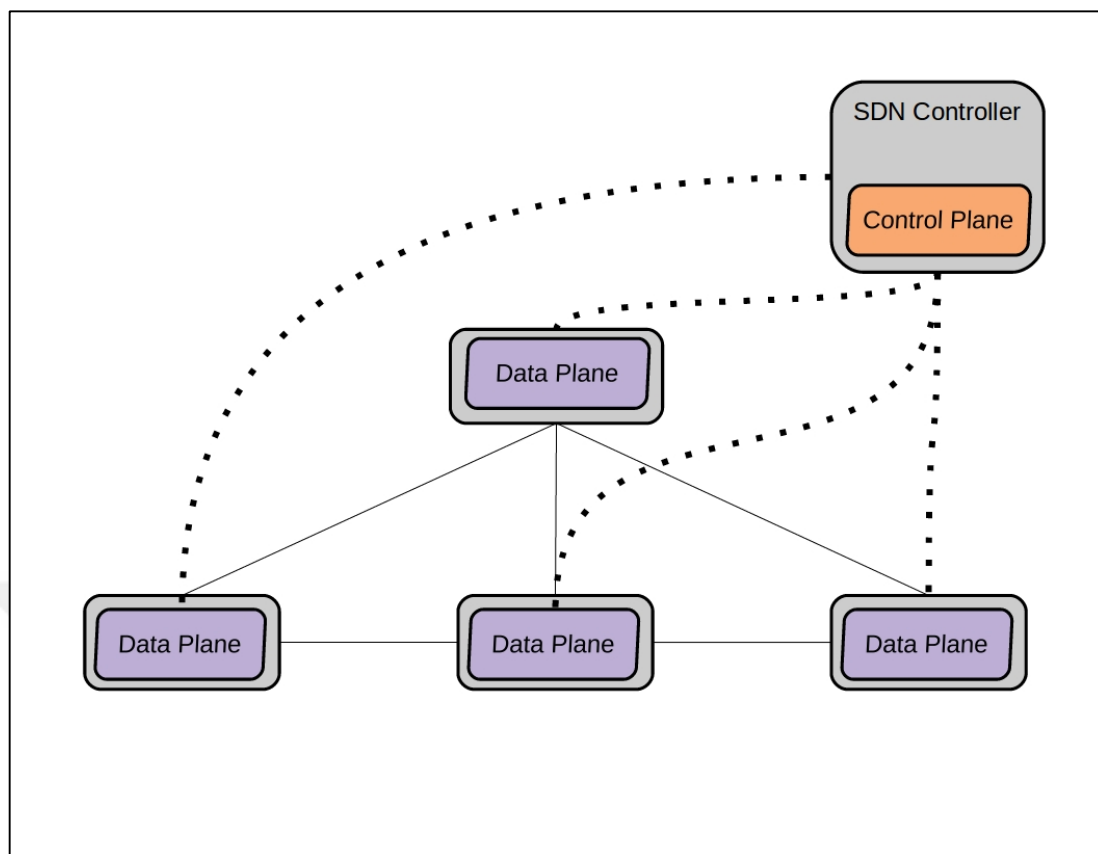


Figure 2.2. SDN infrastructure network device

2.1. Software Defined Networking Definition

In Information Technology and in computer networking engineering, the thought of Software Defined Networking (SDN) or programmable network devices is still growing, through different visions from different designers of its range. A user-driven organization called Open Network Foundation (ONF) is created with the aim of adopting the advancement of SDN, describes it as: “The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices” [2]., which means in the traditional network device, let’s say a Cisco router, this device was doing all the intelligent work, be it routing between paths, or doing network address translation, or even doing some security like access control list to prevent certain devices or networks from accessing each other, all this work was done in the same hardware device. SDN comes to decouple this integration, and allows

certain cheap hardware or software devices to do the data forwarding action, where the intelligence is taken off the device, centralized logically and configured from one spot.

The control plane is directly programmable through coding and programming using Application Programming Interface (API) which conveys to the flexibility of the control plane to exchange information with networking devices. An Application Programming Interface (API) allows to modify network services through the communication to the control plane. This make computer networking more flexible to design and manage at the same time. The idea of old fashioned networking devices was containing data and control planes in the same hardware device and was tightly integrated together.

2.2. Architecture and Concept of SDN

During the last decade, Information Technology was growing very rapidly, and the work needed for administering the computer networks has become more serious and time consuming. As the information networks grow, more computer networking devices as well are required.

The fundamental idea of SDN is to isolate the control plane from the data plane and bring it to one single piece of device, which implies that each system device requires just to deal with the data plane and move information packets starting with one point then onto the next in view of the sending choices made by the SDN controller [1]. In this way, every change is controlled from one controller through Application Programming Interface (API) and the controller is directed with application layer SDN applications. The essential SDN architecture is illustrated in detail in Figure 2.3. and the SDN controller is depicted more precisely in upcoming sections 2.6.

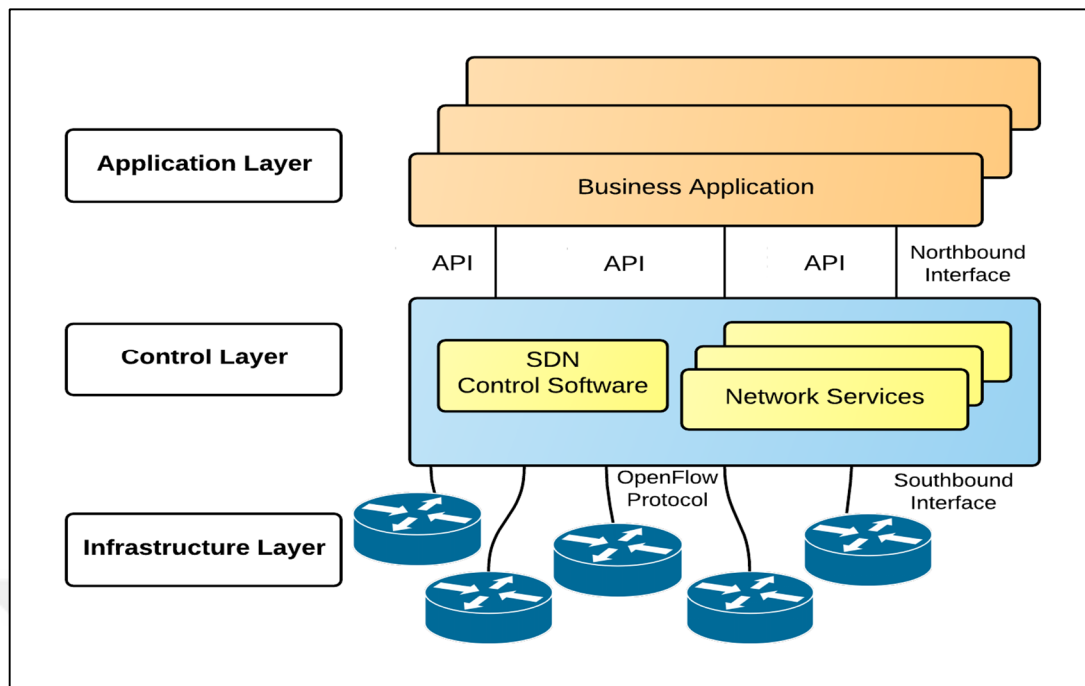


Figure 2.3. SDN architecture in detail [1].

The application layer is the layer on top of the SDN controller where the entire SDN is administered. The administration component might be one SDN application composed for some task of a significantly more complex framework. For instance, OpenStack distributed computing cloud can administer networks by utilizing its own network administration modules. SDN controller constructs flows to SDN-enabled switches and rules how the devices connected to the switch can communicate.

2.3. Northbound Interface

The northbound API is an approach to administer the controller and the entire SDN based networks. By creating an external SDN application we can change network policies and structures. Northbound API is typically applied using restful API (representational state transfer) which causes handling of the controller simple with basic HTTP methods like POST, PUT, GET and DELETE.

External SDN applications are the one managing SDN network by using northbound API, and they are running outside the controller using programming interface for communication. That type of application is capable to administer the controller and

send instructions for it. Northbound APIs are debatably the utmost critical APIs in the SDN network, since the value of SDN is attached to the innovative applications it can possibly support and enable. Because they are so critical, northbound APIs must support a wide variety of applications, so one size will likely not fit all. This is feasibly why SDN northbound APIs are presently the utmost vague element in an SDN network.

2.4. Southbound Interface

Southbound APIs are used to communicate between the switches and routers to the SDN Controller of the network. They can be proprietary or open. Southbound APIs accelerate effective control over the network and enable the SDN Controller to dynamically make changes matching to real-time needs and demands. OpenFlow, which was developed by the Open Networking Foundation (ONF), is the first and perhaps most well-known southbound interface. It is an industry standard that defines the way the SDN Controller should interact with the forwarding plane to modify the network, so it can better adapt to changing business requirements. With OpenFlow, entries can be added and removed to the internal flow-table of switches and potentially routers to make the network more responsive to real-time traffic demands. Beside OpenFlow, Cisco OpFlex (the company's response to OpenFlow) is also a well-known southbound API.

2.5. OpenFlow Protocol

Traditional network has been questioned after the explosion of server virtualization, cloud computing and mobile devices, security issues and the advent of cloud services are among the reasons for the computer networks industry to do dramatical changes. OpenFlow protocol is planned to solve the problem of allocating resources to users in an easy way by giving them the control of the network without interrupting traffic flows [3].

In the old switches and routers, both the data plane and the control plane are in the same device. An OpenFlow Switch splits these functions into two. The function of the data plane still exists on the switch, while the function of the control plane is pushed to a separate device called the SDN Controller that handles the communication between switches and the controllers through Secure Channel, via the OpenFlow Protocol.

The switch contains one or more flow tables, OpenFlow protocol is responsible for adding, updating and deleting switch flow entries. When the packet flow comes to the switch, the OpenFlow enabled switch will check it. If the packets match the flow table; the action described at the flow entry is performed. If not, the packet is either dropped or sent to the SDN Controller. In the following figure we briefly demonstrate an OpenFlow algorithm, how it handles for the coming new packets that entering the OpenFlow enabled switch.

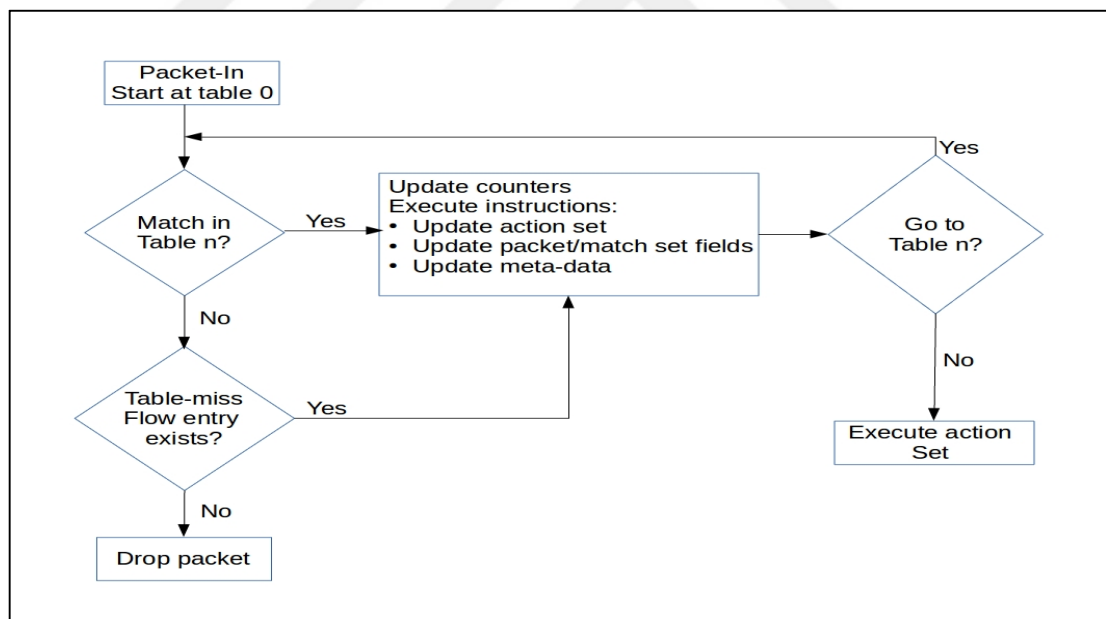


Figure 2.4. OpenFlow algorithm

2.5.1. Switch components

An OpenFlow enabled switch as shown in the Figure 2.5. contains one or more flow tables, which does the following; packet forwarding, packet lookup, and a secure

channel to an external SDN controller and then the SDN controller points the OpenFlow enabled switch over the secure channel using the OpenFlow protocol.

2.5.1.1. Flow table

This section describes the components of flow table entries. A flow entry consists of header fields, counters, and actions.

Each flow table entry holds the following:

- a. Header fields are for matching against packets
- b. Counters for updating the packets
- c. Actions will apply for matching packets

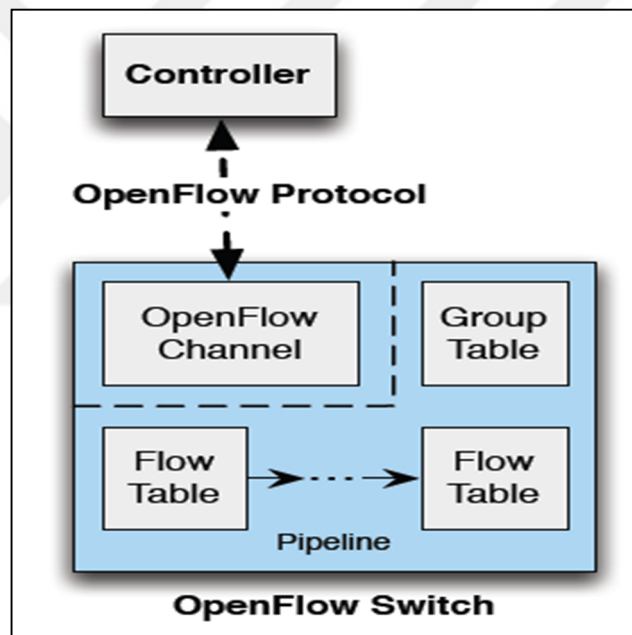


Figure 2.5. OpenFlow switch component [3].

2.5.1.2. Secure channel

An OpenFlow switch and the controller communicate through a secure channel and is instantiated as a single network connection between the controller and the switch, using Transport Layer Security (TLS). One controller can manage multiple secure channels within the topology.

2.6. SDN Controller

In Software Defined Networking, the control plane is a special network component, called the controller. Since it is purely software-based, the developer community is quite large compared to OpenFlow-switches. However, most projects are driven by the same companies (Big Switch, IBM, HP, Cisco, etc.). In addition to commercial solutions, many open source controllers are available. They vary mostly in the way the northbound API is implemented. While the southbound API is standardized, the interfaces on top of the controllers are very different. The Open Daylight project was launched by several ONF members to deal with these disparities.

Big Network Controller and Floodlight are two controllers developed by Big Switch. While the Big Network Controller is a commercial solution [4]. Floodlight was published under an open source license. Since the commercial software is based on Floodlight, modules are compatible with both controllers [5]. The Architecture of Floodlight SDN controller is shown in Figure 2.6. Developers can implement their own Floodlight modules to handle specific OpenFlow traffic. This can be done by extending abstract classes and implementing interfaces that are available in the Floodlight core. Floodlight itself provides core modules that offer basic controller functionalities like layer 2 forwarding and topology discovery.

The Open Daylight Project was started to provide a consistent northbound interface. This was necessary, since controller projects have diverged in the past years. The controller software is open source, making it easy for developers to improve the core components and implement their own applications. The Open Daylight controller provides a northbound API that can be used by applications [6].

Other controller implementations have been developed since the emergence of OpenFlow.

1. NOX was the first OpenFlow controller. It provides an interface for additional modules and is written in C++ and Python. The core components include topology

discovery, layer 2 and layer 3 switching. The first version was published in 2007. Since 2008, NOX is open source. Newer versions of NOX are implemented exclusively in C++. A pure python controller was released under the name POX [7].

2. Beacon is an open source controller, implemented in Java. It was released in 2010 and was widely used in research and as a basis for Floodlight [8]. Beacon uses the OpenFlow J library, which is a Java implementation of the OpenFlow 1.0 specification.

2.6.1. Floodlight SDN controller

Floodlight SDN Controller [9]. is written in Java programming language, it is an open source controller maintained by a team of engineers and programmers from Big Switch Networks. Floodlight is invented to work with the growing number of different types of routers, switches, virtual switches, and access points that support the OpenFlow standard [3].

Floodlight OpenFlow Controller has different features to solve the user requirements, and it is built on top of the Floodlight controller. The Figure 2.6. shows the architecture of Floodlight and the relationship between applications built on top of it. The applications are built using Java modules and compiled with Floodlight, and there are applications built over the Floodlight REST API.

Features of Floodlight:

- a. Offered easy to extend and improvements.
- b. Support a wide range of physical and virtual OpenFlow enabled switches.
- c. Possibility to mix OpenFlow and non-OpenFlow networks and it can direct Multiple of OpenFlow enabled switches.
- d. OpenStack cloud is supported.

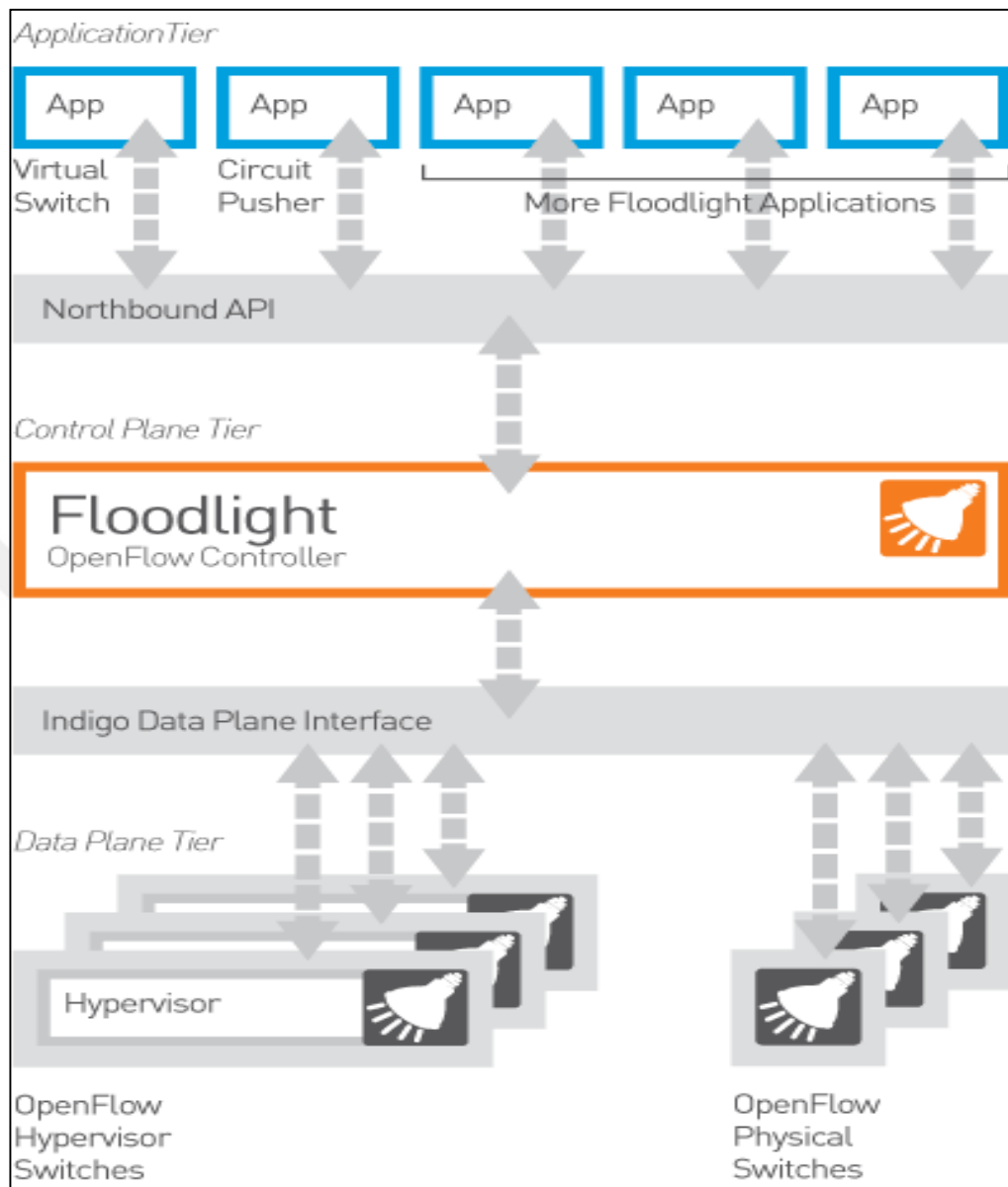


Figure 2.6. Floodlight SDN controller [9]

CHAPTER 3. ANOMALY DETECTION AND DOS ATTACKS

The goal of anomaly detection in the perspective of computer networks is to discover potentially harmful traffic. Anomalies are outlined as Forms in data that do not follow a clear notion of normal activities [10].

Anomaly detection used for intrusion detection firstly have been seen almost 40 years ago [41]. Nowadays, network anomaly detection is a very comprehensive and deeply searched subject but the issue that is still unsolved is finding a general method for a wide range of network anomalies. Commonly used intrusion detection systems against new malicious software are ineffective.

Anomalies might appear in a network for different reasons, for example because of malicious software, misconfigured network elements or software errors. If an anomaly occurs because of malicious packets (e.g. originating from malware), inspecting the packet's payload is an effective way to recognize abnormal traffic. Two types of payload-based classifiers exist: Deep Packet Inspection (DPI) and Stochastic Packet Inspection (SPI) [11].

Those two methods provide very accurate results; however, the computational costs are high. Thus, approaches that merely need header fields instead of packet payload are required. However, building a strict model which can isolate the normal network traffic is very difficult. Hence, detecting anomalies in network traffic is a difficult task [11]. Several machine learning algorithms are suited for this task. This section will give an overview on what can be considered an anomaly in SDN, control traffic will be treated, and possible attack scenarios described.

3.1. Methodological Overview

The first phase to building an anomaly detection tool is to define what kind of data will be used. One option is to aggregate the data based on network flows and/or periods of time [12]. An instance of the dataset would then be composed of header fields describing the network packets and additional meta-data such as the number of transmitted packets. Next, the dataset is analyzed to identify the attributes which seem relevant for classification. With better knowledge of the data, one can choose the appropriate techniques. For example, small datasets can be analyzed and labelled by experts, thus supervised machine learning is suitable. Conversely, it is difficult and very time consuming to label large datasets. The structure of the data allows for deciding a priori what algorithms might be appropriate for classification. However, a more accurate evaluation of the algorithms is necessary, e.g. with the precision/recall metric [13]. When predicting a Boolean output variable (normal or anomaly), two kinds of errors are possible: the case is an anomaly but was not classified as such (False Negative) or the case was wrongly classified as an anomaly (False Positive).

3.2. Using SDN For Anomaly Detection in User Traffic

To perform anomaly detection, data needs to be collected. In traditional networks, the data must be stored and aggregated at different network nodes. In SDN, the controller has a centralized view of the entire network, hence the data collection can be performed at a single point. Furthermore, OpenFlow switches update counters each time a packet is transmitted or received. The controller can therefore request the byte count and packet count for a given network flow. Previous work shows that SDN provides the opportunity to collect and aggregate data more easily [14, 15, 16]. The data that is collected can then be used for anomaly detection.

Several approaches remain conceivable when using SDN for anomaly detection. The OpenFlow protocol can be used to send packets to the controller, to inspect the payload (i.e. DPI). The advantage is that the packets are inspected centrally, without having to deploy the DPI-implementation on multiple network elements. This method, however,

has a downside: the network load is substantially increased, since the packets need to be sent across the network to the controller.

SDN has the potential to simplify the implementation of well-known anomaly detection tools. Mehdi et al. showed that algorithms like Rate-Limiting can easily be ported to SDN networks [17].

3.3. Deep Packet Inspection

Deep packet inspection is a method of checking the header and/or payload of Internet Protocol (IP) packets. It is, still, also used to show those architectural methods to network traffic monitoring that use DPI in an automated way; DPI is combined into fundamentally automated systems. Packet capture and additional analysis can be either distinct processes in both time and space or it can be joint in one task pipeline. The traffic capture method can work as a source of a PCAP file for additional Deep packet inspection based analysis [18].

For the advent of new applications, the so called Deep Packet Inspection technology has been extensively applied to numerous forms of networks, and have seen a speedy development. Administrators can apply policies in all layers and prevent malware and threats by parsing the payload of Internet Protocol (IP) packets.

3.4. DoS Attack and Defense Methods

A denial of service (DoS) attack is described as an effort made by an attacker to block legitimate users from using services provided by an application network or server [19]. This kind of attack can be launched in many ways, one way is sending crafted packets to cause the system to be crashed and exploit a certain software vulnerability in the target system [20]. Another way is by sending massive useless volumes of traffic to devastate and occupy the resources available for legitimate users to benefit from. In this thesis we are focusing on overwhelming the server and occupy the resources available in the server.

3.5. DoS and DDoS Attacks

In order to consume the target's resource, the capacity of traffic for the attack must be large enough. To deny services and achieve more complex attack detection, the attack is originated from several sources. This type of denial of service (DoS) attack is known as distributed denial of service attack (DDoS).

A standard distributed denial of service (DDoS) attack contains three main parts as shown in Figure 3.1. At the first step the attacker selects a group of vulnerable systems (we call them zombies) and installs attack systems in them. After the attack method is installed, the attacker has all the capability to launch attack commands to the zombies through using a secure channel to carry out the DoS attack on the target. The difficulty of this kind of attack increases due to the bots modifying the packets, generally spoofing the source. Therefore, it becomes even more difficult to trace the origin of the attack.

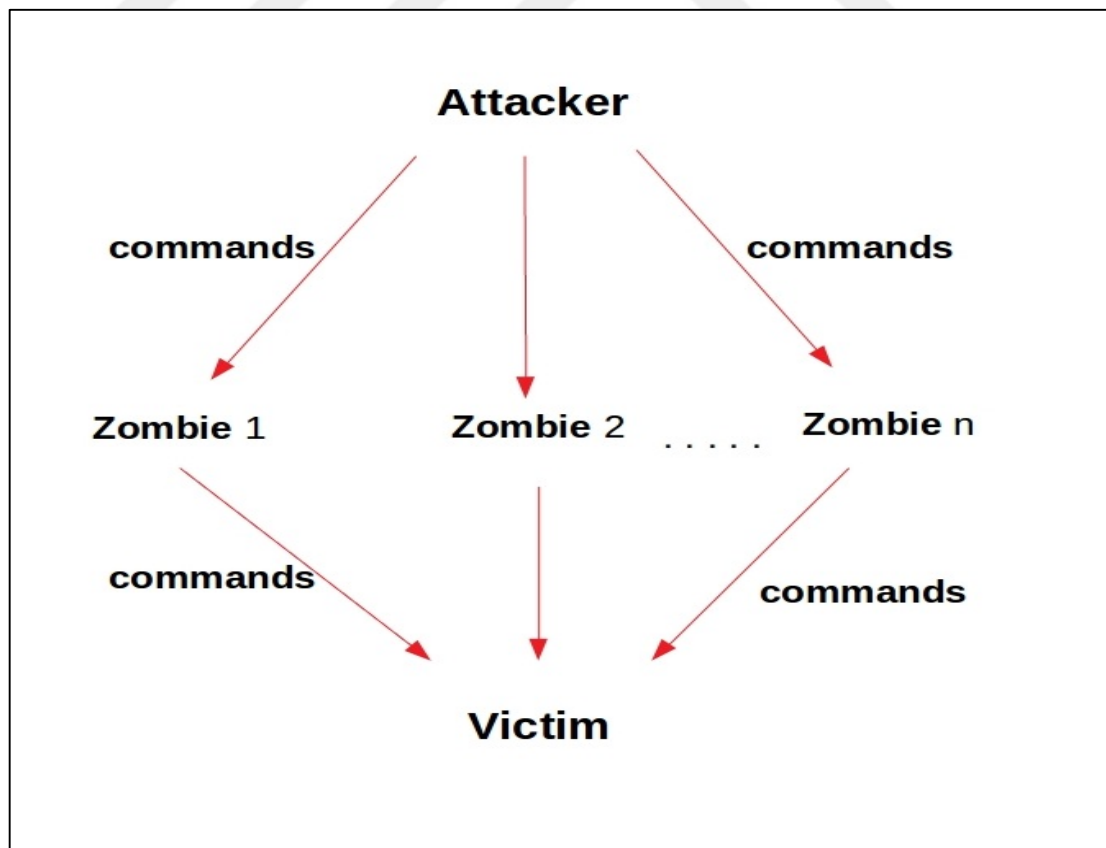


Figure 3.1. DDoS attack structure

Botnet is a group of bot systems, also known as zombie and the arrange of elements the attacker can launch to attack systems over them, and carry out combined attacks, is commonly known as botnet. A significant characteristic of botnets is the ability to update software from the attacker over the security channel between the bots and the attacker.

3.5.1. Attack classification

In order to plan a classification of DDoS attacks, we have to consider some characteristics of the attacks, as well as the means used to prepare and perform the attack, the features of the attack itself and the selection and the effects upon the victim. In this study, we will classify attacks depending on the victim type, and based on that classifying will be protocol attacks, logic attacks, and bandwidth attacks. There are some other classifications and they are mentioned in greater detail in [21, 22, 23].

3.5.1.1. Protocol attacks

To take advantage of the inherent design of common network protocols, the attacker continuously sends packets to the server at a particular rate. In other words, these kinds of attacks try to exploit the weakness of the system, bearing in mind the expected behavior of protocols such as ICMP, UDP, and TCP. A UDP flood attack is a protocol attack which has the purpose of bringing down the server by sending UDP packets, the victim will be forced to send back ICMP packets, but to an unreachable destination [24]. SYN flooding attacks, as the name implies, flood the server by sending SYN packets that consume its resources and fill up the backlog. Other examples are Smurf attacks [25]. and ICMP attack (This thesis focuses on this type of attack, which is further explained in Chapter 4).

3.5.1.2. Bandwidth attacks

Bandwidth attacks can consume all available data volume between an Internet Service Provider (ISP) and the target. To route from many resources to many destinations the

ISP networks need to have high bandwidth due to the heavy traffic that is required for routing. The connections between the victim and the ISP usually have less volume than the ones internal to an ISP, so when high volume of traffic generated from the ISP go through these connections, the links are filled up and legitimate traffic slows down. An attacker then can easily consume the bandwidth by transmitting any traffic to all network connection [26]. For example, high capacity of simple ICMP packets can consume the bandwidth [27].

3.5.1.3. Logic attacks

In software or logic attacks, a small number of malformed packets exploit known specific software bugs in an application or in the operating system of the target system. This can actually disable the target's machine with one or multiple packets. These kinds of attacks can be avoided by installing or updating the software that eliminates vulnerabilities [28]. or by adding specialized filter rules to filter out malformed packets [29]. In the ping of death attack, the attacker sends a ping message with the packet size over the internet. Other examples include Land attacks, Teardrop attack [24].

3.5.2. Defense classification

DoS defense methods have become one of the most significant challenges in network security. Therefore, a large number of defense taxonomy and classifications have emerged [22,23]. In this study, we will present four broad categories. The intention of this classification has been to highlight the core features of each group of defenses.

Attack Prevention: its purpose is to stop the attack before it can reach its targets (see 3.5.2.1); Attack Detection: this section aims to detect the attack when it occurs (see 3.5.2.2); Attack Source Identification: its purpose is locating the source of the attack (see 3.5.2.3); and Attack Reaction: its aim is to reduce or eliminate the effects of the attack (see 3.5.2.4).

3.5.2.1. Attack prevention

This kind of category aims to stop attacks before they actually cause damage, and tries to deny traffic that can be recognized as malicious, based on known patterns. The best place to put this kind of devices is in the edge routers and edge hosts, which implies fixing all the vulnerabilities of all Internet hosts that can be misused for an attack. Some useful methods to prevent DDoS attack against a target machine are:

Filtering: This measure implies installing ingress and egress packet filters on all the routers. In order to protect the target from attacks arriving to the network and prevent the network itself from being a potential attacker, filtering all the packets entering and leaving the network might be a good option.

Firewall: Before an attack is carried out, a firewall might be useful to filter out traffic according to the protocol, ports or incoming IP addresses. But, the problem is that firewalls cannot differentiate between an attack and legitimate traffic, and denying all traffic for a specific port or protocol is not appropriate. Only in those attacks in which the signature patterns are known, may these patterns be avoided. However, an insignificant variation or new attacks can make the attack go undetected.

Protocol Security: Addresses the problem of protocol design weaknesses in order to prevent Protocol Attacks such as a TCP SYN Attack, ICMP Ping Flood Attack, malformed packets, UDP Flooding, etc. [30].

3.5.2.2. Attack detection

When the attack is in process, an attack detection method must recognize if it is actually an attack or just legitimate traffic. Also, in an attack situation, legitimate traffic must flow without being misclassified and disrupted. False positive occur when the IPS reports certain benign activity as malicious. False negatives occur when the IPS does not detect and report actual malicious activity [31]. An effective attack

detection method must keep the balance between false positives and false negatives. There are basically two kinds of detection structures [32, 33].

Pattern Detection: An attack can constantly be detected by comparing incoming traffic with known attacks signatures stored in a database. These patterns are constructed by network security experts based on previous attacks. If the attack matches the database, this method becomes very efficient with almost no false positives. Problems arise when there are new attacks or slight variants that can dodge the defense. SNORT [33]. and Bro [34]. Are two commonly used pattern detection methods.

Anomaly Based Detection: It identifies malicious activity in a network by detecting anomalous network traffic patterns. Some network analysis behavior such as detecting the attacks based on the size of the packet, since those being too short violate specific application layers protocols. Rate-based detection is also an important network analysis. It perceives changes in the traffic flow, detecting floods by using a time-based model of normal traffic volumes. The parameters on which the defense method is based to detect the anomaly can be standard, they rely on protocol standards for example, an attack detection can detect half-open TCP connections and trained, which generates allowed threshold values normal conditions based on the system's behavior under normal conditions.

3.5.2.3. Attack source identification

Once an attack is detected, the best response is to block the attack traffic at its source. It aims at locating the attack sources regardless of whether the source address field in each packet contains correct or erroneous information. Once the attack detection phase is over, the IP attack traffic should be traced back to its source. This is taken care of in phase [20]. Unfortunately, it is not easy to track IP traffic down to its source. This is due to two aspects of the IP protocol. The first is the ease with which IP source addresses can be forged. The second is the stateless nature of IP routing, where routers normally know only the next hop for forwarding a packet, instead of the complete end-to-end route taken by each packet.

3.5.2.4. Attack reaction

Attack reaction tries to eliminate the effects of an attack and filter the attack traffic without disturbing legitimate traffic. The reaction to the attack must minimize the damage caused by the attack by developing a reaction scheme while the attack is in progress.

Rate Limiting: The rate of malicious traffic packets is reduced with this method when there is a high number of false positives and traffic has been identified as malicious by the detection methods. Max-Min fair share sets up maximum and minimum thresholds by the routers fixed by the servers. Level-K controls the traffic admission rates of the routers; k hops away the victim using a max-min fairness approach [35].

Filtering: Dropping the traffic conserved as unwanted or malicious is an effective way to prevent a DDoS attack. The problem is that some attacks use well-formed packets and legitimate requests to servers, making them non-filterable. There is also the risk of accidentally denying service to legitimate traffic. However, it is an efficient method against spoofed IP packets. Dropping spoofed incoming packets by ingress filtering [36]., identifying and dropping packets based on the change of the time-window-size, saving proved previously legitimate IPs are some of the attack reaction method on filtering [37].

CHAPTER 4. EXPERIMENTAL IMPLEMENTATION

In this chapter, we will present the experimental implementation in a virtual environment as well as the attack scenario to validate this testbed. As previously mentioned in Chapter one the scope of this work is to study how SDN can do anomaly detection and help to stop one of the most harmful attacks called denial-of-service (DoS). This work presents a network-based defense mechanism developed with the floodlight controller. To test this method, a virtual set-up is simulated, using the following technologies: We carried this experiment in a virtual environment using Mininet [38]. As a testbed, in Mininet virtual machine we have installed several things like sFlow monitor tool, Mongo database for storing the packets for later use in data loss prevention but it is out of the scope of this project, and we have created a system which has a graphical user interface where we can simplify the administration tasks Figure 4.1. is the login page of the developed system.

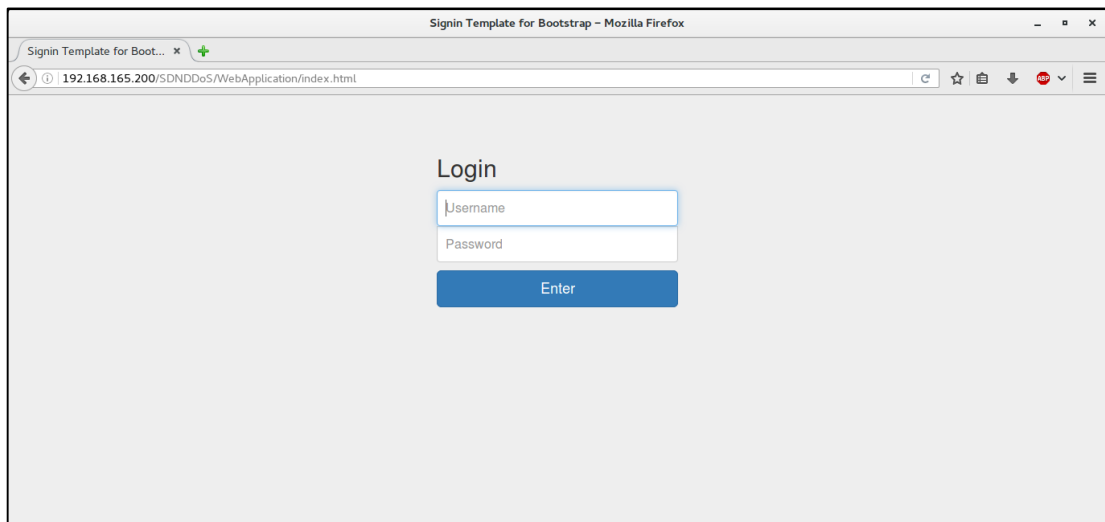


Figure 4.1. Anomaly detection system login screen

We developed the web based system using PHP as back-end and it has several modules, one of the modules that we created in our system is a firewall module shown in Figure 4.2. This firewall setting is simple, it sends commands that the user can provide in the textbox provided, and the example is shown in the figure. To enable the firewall module is made easy, only in the textbox provided write the command and click the enable button to be enabled. On the other hand, if you want to disable the firewall module, write the command then click disable button to disable the firewall module. If you enable the firewall with the default settings it means all the connections will be blocked and only the specified IP addresses will be allowed to pass through the network, and if you click disable with the default settings provided, all the communication will be allowed, unless otherwise you specify which part you want to not pass the network.

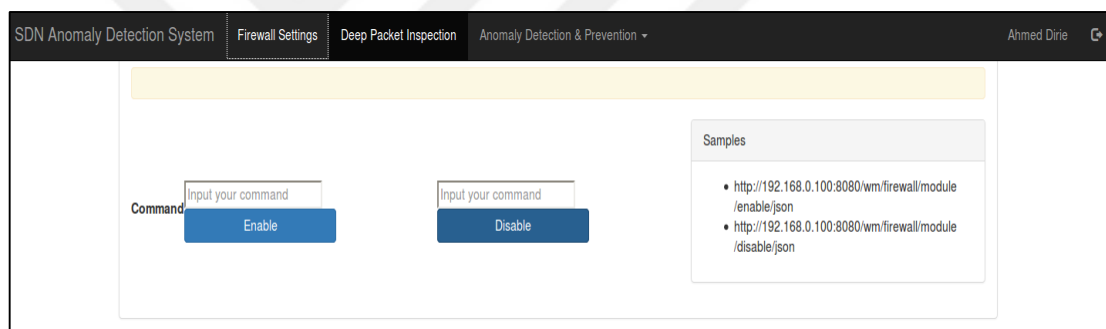


Figure 4.2. Firewall module settings

Our system also has a module doing deep packet inspection, where we are capturing all ongoing packets, the captured packets consists of the following parameters, source and destination MAC addresses, IPV4 source and destination, and finally transport layer whether it is TCP or UDP. The created testbed is doing full packet capture, and later we can investigate the payload of the packet, we can dig deep and do analysis to the packet in order to find any anomalies and suspicious attacks like SQL injection or cross site scripting (CSS) in the packet that is captured. A screenshot about the module is provided in the Figure 4.3.

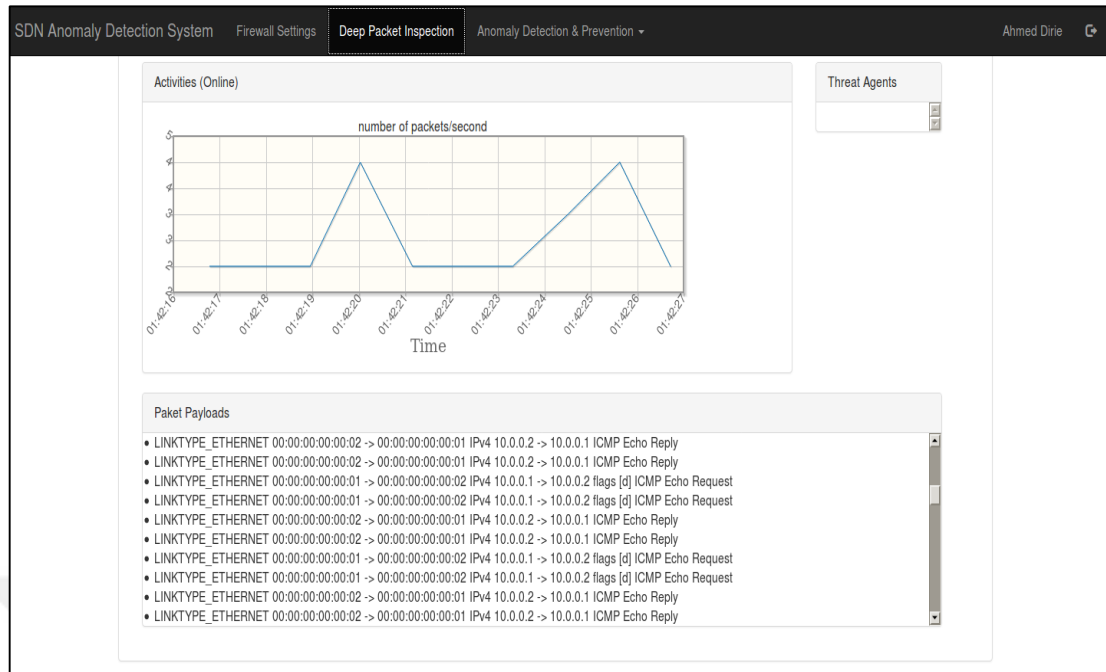


Figure 4.3. Deep packet inspection module

4.1. Tools Used for The Testbed

There are several tools that we used in this thesis. Mininet is an emulator program that will handle all the topology, OpenFlow-enabled switches, simulated hosts, as well as the servers, sFlow on the other hand will monitor ongoing packets and it will feed sFlow agents, on the background we are collecting all the data and we are storing it into mongo database, in case we need to do data loss prevention. In the following section, we will briefly describe them in detail.

4.1.1. Mininet

Mininet is a system that allows us to quickly emulate a big and complex network on the limited resources of a single computer. Mininet can be used as a standalone appliance or in virtual machine, Mininet can emulate a realistic network which is running in real Linux kernels. Mininet allows to experiment Software Defined Networks using OpenFlow and any controllers. This experiment and the creation of this network topology is done through Mininet [19]. Mininet the new archetype environment keeps this workflow by using trivial virtualization. Users can employ a

new prototype network, check it on big topologies with normal traffic, and then utilize the literal equivalent code and check the script into a real network. The following are some of Mininet features but not limited to:

- a. Flexibility: by using familiar programming languages and common operating systems.
- b. Deployability: to deploy a prototype on physical hardware does not require to change the code.
- c. Scalability: the archetype should scale well to the networks with hundreds of switches on one computer.
- d. Realistic: archetype performance should signify real performance with a high degree of confidence.
- e. Share-able: archetype created with Mininet should be easily shared with co-workers, who can then run and change our experiments.

4.1.2. sFlow

sFlow [39]. Is used for complex networks to monitor and manage traffics. It can be used in real hardware or virtual networks. It has a JSON API, which will inform the Floodlight controller about the ongoing packets. sFlow offers the data required to excellently manage and control usage of networks, confirming the services of the network to provide a competitive advantage. Here are a few examples of sFlow applications:

- a. Diagnosing, detecting, and fixing problems of the networks.
- b. Congestion management in real-time.
- c. Familiarity of applications mix and changes (e.g. Web).
- d. Audit trail analysis to identify unauthorized network activity and trace the source of denial-of-service attacks.
- e. Capacity and trending planning.
- f. Peering optimization and route profiling.

sFlow is sampling technology that converges the key needs for network traffic monitoring solution:

- a. sFlow is scalable, without affecting the operation of the core internet routers and switches it can monitor links of speed up to 10Gb/s and without adding significant network load.
- b. sFlow provides an active route of a wide network view and usage. It is a scalable method for computing network traffic, gathering, storing, and analyzing traffic data. This enables tens of thousands of interfaces to be observed from a specific location.
- c. sFlow is a low-cost solution. It has been implemented on a wide range of devices, from simple L2 workgroup switches to high-end core routers, without requiring additional memory and CPU.
- d. sFlow is an industry standard with a growing number of vendors delivering products with sFlow support.

Using sFlow to constantly observe flows of the traffic on all ports gives network wide visibility into the use of the network. This visibility replaces presumption, basically changing the way that network services are administered.

4.2. Environment

After the virtual environment is ready and well prepared, it is significant to carry out the essential tests to analyze its behavior. Therefore, a virtual network topology has been implemented through Mininet with the purpose of emulating a real environment. In this topology, some of the hosts will act as attackers and some of them will act as a server which behind the scenes is running an HTTP server based on Python.

4.2.1. Topology

Any system such as a Web server, FTP server, or Mail server, connected to the Internet and providing TCP/UDP based network services, is a real target to Denial of Service attacks [21]. In this implementation, a Web Server has been used as the victim of the attack.

To run this system several services should be run. Here we will present how to run each application with the commands associate with it. Firstly, we will start the Floodlight SDN controller, our SDN controller is listening to switch connection on port 6633, by running the command shown in the figure below Floodlight is up and running and waiting to be connected by any OpenFlow enabled switch.

```

mininet@mininet-vm:~/floodlight-0.91$ ./floodlight.sh
Starting floodlight server ...
OpenJDK 64-Bit Server VM warning: ignoring option PreBlockSpin=8; support was removed in 7.0_40
2016-05-01 06:23:29.306 INFO [n.f.c.m.FloodlightModuleLoader] Loading default modules
2016-05-01 06:23:30.299 INFO [n.f.c.i.Controller] Controller role set to MASTER
2016-05-01 06:23:30.316 INFO [n.f.c.i.Controller] Flush switches on reconnect -- Disabled
2016-05-01 06:23:32.179 INFO [n.f.l.i.LinkDiscoveryManager] Setting autoportfast feature to OFF
2016-05-01 06:23:32.645 INFO [o.s.s.i.c.FallbackCCProvider] Cluster not yet configured; using fallback local configuration
2016-05-01 06:23:32.646 INFO [o.s.s.i.SyncManager] [32767] Updating sync configuration ClusterConfig [allNodes={32767=Node
[hostname=localhost, port=6642, nodeId=32767, domainId=32767]}, authScheme=CHALLENGE_RESPONSE, keyStorePath=/etc/floodlight/
auth_credentials.jceks, keyStorePassword is unset]
2016-05-01 06:23:32.936 INFO [o.s.s.i.r.RPCService] Listening for internal floodlight RPC on localhost/127.0.0.1:6642
2016-05-01 06:23:33.554 INFO [n.f.c.i.Controller] Listening for switch connections on 0.0.0.0/0.0.0.0:6633
2016-05-01 06:23:48.203 INFO [n.f.j.JythonServer] Starting DebugServer on :6655

```

Figure 4.4. Launching floodlight SDN controller

After running the background process of the Floodlight SDN controller, here is a simple screenshot for the topology created, the created topology consists of three OpenFlow enabled switches and three emulated hosts connected to each other, and all the OpenFlow enabled switches are connected to Floodlight SDN controller. The graphical user interface of Floodlight SDN controller is made simple, it consists of a dashboard, topology, switches, and hosts. In the dashboard it shows how much RAM it consumes, how much CPU it runs on, and for how long the controller was up and running. In the topology section as shown in the figure below, it shows the exact topology created in SDN environment, how many hosts and how many switches are there, how they are linked to each other, and all the details about the topology is shown in this section. While in Switches it shows the switch IDs, and switch MAC address and the like, on the last section which is the hosts section, it shows the hosts IP address and host IDs. The only part was captured is the topology which is the most important part where it shows the whole topology created in our testbed.

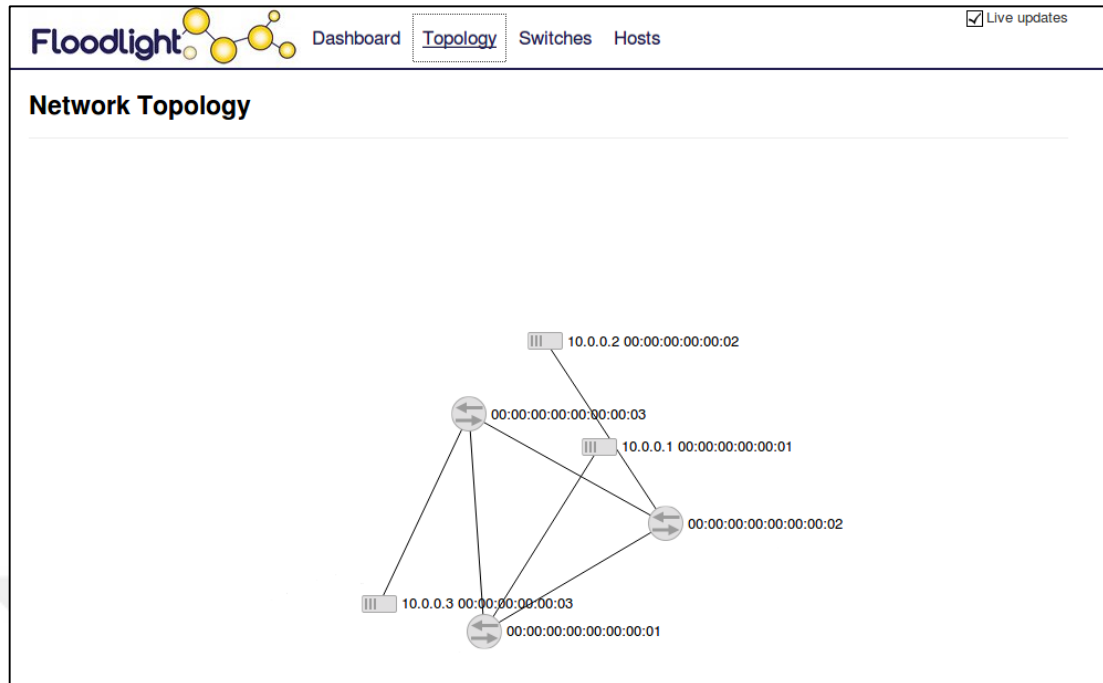


Figure 4.5. Floodlight built-in topology

In Figure 4.4. We will start sFlow as a monitoring tool. The command which is used to run the sFlow service in our system is shown below. To run the sFlow monitoring tool, the following command will be run in the command line, it is background process and it is listening to port 6343. To access it by using graphical user interface, we will write `http://localhost:8008` in any browser available in the system. Usage of sFlow and its benefit is discussed later in the upcoming sections.

```
mininet@mininet-vm:~/sflow-rt$ ./start.sh
2016-04-26T05:30:25-0700 INFO: Listening, sFlow port 6343
2016-04-26T05:30:25-0700 INFO: Starting the Jetty [HTTP/1.1] server on port 8008
2016-04-26T05:30:26-0700 INFO: Starting com.sflow.rt.rest.SFlowApplication application
2016-04-26T05:30:26-0700 INFO: Listening, http://localhost:8008
2016-04-26T05:30:26-0700 INFO: init.js started
2016-04-26T05:30:26-0700 INFO: init.js stopped
```

Figure 4.6. Starting sFlow monitor services

Figure 4.7. Shows a topology created through Mininet in a virtual environment. This topology is initialized and created by executing a Python script which is easily understandable and configurable, it is meant to create the topology with OpenFlow enabled switches, emulated hosts, and the connection between hosts and switches, also it creates the connection between switches and our SDN controller. In the following

section we will present the code in Python and we will explain briefly what each part does.

```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call
def myNetwork():
    net = Mininet( topo=None,
                  build=False,
                  ipBase='10.0.0.0/8')
    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                        controller=RemoteController,
                        ip='127.0.0.1',
                        protocol='tcp',
                        port=6633)
    info( '*** Add switches\n')
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch, dpid='00:00:00:00:00:00:00:02')
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch, dpid='00:00:00:00:00:00:00:03')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch, dpid='00:00:00:00:00:00:00:01')
    info( '*** Add hosts\n')
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1',
                    mac='00:00:00:00:00:01',defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='10.0.0.3',
                    mac='00:00:00:00:00:03',defaultRoute=None)
```

```

h2          =          net.addHost('h2',          cls=Host,          ip='10.0.0.2',
mac='00:00:00:00:00:02',defaultRoute=None)
info( '*** Add links\n')
net.addLink(h1, s1)
net.addLink(h2, s2)
net.addLink(h3, s3)
net.addLink(s1, s2)
net.addLink(s1, s3)
net.addLink(s2, s3)
info( '*** Starting network\n')
net.build()
info( '*** Starting controllers\n')
for controller in net.controllers:
    controller.start()
info( '*** Starting switches\n')
net.get('s2').start([c0])
net.get('s3').start([c0])
net.get('s1').start([c0])
info( '*** Post configure switches and hosts\n')
CLI(net)
net.stop()
if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()

```

The first part of the code will import the necessary libraries, including importing Mininet from mininet.net, the controller, open virtual switch controller and remote controller. Also, it is necessary to import hosts, nodes, open virtual switch kernel switch will be imported, and last but not least, command line interface (CLI) will also be imported in the above commands in python.

The second part of the code will define the topology, the controller which in this case is a remote controller, and the range of IP addresses, and transport protocol being used and on which port is listening is mentioned in this section of the code. Last but not least, switches are added into the network. In this testbed we are using open virtual switch (OVS), and later it will build the connection and it gives readable MAC addresses to the open virtual switches.

The last section of the code consists of adding hosts and assigning IP addresses to them, then building the connections between hosts and switches. It also shows for example how host 1 is connected to switch 1. Then later starting the function and running the network.

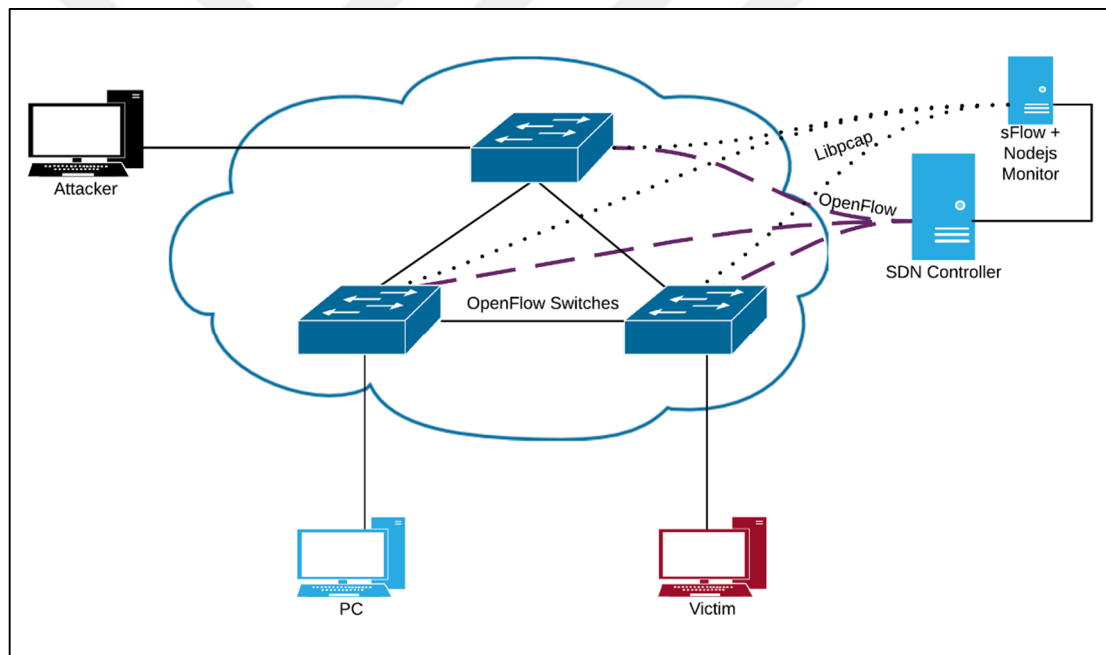


Figure 4.7. Topology created by python script

The black computer shown in the Figure 4.7. represent the attacker, whose attack mechanism or attack method is flooding the server with tens of thousands of pings called ping of death and it is a type of denial of service attack, i.e. sending too many echo requests which the server cannot handle because it has limited resources available in it. To check the availability of the server which will be overwhelmed the server and cause denial of service for legitimate users. The red one is the web server, in the server

there is a web services running on port 80 which is based on simple python script emulated in our testbed.

Overall, in the anomaly detection module that we created using in Nodejs is logically in the central point of the system where all packets pass through, the system does full packet capture, and triggers the Floodlight SDN controller when the packet reaches beyond the threshold value to block the source of the packet, in the coming figure below we will show some of the code that starts the anomaly detection server build by Node.js.

```
var express = require('express');
var app = express();
var http = require('http').Server(app);
var io = require('socket.io')(http);
var pcap = require("pcap");

//JS files (jqplot, jquery ...)
app.use(express.static(__dirname + '/JS'));

// sending index.html to the connected clients
app.get('/', function(req, res){
  res.sendFile(__dirname + '/index.html');
});

// Start server

var port=8001;
http.listen(port, function(){
  console.log('Listening ' + port);
});

//Web Socket
require('./pcap')(io);
```

Figure 4.8. Launching node.js server

The code above shows running Node.js and launching the server, it is an event based server execution procedure. Here we defined some important variables, like port number which is on port 8001, IO variable and it is using socket IO library.

```

1 var fs = require("fs");
2 var http = require('http');
3 var keys = 'inputifindex,ethernetprotocol,macsource,macdestination,ipprotocol,ipsource,ipdestination';
4 var value = 'frames';
5 var filter = 'outputifindex!=discard';
6 var thresholdValue = 100;
7 var metricName = 'ddos';
8 // mininet mapping between sFlow ifIndex numbers and switch/port names
9 var ifindexToPort = {};
10 var nameToPort = {};
11 var path = '/sys/devices/virtual/net/';
12 var devs = fs.readdirSync(path);

```

Figure 4.9. Mapping between mininet and sflow

In the above Figure we present some of the codes that is so important, we defined some variables like the threshold value, http variable which requires the library to be imported, the overall code above maps between our OpenFlow enabled switches with sFlow monitoring tool that we used as part of the research experiment.

4.2.2. Attack scenario

To carry out a ping flooding attack, the attackers will send tens of thousands of packets per second to the server, which means flooding it with echo request packets. Figure 4.10. illustrates a sequence diagram, where it explains the attack scenario and how the testbed deals with ping flooding attacks. In the sequence diagram shown below, the attacker sends normal ping, the ping goes to the OpenFlow enabled-switch, which is called packet-in. The switch checks the flow table, and because the destination is unknown to the switch (table miss), it forwards the packet to the SDN controller (Floodlight SDN controller) to ask the path to where it should send the packet. The SDN controller replies with a packet-out which includes information needed by the switch, then the switch stores the destination of the route into its flow table.

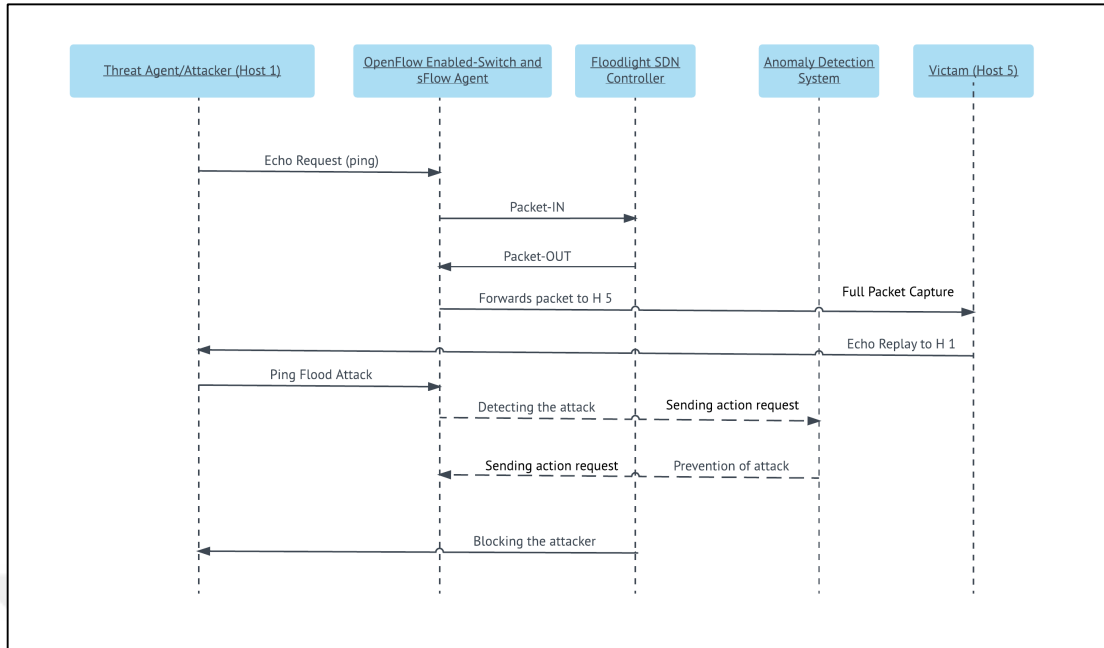


Figure 4.10. Sequence diagram

In the ongoing attack spectrum, we tried to capture in Wireshark to see the flooding attack, in the below Figure, we can see how many pings have been sent to the victim in less than a second. In Wireshark it is easy to illustrate in graphical interface, the source who originates the attack, the destination who receives the packet, and the protocol type as well as the length of the packet, Moreover the information related to the single packet is presented in the Figure below.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=1/256, ttl=64 (reply in 2)
2	0.005644623	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=1/256, ttl=64 (request in 1)
3	0.006384869	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=2/512, ttl=64 (reply in 4)
4	0.007381811	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=2/512, ttl=64 (request in 3)
5	0.007533676	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=3/768, ttl=64 (reply in 6)
6	0.008156660	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=3/768, ttl=64 (request in 5)
7	0.008326347	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=4/1024, ttl=64 (reply in 8)
8	0.008352368	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=4/1024, ttl=64 (request in 7)
9	0.008402559	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=5/1280, ttl=64 (reply in 10)
10	0.008448325	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=5/1280, ttl=64 (request in 9)
11	0.008497747	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=6/1536, ttl=64 (reply in 12)
12	0.008515873	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=6/1536, ttl=64 (request in 11)
13	0.008559738	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=7/1792, ttl=64 (reply in 14)
14	0.008575952	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=7/1792, ttl=64 (request in 13)
15	0.008618305	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=8/2048, ttl=64 (reply in 16)
16	0.008634088	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=8/2048, ttl=64 (request in 15)
17	0.008676827	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=9/2304, ttl=64 (reply in 18)
18	0.008693502	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=9/2304, ttl=64 (request in 17)
19	0.008736655	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=10/2560, ttl=64 (reply in 20)
20	0.008753023	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=10/2560, ttl=64 (request in 19)
21	0.008796086	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=11/2816, ttl=64 (reply in 22)
22	0.008812182	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=11/2816, ttl=64 (request in 21)
23	0.008855642	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=12/3072, ttl=64 (reply in 24)
24	0.008894797	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=12/3072, ttl=64 (request in 23)
25	0.008941841	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=13/3328, ttl=64 (reply in 26)
26	0.008959378	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=13/3328, ttl=64 (request in 25)
27	0.009003335	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=14/3584, ttl=64 (reply in 28)
28	0.009019704	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=14/3584, ttl=64 (request in 27)
29	0.009062063	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x08dc, seq=15/3840, ttl=64 (reply in 30)
30	0.009078508	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08dc, seq=15/3840, ttl=64 (request in 29)

Figure 4.11. Full packet capture in wireshark.

4.3. Results

After running the virtual environments and executing the attack, we will test our system. We fire up everything and here is the screenshot for doing anomaly detection. In the following Figure we are monitoring the packets checking for anomalies. As we can see there are packets going on and we defined a static value for the threshold. If the packets reach beyond the static value, it will indicate anomalies and trigger the SDN controller to take an action to drop the specified machines, causing the denial of service attack.

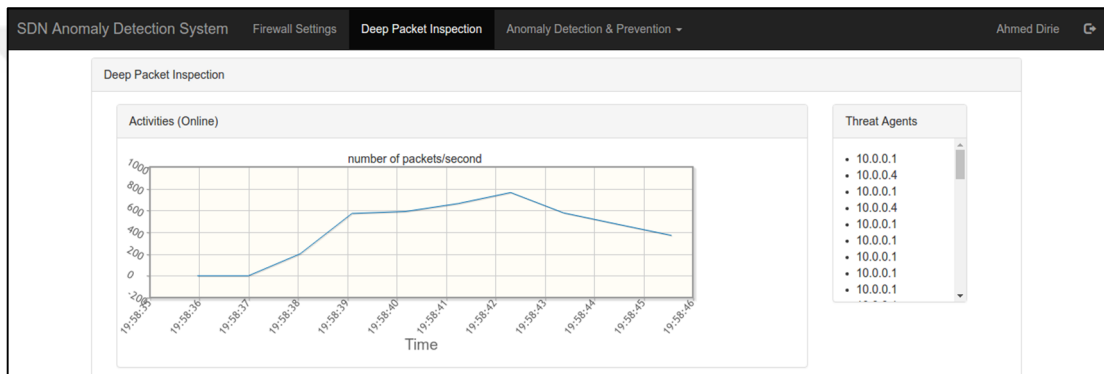


Figure 4.12. Anomaly detection in SDN.

Then finally, after launching the denial of service attack the mitigation script will be generated to block the attacker from sending any further packets to the victim in the network. Figure 4.13. shows the dropping down the packets.

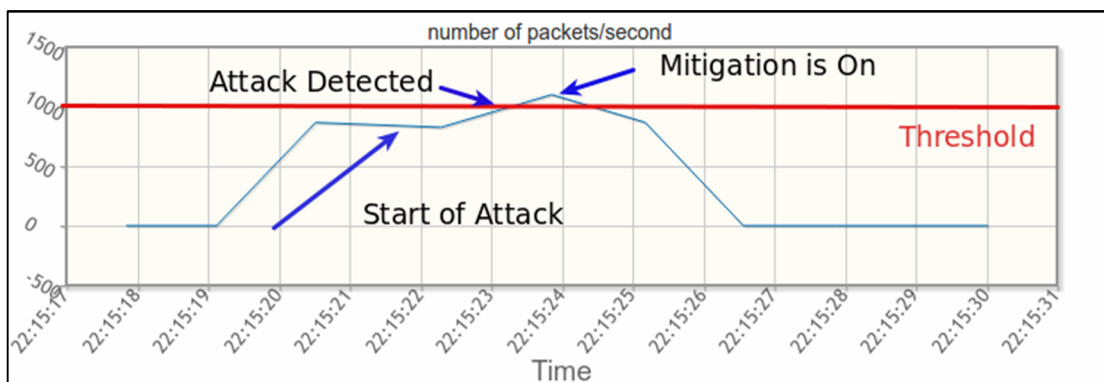


Figure 4.13. Mitigation of DoS attack.

CHAPTER 5. CONCLUSION AND FUTURE WORK

In this final chapter, we will present the conclusion and the future work for this study. Software Defined Networking (SDN) technologies help centralize the control plane which creates many suitable benefits. Firstly, network configuration and policies can be specified at a single location, decreasing an otherwise difficult management overhead. Secondly, the same applies for security concepts and fault detection facilities, which uses from centralized processing of all network events.

This work shows the Anomaly Detection mechanism in the new era of networking called Software Defined Networking by leveraging Deep Packet Inspection (DPI), to test our project, we have used a virtual environment with Mininet, and we have taken data results graphics with the help of sFlow and the created system.

Using static value for thresholding in anomaly detection can be improved by using dynamic value instead of static, for this study we present when the packets reach beyond the baseline it will identify it as anomalies, hence the packet will be dropped and no longer will be accessible for that device in the network.

Mininet is the emulation platform which helps us to simulate Software defined networking topology easily, we have used Floodlight SDN controller to help us control the entire network and command it from one place, sFlow is a monitoring tool which takes sample for flowing packets and represent it in a graphical user interface.

We have created a graphical user interface which simplifies the overall control of the network by simply clicking and behind the scenes the required commands will be generated. DoS attacks is one of the most harmful attacks in the history of computing, defeating such kind of attack requires a lot of work and study, we presented in this

study a way to stop this attack, with dropping the packets generated from the attacker if it reaches beyond the threshold value.

We can conclude in this study that SDN helps defeat such an attack by logically centralizing the whole network in one place, and all necessary commands will be written in the controller which on the other hand controls and monitors the network, OpenFlow on the other hand is the suitable protocol which takes the instructions from the controller and pass them to the physical or virtual forwarding device.



REFERENCES

- [1] Kreutz, D., Ramos, F., Esteves, P., Verissimo, C., Uhlig, S., Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.
- [2] O.N.F., Software-defined networking: The new norm for networks, ONF White Pap., vol. 2, pp. 2–6, 2012.
- [3] Mckeown, N., Anderson, T., Balakrishnan, H., OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM*, vol. 38, no. 2, pp. 69–74, 2008.
- [4] <http://www.bigswitch.com>, Access Date: 13.08.2017.
- [5] <http://www.bigswitch.com>, Access Date: 13.07.2017.
- [6] <https://www.opendaylight.org>, Access Date: 13.06.2017.
- [7] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., Mckeown, N., Shenker, S., Nox. *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, p. 105, 2008.
- [8] <https://openflow.stanford.edu/display/Beacon>, Access Date: 13.05.2017.
- [9] <https://floodlight.atlassian.net/wiki>, Access Date: 13.04.2016.
- [10] Chandola, V., Banerjee, A., Kumar, V., Anomaly detection: A survey. *ACM Comput. Surv.*, 15:1–15:58, 2009.
- [11] Valenti, S., Rossi, D., Dainotti, A., Pescape, A., Finamore, A., Mellia, M., Data Traffic Monitoring and Analysis, volume 7754 of *Lecture Notes in Computer Science*, pages 123–147, 2013.
- [12] Callegari, C., Coluccia, A., D'alconzo, A., Ellens, W., Giordano, S., Mandjes, M., Pagano, M., Pepe, T., Ricciato, F., Ziuraniewski, P., A methodological overview on anomaly detection., volume 7754 of *Lecture Notes in Computer Science*, pages 148–183, 2013.

- [13] Manning, C., Raghavan, P., Schütze, H., Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA, 2008.
- [14] Zhang, Y., An Adaptive Flow Counting Method for Anomaly Detection in SDN. In Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT '13, pages 25–30, 2013.
- [15] Moshref, M., Yu, M., Govindan, R., Resource/accuracy tradeoffs in software-defined measurement. In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, pages 73–78, 2013.
- [16] Yu, M., Jose, L., Miao, R., Software defined traffic measurement with opensketch. In Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), pages 29–42, 2013.
- [17] Mehdi, S., Khalid, J., Khayam, S., Revisiting traffic anomaly detection using software defined networking. In Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection, RAID'11, pages 161–180, 2011.
- [18] Kumar, S., Sehgal, R., Bhatia, J. S. Hybrid honeypot framework for malware collection and analysis. In Industrial and Information Systems (ICIIS), 2012.
- [19] http://www.cert.org/tech_tips, Access Date: 02.03.2016
- [20] Peng, T., Leckie, C., Ramamohanarao, K., Survey of network-based defense mechanisms countering the dos and ddos problems. ACM Computing Surveys (CSUR) 2007.
- [21] Huusain, A., Heidamann, J., Papadopoulos, C., A framework for classifying denial of service attacks. In Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications, ACM, pp. 99-110, 2003.
- [22] Mirkovic, J., Reiher, P., A taxonomy of ddos attack ddos defense mechanisms. ACM SIGCOMM Computer Communication Review, 39-53 2004.
- [23] Specht, S. M., Lee, R. B., Distributed denial of service: Taxonomies of attacks, tools and countermeasures. In ISCA PDCS, pp. 543-550, 2004.

- [24] Gupta, B., Joshi, R. C., Misra, M., Defending against distributed denial of service attacks: issues and challenges. *Information Security Journal: A Global Perspective.*, 224-247, 2009.
- [25] <http://www.cert.org/historical/advisories.>, Access Date: 01.12.2016
- [26] Kumarasamy, S., Gowrishankar, A., An active defense mechanism for tcp syn flooding attacks. 2012.
- [27] Bellovin, S. M., Leech, M., Taylor, T., Icmp traceback messages. Internet Engineering Task Force, Marina del Rey, Calif. 2003.
- [28] Srivastava, A., Gupta, B., Tyagi, A., Sharma, A., Mishra, A., A recent survey on ddos attacks and defense mechanisms. In *Advances in Parallel Distributed Computing*. Springer., pp. 570-580, 2011.
- [29] Molsa, J., Mitigating denial of service attacks: A tutorial. *Journal of computer security.*, 807-837, 2005.
- [30] Chang, R. K., Defending against flooding-based distributed denial of service attacks: A tutorial. *Communications Magazine, IEEE.* 42-51, 2002.
- [31] Carl, G., Kesidis, G., Brooks, R. R., Rai, S., Denial of service attack detection techniques. *Internet Computing, IEEE.*, 82-89, 2006.
- [32] Keshariya, A., Foukia, N., DDoS defense mechanisms: a new taxonomy. In *Data Privacy Management and Autonomous Spontaneous Security*. Springer., pp. 222-236, 2010.
- [33] Roesch, M., Snort: Lightweight intrusion detection for networks. In *LISA.*, vol. 99, pp. 229-238, 1999.
- [34] Paxson, V., Bro: a system for detecting network intruders in real-time. *Computer networks.*, 2435-2463, 1999.
- [35] Yau, D. K., Lui, J., Liang, F., Yam, Y., Defending against distributed denial of service attacks with max-min fair servercentric router throttles. *IEEE/ACM Transactions On Networking (TON).*, 29-42, 2005.
- [36] Ferguson, P., Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. *Amaranth Networks Inc.* 2000.
- [37] Peng, T., Leckie, C., Ramamohanarao, K., Protection from distributed denial of service attacks using history-based ip filtering. In *Communications. ICC'03. IEEE*, pp. 482-486, 2003.

- [38] Lantz, B., Heller, B., Mckeown, N., A network in a laptop: rapid prototyping for software-defined networks., pp. 1–6, 2010.
- [39] <http://www.sflow.org.>, Access Date: 08.04.2016.
- [40] Siris, V., Papagalou, F., Application of anomaly detection algorithms for detecting SYN flooding attacks. *Comput. Commun.*, vol. 29, no. 9 SPEC. ISS., pp. 1433–1442, 2006.
- [41] Denning, D., An intrusion-detection model. *IEEE Trans. Softw. Eng.*, 13, 222–232, 1987.



RESUME

Ahmed M. Dirie was born in 16.02.1989 in Yunbu Albahr in Kingdom of Saudi Arabia. He completed his primary school in Alhuda Primay School in 2004, he studies his secondary school at Hamdan Secondary School, 2007. In 2012, he received his first degree in Information and Communication Technology (ICT) from Admas University College, Somaliland. Ahmed starts his career as Mathematic Teacher in Iftin Schools, in 2011. In 2012 Ahmed starts his role in Information Technology Administrator at CAC Financial Service, Hargeysa, Somaliland. Ahmed Holds multiple certification which improve his hands-on skills from Cisco Company, CCNP Cisco Certified Networking Professional in 2016. CCNA Security and CCNA Route & Switch in 2015 and 2013 respectively.