REAL-TIME DENIAL OF SERVICE ATTACK DETECTION AND MITIGATION ON CONTROLLER IN SOFTWARE DEFINED NETWORK ENVIRONMENT

BILAL ISHFAQ

FACULITY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2016

REAL-TIME DENIAL OF SERVICE ATTACK DETECTION AND MITIGATION ON CONTROLLER IN SOFTWARE DEFINED NETWORK ENVIRONMENT

BILAL ISHFAQ

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

FACULITY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2016

UNIVERSITY OF MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Bilal Ishfaq

Registration/Matric No: **WGA140052**

Name of Degree: **MASTER OF COMPUTER SCIENCE**

Title of Dissertation: **Real-Time Denial of Service Attack Detection and Mitigation on Controller in Software Defined Network Environment.**

Field of Study:

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:

Designation:

ABSTRACT

Software Defined Network (SDN) decouples the control plane from the data plane to provide logically centralized control of the network. The control plane is considered as a brain of the network that controls the entire network. Due to such a unique feature, the control plane becomes the central point of attraction to different adversaries in SDN. If the controller is malfunctioned by the attacker than the whole operation of the SDN will be affected. The DoS attack is one of the attacks which affect the controller in the control plane in terms of network and computational resources. In this work, the focus is on the computational aspect of the controller and proposed a solution which assists to detect the attack at its early occurrence. The limitations in early proposed methods, such as early detection of DDoS attack and time-based DDoS attack detection methods, only detect the attack at controller, however it does not provide any information about its solution, such as how to handle these attacks. The objective of this study is to protect the SDN controller from DoS attack that will prevent the controller from being unreachable. The proposed technique not only detects any DoS attacks but also mitigate in real-time. This proposed technique is a lightweight solution which consumes less controller resources in detecting and mitigating the DoS attack. The DoS policy of the attack is implemented which blocks the traffic coming from the malicious node in SDN.

ABSTRAK

Software Defined Network (SDN) memisahkan planar kawalan daripada planar data untuk menyediakan rangkaian kawalan pusat yang logik. Planar kawalan berfungsi sebagai pusat rangkaian yang mengawal seluruh rangkaian. Di sebabkan oleh ciri yang unik tersebut,lapisan kawalan menjadi titik utama telah menarik serangan keatas SDN. Sekiranya pusat kawalan diserang oleh penyerang, maka kesemua operasi SDN akan terjejas. Penyerang DoS merupakan salah satu penyerang yang akan mempengaruhi pusat kawalan sumber rangkaian dan komputasi di planar kawalan. Dalam kerja kami, kami tertumpu pada aspek komputasi pada pusat kawalan dan mencadangkan penyelesaian yang dapat membantu mengesan penyerang pada peringkat awal. Kaedah yang dicadangkan sebelum ini, seperti pengesan awal penyerang DDoS dan kaedah pengesan penyerang DDoS berdasarkan masa mempunyai batasan kerana mereka hanya berupaya mengesan penyerang di pusat kawalan. Di samping itu, kaedah-kaedah tersebut tidak berupaya menyediakan sebarang maklumat tentang cara menyelesaikan dan mengendali serangan. Objektif kajian kami adalah untuk melindungi pusat kawalan SDN daripada serangan DoS yang akan menghalang pusat kawalan daripada menerima maklumat daripada sumber. Kaedah yang dicadangkan oleh kami bukan sahaja dapat mengesan segala penyerang DoS malah dapat mengurangkan penyerang di masa sebenar. Kaedah yang dicadangkan merupakan penyelesaian yang mudah kerana ia mengurangkan pengunaan sumber daripada pusat kawalan dalam mengesan dan mengurangkan serangan DoS. Kami telah melaksanakan polisi serangan DoS dan menghalang laluan trafik daripada nod yang berniat jahat di SDN.

ACKNOWLEDGEMENTS

First of all, I am thankful to Almighty Allah who has given me the power to study. I would like to offer special thanks to my supervisor: Dr. Rosli Salleh for his invaluable guidance, supervision, and encouragement to me throughout this research. He not only provided me helpful suggestions, but also accepted responsibility to oversee this research, and guided me to the successful completion of this thesis. This thesis would not have been produced without his invaluable advice, excellent knowledge, unceasing support and enormous patience.

I would like to express my sincerest gratitude and appreciation to my parents for their endless love and support during my life. Without their moral support, this dissertation would never have been completed. Last but not least, to my all sisters, their love and encouragements made this thesis possible. I would like to express my deep appreciation to my dear lab friends, who provided so much support and encouragement throughout this research and studies process. I wish them all the best in their future undertaking.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

BGP	Border Gateway Routing Protocol
DoS	Denial of Service
FSM	Finite State Machine
IP	Internet Protocol
IRP	Internet Routing Protocol
SDN	Software Defined Network
TCP	Transmission Control Protocol
TLS	Secure Socket Layers

CHAPTER 1: INTRODUCTION

Software Defined Network (SDN) decouples the control plane from the data plane to provide logically centralized control of the network. The control plane is considered as a brain of the network that controls the entire network. Due to such a unique feature, the control plane becomes the central point of attraction to different adversaries in SDN. If the controller is malfunctioned by the attacker than the whole operation of the SDN will be affected. The DoS attack is one of the attacks which affect the controller in the control plane in terms of network and computational resources. In this work, the focus is on the computational aspect of the controller and proposed a solution which assists to detect the attack at its early occurrence. The limitations in early proposed methods, such as early detection of DDoS attack and time-based DDoS attack detection methods, only detect the attack at controller; however it does not provide any information about its solution such as how to handle these attacks. The objective of this study is to protect the SDN controller from DoS attack that will prevent the controller from being unreachable. The proposed technique not only detects the DoS attacks but also mitigate in real-time. This proposed technique is a lightweight solution which consumes less controller resources in detecting and mitigating the DoS attack. The DoS policy of the attack is implemented which block the traffic coming from the malicious node in SDN.

This chapter presents the hypothetical framework and motivation for our planned research. First, the background and motivation to undertake the research are presented. Then, the statement of problem and objective of the research, and proposed methodology are also presented. Finally, the outline of the thesis is highlighted.

This chapter consists of six sections. Section 1.1 discusses the research background. Section 1.2 explains the proposed work importance by highlighting the motivation factors. Section 1.3 addresses the statement of problem. Section 1.4 lists the aim of research and presents the objectives. Section 1.5 sums up the research methodology and section 1.6 describe the thesis layout.

1.1 Background

Computer network is made up of computers connected to each other for sharing the resources. These computers are connected to each other through network devices, such as switches and routers and considered as the reliable and fastest way of communication between computers. However the conventional network devices are vendor proprietary. These devices make the network administrator job challenging, for example, in conventional network, each device is configured separately. In addition to this, if network devices are products of different vendors then configuration method is also different which results the compatibility issue. To overcome these problems, SDN is introduced which control the network in different way (Mousavi, 2014).

SDN stands for Software Defined Network is the separation of control plane and forwarding plane. The forwarding plane, also known as data plane, consists of devices such as router and switches. The network traffic is forwarded through these devices according to defined rules. The control plane, connected to the data plane via southbound interface, pushes the flow entries on data plane to forward the traffic (Kreutz et al., 2015). By separating the control plane and data plane, the network operator can see and control the network from single point. The main concept of Software Defined Networking is the separation of control plane from data plane.

In SDN, switches are considered as the forwarding devices which forward the packets according to the rule defined by controller. The decision of forwarding the

packet towards the specific destination is made by control plane. Controller on control plane defines the flow rules on switches and each forwarding packet is checked against the flow rule (Dao, Park, Park, & Cho, 2015; Giotis, Argyropoulos, Androulidakis, Kalogeras, & Maglaris, 2014).

For SDN architecture, different threat vectors have been identified by researchers, e.g. spoofing, tampering, information disclosure, denial of service and access attack (Casado et al., 2006). Some threat vectors are more similar as for existing conventional networks. SDN specific threat vectors can affect the control plane, data plane, northbound interface, southbound interface and application plane (Kreutz et al., 2015). To prevent the control plane from network attacks, there is a need of defining a network attack prevention policy which is the ultimate objective.

1.2 Motivation

Controller is considered as the brain of the network because all the forwarding decisions take place at control plane and if due to DoS attack the connection between controller and switches is lost, the SDN architecture will no longer be functional.

DoS attack affect the controller in terms of resource consumption as large number of packets with spoofed IPs is sent to the network, as a result, the controller resources exhaust and controller is no longer be functional (Dharma, Muthohar, Prayuda, Priagung, & Choi, 2015).

Since DoS attack can fail the operations of controller which results the failure of SDN architecture. Therefore, it is important to make the SDN controller secured from DoS attack. To secure the controller, DoS policy is defined using Pyresonance platform. Pyresonance changes the configuration of network by aid of state machine if event occurs.

1.3 Statement of Problem

In SDN, controller is considered as desired point to attackers for making the entire network functionality abnormal. DoS attack is ideal to make the controller fall down as the flooding of packets by DoS attack from compromised host exhaust the processing capabilities of the controller. As a result, the controller will no longer be functional, collapse the SDN architecture.

1.4 Research Aim and Objective

SDN architecture presents number of weak points vulnerable to network attacks. In the existing studies, the main concern was to figure out these weak points for DoS attack. As a result, controller is found as a weak point. The aim of this proposed research is:

"To propose a technique which detect and mitigate the DoS attack using DoS policy in real time on controller in SDN environment during the attack is occurred". The DoS policy is chosen as it mitigates the DoS attack on controller in real time.

. The main objectives of this research are as follow.

i. To review the current state of the art techniques related to DoS attack detection and mitigation on controller in SDN.
ii. To design an automated system which will detect DoS attack on controller in SDN environment and also mitigate the attack on controller in order to prevent it and block the traffic coming from malicious host in real time by using DoS policy.
iii. To implement the designed automated system by the use of Mininet emulator.

1.5 Proposed Methodology

To identify the limitations in existing proposed techniques, this research studied the state-of-the-art techniques related to SDN security and proposed a research problem. The investigation of research problem is carried out by studying the existing proposed techniques. After finding the research gap in existing proposed techniques, the design is proposed for detection and mitigation. Then performed experiments using Mininet emulator to ensure that the proposed technique functioning is different and more efficient than existing techniques.

In this work, the DoS attack is launched towards the controller which is further connected to four switches and six hosts. The DoS attack is launched by one compromised host towards the controller. CPU resource consumption is used for evaluation.

It is desirable that the DoS attack should be detected before the completely failure of controller and then appropriate technique should be applied to mitigate the DoS attack.

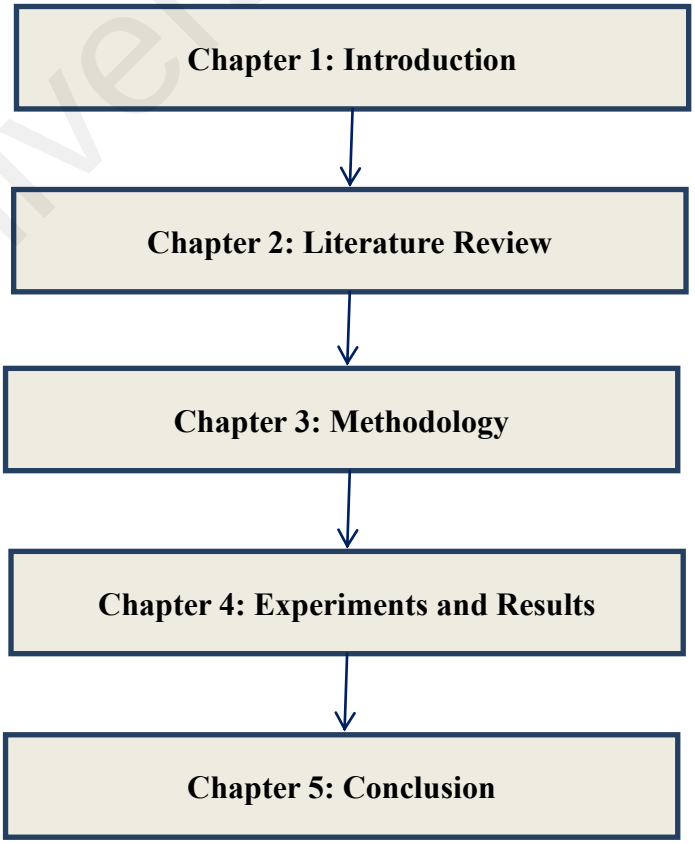
In this proposed work, sFlow-rt is used as a flow collector besides controller. sFlow-rt is used to collect all the network measurement. A time based threshold is defined in sFlow-rt. When abnormality in the traffic is observed greater than the defined threshold time, sFlow-rt sends the signal of activation to the Pyresonance. Pyresonance application is used to change the configuration of network when event occurs. Pyresonance is the implementation of state machine at pyretic. In this proposed work, the DoS policy is defined which blocks the traffic coming from compromised host by detaching it from the network. However, the remaining hosts are able to communicate with each other even during the occurrence of attack.

The test is conducted by connecting the sFlow-rt to the controller for detection of DoS attack and DoS policy is defined on controller, using Pyresonance, to mitigate the DoS attack.

1.6 Thesis Organization

This research work is categorized into five chapters as follow.

Chapter 2 discusses the SDN architecture and the components of SDN. After that it discusses the components vulnerable to DoS attack and then highlights the different techniques for detection and mitigation of DoS attack. The literature review and first objective of this proposed research is discussed at the end of this chapter. In Chapter 3, solution proposed to the DoS attack is discussed. In chapter 4, the explanation of experimental setup is given and finally obtained the results of solution. Chapter 5 is concluding chapter. It gives the overview of research and discusses the results. Thesis layout can be comprehended with the help of following Figure which highlights the headings of all chapters.



CHAPTER 2: LITERATURE REVIEW

This chapter includes SDN introduction including its origin and emerging technologies. It also discusses the conventional network, comparison with conventional network and OpenFlow. Then it explains the threat vectors affecting the SDN architecture and highlights the security issues. Finally, it reviews the existing SDN controller security related techniques in related work and highlight the limitations and significant issues to present the research gap.

The Chapter includes ten sections. The section 2.1 gives the brief introduction of SDN. The following section which is section 2.2 explains the origin of SDN. Next to this section, there is section 2.3 which discusses the emerging technologies. Section 2.4 compares the conventional network with SDN. Section 2.5 presents the infrastructure of SDN. In section 2.6, the detailed explanation of OpenFlow is given. Section 2.7 discusses the related work. Section 2.8 highlights the studies related to DoS attack on controller. Section 2.9 identifies the research gap. Finally, section 2.10 summarizes the chapter.

2.1 Introduction

SDN, stands for Software Defined Network is the separation of control plane and forwarding plane, as shown in Figure 2.1 (source: online), where forwarding plane, also known as data plane, consists of devices such as router and switches. Traffics are forwarded by these devices according to defined rules. The control plane, connected to the data plane via southbound interface, pushes the flow entries on data plane to forward the traffic. By separating the control and data plane, the network operator can see and control the network from single point (Salman, Elhajj, Kayssi, & Chehab, 2016). The application plane is connected to the controller via northbound interface. The

application plane consists of applications defined by users. The application is software which is designed by user for SDN environment.

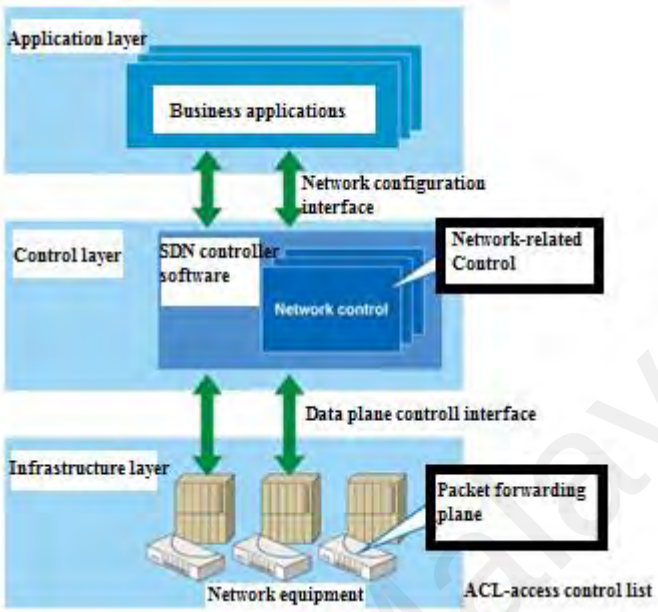


Figure 2.1: SDN architecture (Source: Online)

SDN advantages can be viewed by comparing it with static architecture of traditional networks as SDN centralizes the network control. In conventional networking, the control plane and data plane are on the same device. However, in SDN, data plane and control plane are separated as shown in Figure 2.2 (Source: Online). The data plane forwards the packets towards the destination according to the flow rules defined by controller. Control plane is connected to the application plane via northbound interface which resides at the top of control plane (Salman et al., 2016).

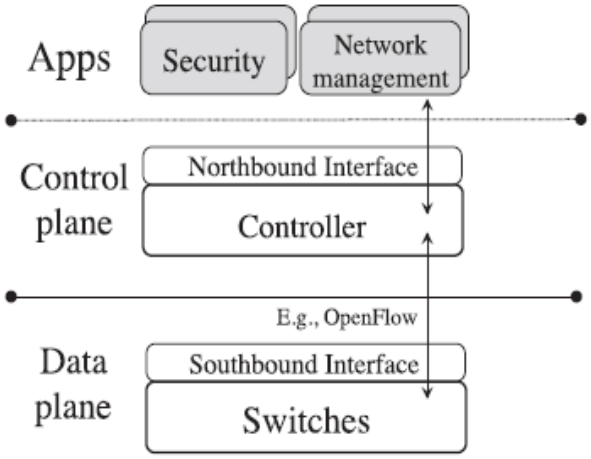


Figure 2.2: SDN components (Source: Online)

2.2 SDN Origin

Originally SDN architecture is inspiration of distributed system. However the configuration of distribution system is volatile and buggy. To overcome this unpredictability, enormous amount of researches were performed. In conclusion to all studies, it was observed that one control point can solve the problems of unpredictability and buggy configurations. IRP (Internet Routing Protocol) and BGP (Border Gate way Routing Protocol) used this idea of centralization. By the passage of time, this idea was further generalized with the name 4D architecture. The name 4D architecture (Prasad, 2014), represents the four layers of architecture. The name of these layers is as follow.

i. Data Plane
ii. Discovery Plane
iii. Dissemination Plane
iv. Control Plane

Each of these planes has different functionality. Data plane is only used to forward data packets to the destination. Discovery plane directly report to the control plane and

control plane knows the available resources whereas dissemination plane is for the topology discovery. The fourth plane, which is considered as the brain of the network, is the top layer responsible for all the traffic flow of network. To simplify the 4D architecture, SDN was evolved in which all the network devices can be controlled by single controller and devices are connected to the controller via OpenFlow protocol.

SDN history started right after java which is released by Sun Microsystems in 1995. The very first project of SDN was GeoPlex proposed by AT&T (Vanecek, Mihai, Vidovic, & Vrsalovic, 1999). GeoPlex is nothing except middleware which uses operating systems, connected to the internet, running on computers. It is a platform of services and manages the networks. AT&T required switches that can be reprogrammed which was the limitation in GeoPlex project. The switches used in this project were not reprogrammable and was the first barrier to SDN (Lerner, Vanecek, Vidovic, & Vrsalovic, 2006).

To overcome this barrier, Mark Medovich started a company with the name WebSprocket in 1998 and launched soft switch which was java sported and reconfigure able in real time. Later in 2000, the same company launched VMFoundry and VMServer. By the use of VMFoundry, the image of conventional network devices were preloaded and then connected to the VMServer by deploying on the network using TCP or UDP. Hence, WebSprocket controller consisted of code which could change or rewriteable (Lerner et al., 2006).

2.3 Emerging Technologies Contribution to SDN

SDN introduces the programming which makes the network easier to manage and also it facilitates the flexibility which allows the addition of new services without major changes. All these factors make the SDN economical.

2.3.1 Centrally Managed

In the beginning, both the data plane and control plane were sharing the one channel for operations. To make it clearer, consider the example of "IN-BOUND SIGNALING". This example is specifically for phone network in which the control and voice sharing one channel. Sharing a one channel was not secure which is considered as disadvantage. After that, the improvement in telephone networking was seen by the struggle of AT&T in 1980s. It gave the new idea with the name "NETWORK CONTROL POINT"(Nunes, Mendonca, Nguyen, Obraczka, & Turletti, 2014) by detaching the control from the data plane. The key idea was to use separate channel for each plane such as signaling which is control plane and voice which is data plane. This technique had more advantages than previous one as it was flexible to add new services with less time required to install. These advantages ultimately eliminate the IN_BOUND SIGNALING technique because of efficiency and have better view of network.

2.3.2 Programmability

Programmability in a network allows software to handle the network behavior, whereas the software is independent of network hardware. In a programmable network, a network engineer can re-configure the infrastructure of network instead of building it from scratch (Feamster, Rexford, & Zegura, 2014).

In 1990s, the programming in the networking was introduced, which is the inspiration of active network projects (Prasad, 2014). Active network can be defined as the networks in which network devices analyze and trace the packets or do custom operations on packets. At present, firewall can be considered as the active network.

The active network idea is introduced by the research community with the name "DARPA" in mid 1990s. It was observed that it is difficult to add new services in the

network and the solution is to change the equipment or change the infrastructure. To make the use of existing network for adding new services, active network gave the concept of innovation such as network devices can be upgraded instead of changing the complete device. In active network, devices only forward the packets and these devices are controlled by programmable controller.

Active network based on two approaches. In the first approach, it is considered that the packet which is moving also contains the program and contains the information about delivery of packet to destination. This packet will be processed by the active mode, for example, programmable switch or router.

In second approach, the process is reversed as all the forwarding rules are already installed on programmable switches and packets are processed on switch and sent to the destination. This approach is closely related to SDN.

2.3.3 Network Virtualization

Network virtualization combines the software and hardware resources of network to one administrative entity which is software-based and called virtual network. Virtualization of network is taking more than one instance of network, in which there are may be more than one network topologies. The most common example of network virtualization is VLANs (Virtual Local Area Network). There are several advantages of network virtualization:

i. Virtualization of network facilitates more than one instances of network by using the one platform.
ii. It provides the sharing properties, such as virtualization is installation of logical system over physical system and both share the same resources.

The example of network virtualization is Mininet which is used for experiments. In Mininet, the virtual network is launched and performed experiments to obtain the results.

2.3.4 Separate Control Plane

There are many advantages of separating the forwarding plane from control plane. Controller in control plane makes the easier control of network. Network administrator can have network view from single central point.

In late 2000, the data packets were controlled by Ethane technique (Casado et al., 2007). This technique allows the applications to control the packets at data plane while applications are installed at control plane. In short, this technique supports the communication between control plane and data plane. The controller is able to control the flow tables of switches and can modify the flow rules according to the requirements. The main disadvantage of this technique was vendor proprietary. This disadvantage motivated the researches to find the solution which should not be proprietary. As a result, it led to the OpenFlow.

The OpenFlow (McKeown et al., 2008) as shown in Figure 2.3, works in a way that flow entries of flow tables in switches are controlled by controller and new flow rules can be added to flow tables without depending on vendor. The controller manages the flow of network traffic and directs the traffic towards destination

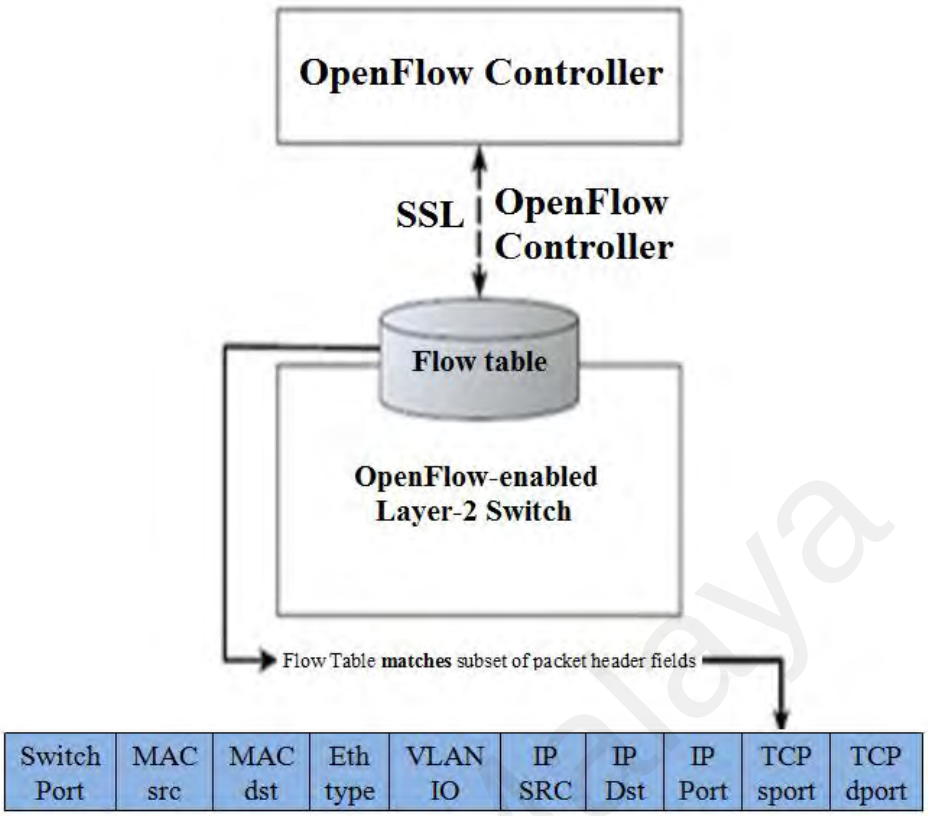


Figure 2.3: Open hardware (Source: (Mousavi, 2014))

2.4 Conventional Networking and SDN

Separation of forwarding plane and control plane has made the network easy to manage. In SDN, the co-ordination among different devices such as load balance and security devices is easier than conventional networking. These devices perform the tasks without interference. Another advantage which cannot be ignored is the control of network from single point. The entire network can be controlled from one controller which manages the hundreds to thousands of devices and traffic flow can also be changed by using single command for all devices. However, in conventional networking, each device is configured separately which makes the things complex for network operator. In SDN, the whole network can be seen from single point and by viewing the simple program; the network behavior can be recognized. Finally, SDN combines the programming with networking which provides the opportunity to implement conventional approaches in networking. In conventional networking, only

some commands are used to manage the network. SDN uses the concept of OpenFlow which lifts the conditions of vendor specification (Feamster et. al, 2014).

2.5 Infrastructure of SDN

SDN induced the programming into networking and detaches the control and data plane. The control plane might be at isolated place and can be programmed according to requirements. The controller could be a laptop or desktop system. It could be installed on virtual machine and network resources can be used without knowing the location.

Controller is considered as brain in SDN because the intelligence of network exists here and it is central point from where the whole network can be viewed.

SDN uses the OpenFlow concept which facilitates the enterprises to control the network from single point and also eliminate the condition of vendor specification. It makes the network simpler, agile and improves the performance. In SDN, the controller is configured programmatically and can change the behavior of network on fly if any event will happen, which makes the network secure. In addition, the services and applications can be deployed on network in time less than conventional network.

2.6 OpenFlow

SDN, at present, is emerged in OpenFlow (McKeown et al., 2008). It is moderately new system created by Stanford at first to give an approach to analysts to run tests convention in the system. The protocol provided by an OpenFlow enables the controller to manage the flow rules in OpenFlow switch.

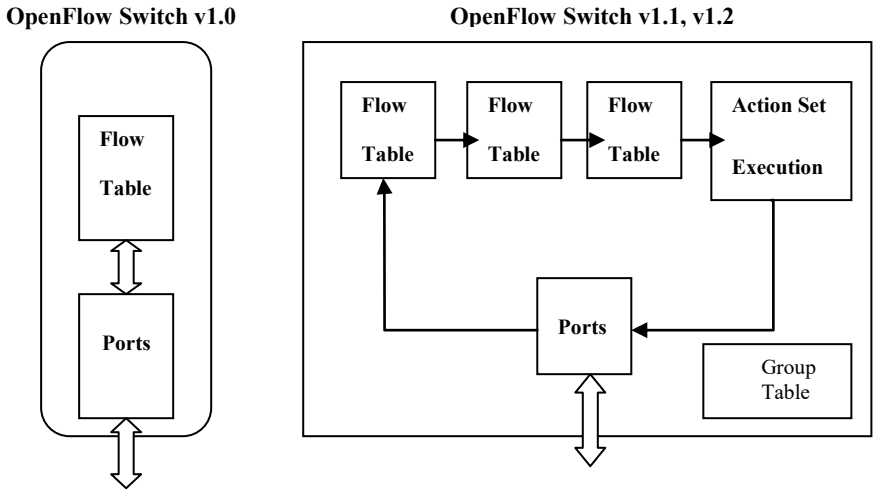


Figure 2.4: Model of OpenFlow Switch (Source: (Mousavi, 2014)

Figure 2.4 illustrate that OpenFlow switch has flow tables according to the version. For instance, OpenFlow with switch version1.1 consists of more than one flow table. However, the OpenFlow switch with version 1.0 has one flow table. The flow table consists of flow entries which match the packets. The flow entries consist of match fields. The match fields are compared with incoming packets and if the packet matches the fields, more than one action will be accomplished. These actions could be forwarding a packet to the destination or dropping packets in such a way that If there is no flow entry, then packet is forwarded to the controller. The flow entries in the flow table contain:

i. Match Field
ii. Instructions
iii. Counter
iv. Timeouts
v. Priority
vi. Cookie

The switch behavior is dependent on the state of flow table. The flow tables of all the switches are maintained by the controller to which these switches are connected. The controller defines the rules for the packets which are not matched to the flow entries in

switch. Figure 2.5 shows the switches based on OpenFlow. The SDN controller and network devices are connected by OpenFlow protocol. The switches which are OpenFlow-enabled are of two types.

i. OpenFlow Only

ii. OpenFlow Hybrid

The first type of switches only supports the operations for OpenFlow and second type of switches supports the operations for OpenFlow and for the Ethernet switching also known as layer-2 switching. In short, OpenFlow is channel used to connect the controller and OpenFlow switches. It also supports different APIs to change the network behavior on fly.

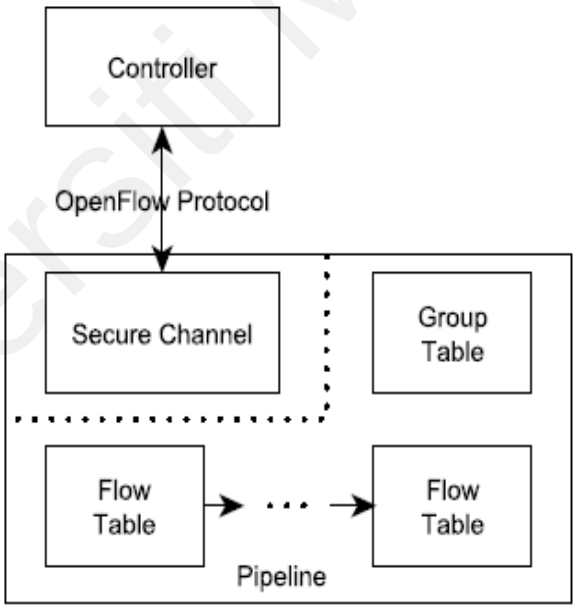


Figure 2.5: Components of switches based on OpenFlow (Taha et. al, 2014)

2.6.1 OpenFlow Ports

Ports, for connecting the switches with each other, are interfaces of network for transferring the packets in the network. Plenty of OpenFlow ports are available by

OpenFlow switch for processing. The ports those are supported by OpenFlow switch are of three types:

i. Physical Ports

The ports which are hardware interface and are defined by switch are known as physical ports. At the top of switch hardware, the OpenFlow switch is virtualized for some cases, which represents the virtual ports of corresponding physical hardware.

ii. Logical Ports

Logical ports do not have hardware interfaces and are defined by non-OpenFlow procedures in switches. The physical ports can be differentiates from logical ports by justifying that packets for logical ports have information about tunnel ID and physical port for reporting to controller. For physical ports, the packets do not have Tunnel ID information.

iii. Reserved ports

These ports are reserved for specific actions, e.g. flooding or ports reserved to send specific packets to the controller. Only the ports with the tag “required” are supported by the switch. Reserved ports with the name are stated below.

a) Required (ALL)

The required all indicate that for forwarding specific packet, switch have authority to use all ports. Switch uses all the output ports and one packet is broadcast to all ports except the ports which are configured not to forward the packets.

b) Required (Controller)

These ports are representation of OpenFlow controller with control channel. An entrance port and output port, both may be used for this type. The output port is used when there is new packet encapsulated by packet-IN message and forwarded to the controller via OpenFlow protocol. The use of entrance port represents that the source of packet is controller.

c) Required (IN Port)

This is the representation of Ingress port and it is used as output port only. Packets are sent out using Ingress port.

d) Required (ANY)

These ports are neither used as output nor as Ingress Port and when none of the port is specified the OpenFlow commands are used by special value.

2.6.2 OpenFlow Tables

The OpenFlow switch consists of one or more flow tables but at least one flow table is necessary for each OpenFlow switch. In OpenFlow switch, there is concept of OpenFlow pipeline, consists of more than one flow tables. Each flow table consists of flow entries. The matching of packets with appropriate flow entries in the flow table is defined by pipeline processing. If there is flow entries in flow table for incoming packets, the instruction of that specific flow entry is performed. There is possibility of forwarding the packet towards a new flow table depending upon the instruction which will be performed. On the contrary, if the packet is not matching with the flow entry in flow table, then table miss will occur. At the occurrence of table miss, one of the following possibilities will be considered for processing the packet:

i. Drop the Packet
ii. Forwarding the packet towards new flow table
iii. Forwarding it to controller

There are two methods to remove the flow entries. The first one is the use of controller and the second is defining the expiry time for flow entries.

2.6.3 Pipeline processing

In an OpenFlow switch, as shown in Figure 2.6, there is an OpenFlow pipeline which consists of more than one flow tables and one flow table consists of more than one flow entries. The processing of OpenFlow pipeline describes how flow tables interact with packets. There must be one flow table for OpenFlow switch. However, more than one flow table is optional. The pipeline processing of OpenFlow switch is simple for one flow table and it is most desirable.

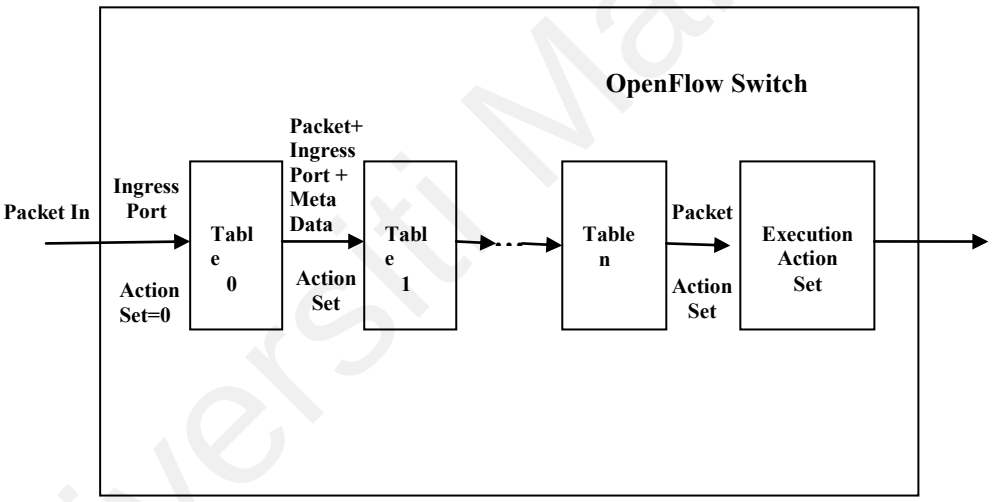


Figure 2.6: Switch pipeline processing (Taha et. al, 2014)

In OpenFlow switch, flow tables are numbered in ascending order and always start from “0” flow table and moves sequentially towards “n” flow table. The remaining flow tables are dependent on first flow table results, for example, if flow entry of first flow table executes the instruction to pass the packet to the second flow table then packet will move towards second flow table.

2.6.3.1 Flow Table

There are many flow entries in one flow table as shown in Table 2.1. The one flow entry is distinguished from another by combining the match field with priority. Flow entry which priority is zero and match field is omitted is known is table miss.

Match Field	Priority	Counters	Instructions	Timeout	Cookie

Table 2.1: Flow entry components (Taha et. al, 2014)

2.6.3.2 Group Table

By combining group entries, a group table is formed as shown in Table 2.2. Group entry components are as followed:

i. Group Identifier

Group identifier identifies the group entry and it is 32 bit long integer.

ii. Group Type

It determines the group semantics.

iii. Counter

Counter count the number of processed packets and is updated after processing.

iv. Action Buckets

These are the buckets, which consist of instructions and these instructions are executed corresponding to defined parameters.

Group Identifier	Group Type	Counters	Actions Buckets

Table 2.2: Group entry components in group table (Taha et. al, 2014)

2.6.3.3 Matching

Figure 2.7 illustrates the packet processing of OpenFlow Switch. When the packet arrived at OpenFlow switch, the table lookup process is performed by OpenFlow switch and it looks for the flow entry in first flow table. However by following the pipeline processing, it may look-up from other flow tables. Different dependencies are in table lookup for match field, for example, packet types and header fields of packet. The header fields include source address or destination address of packets. The two other ways to match the fields are metadata and entrance port. However, metadata is useful for exchanging information among tables of a single switch.

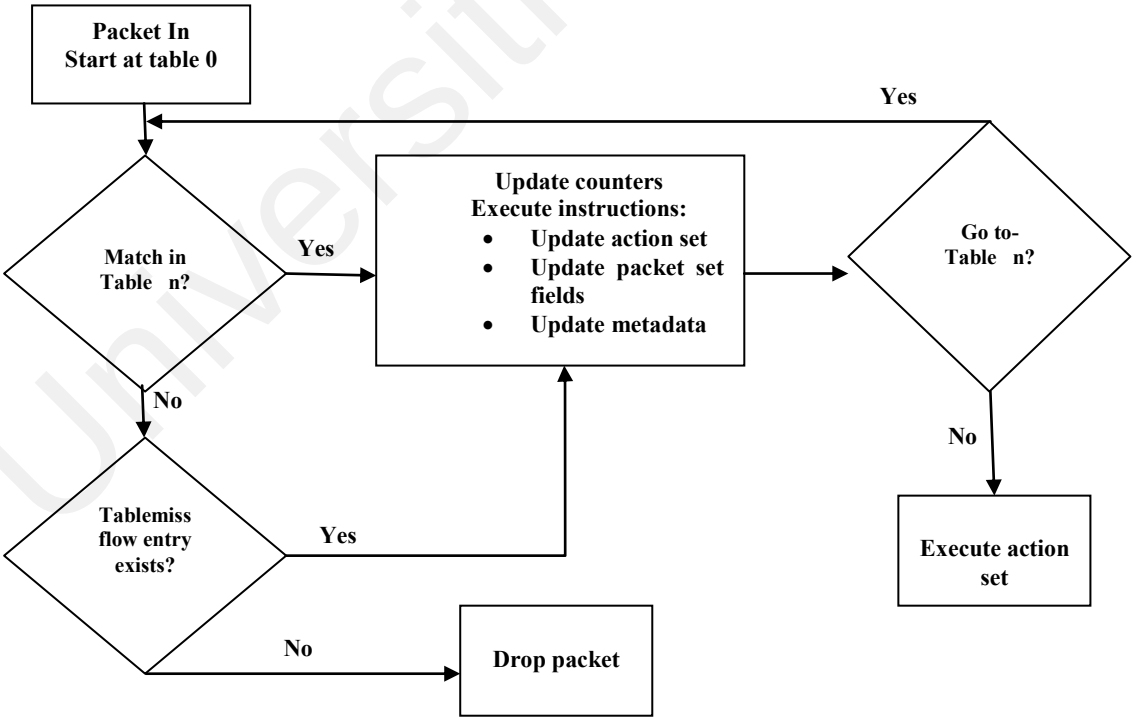


Figure 2.7: Packet processing of OpenFlow switch (Mousavi, 2014)

2.6.4 OpenFlow Channel

In SDN, the southbound interface also known as OpenFlow channel provides the connectivity between switch and controller. All the communication between switch and controller takes place via this interface. OpenFlow channel utilizes the OpenFlow protocol. To make the communication secure, the OpenFlow channel encryption is done by TLS. However, TCP protocol is used when security is not essential.

2.6.4.1 OpenFlow Protocol

There are three types of messages, OpenFlow protocol can support.

i. Controller to switch

These are the messages, which controller starts for managing and controlling the network devices.

ii. Asynchronous Messages

These are the type of messages which switch starts for updating controller about the events, occured in the network.

iii. Symmetric Messages

The controller or switch can start these types of messages without solicitation.

2.6.4.2 Message Handling

Delivery of messages and packet processing is reliable via OpenFlow protocol. However, it does not acknowledge the processing which can be understood by following factors.

i. Message Delivery

The delivery of message across OpenFlow protocol is guaranteed until the OpenFlow protocol is working properly. In case of failure, the controller will be totally disconnected from switch.

ii. Message processing

It is desired that all messages sent by controller towards switch should be processed and in return acknowledgement should be sent back to the controller. In case of any failure of message processing, error message should be sent to controller. The generation of error message does not happen in Packet-Out message and after processing of message at OpenFlow, there is no guarantee that whether the packet is dropped or received by the switch. Several factors can lead to the dropping of packets such as congestion, quality of Service (QoS) or invalid port.

iii. Message Ordering

Barrier messages are used to control the message ordering. Whereas barrier messages are the messages, separate the two dependent messages sent by controller. To enhance the performance, messages can be reordered by switches when the barrier messages are not present. In this case, controller is independent of any processing order.

2.6.4.3 Connections of OpenFlow channel

As it is mentioned in the introduction of SDN in this chapter that message exchange between controller and switches takes place via OpenFlow channel. One OpenFlow controller can be connected to more than one OpenFlow channels and via these channels, can be connected multiple OpenFlow switches. Similarly, OpenFlow switch can be connected to one controller via OpenFlow channel and can also be connected to more than one controller to avoid the network collapse if one controller is failed for

connectivity, TCP/IP protocol may be good enough. The connection is always started by OpenFlow switch towards controller.

i. Connection Setup

Connection setup between controller and switch takes place via OpenFlow channel and always started by switch. The establish connection may be TCP or TLS and switch uses controller IP address to establish this connection. The OpenFlow switch should be smart enough to identify the incoming packets according to the flow entries.

ii. Connection Interruption

There are two modes which may be active when the connection between the switch and controller is lost:

a) Fail Secure Mode

b) Fail Stand Alone Mode

The connection may be lost because of timeout of TLS session or time out of repeat request. In case of fail secure mode, the working of switch is normal, such as, all the flow entries are expired at defined timeout.

In fail stand-alone mode, reserved ports are used to process the packets. In this mode, after losing connection with controller, all flow entries are present and after establishing connection, it depends on controller to keep or delete the flow entries.

iii. Encryption

It is used to make the connection secure. TLS connection takes place between controller and switch for securing the communication; it uses “6633” TCP port. Mutual authentication takes place between controller and switch using private key for signing a certificate. However, communication between controller and switch may takes place via

plain TCP. Security measures other than TLS are desirable while using TCP in order to prevent the communication from different network attacks.

2.7 Related Work

The related work is classified into planes which explain the network attacks corresponding to each plane.

2.7.1 Application plane

Application plane in SDN defines the rules on controller via north-bound interface. Several studies for securing application layer have been carried out. The possible attacks that can affect the application plane are unauthorized access, DoS attack, malicious application and configuration issues.

In (Anwer, Benson, Feamster, Levin, & Rexford, 2013), central controller is proposed. The architecture is named as slick. Middle-boxes are used on which controller is considered responsible of fix and transfer functions. Controllers are dependent on applications which tell the controller about necessary functions to install for routing. Security parameters are defined while functions are installed on middle-boxes.

In FlowTag architecture (Fayazbakhsh, Sekar, Yu, & Mogul, 2013), communication between controller and middle-boxes takes place via application programming interface. Packet headers in FlowTag architecture contains traffic flow information for controlled routing of packets which are tagged. However, this architecture is not suitable for dynamic actions and it supports only predefined policies which is considered as disadvantage and not suitable for current technologies.

SIMPLE policy enforcement layer (Qazi et al., 2013) requires no modification in SDN architecture which makes this technique different from (Anwer et al., 2013),

(Fayazbakhsh et al., 2013). From the application layer, traffic can be directed to the middle-box on which rules are defined in order to avoid the possible attacks.

2.7.2 Application-Control Interface

Application-control interface also known as north bound interface is source of communication between application plane and control plane in SDN. In (Ballard, Rae, & Akella, 2008), OpenSAFE technique is proposed. In this proposed technique, traffic routing through network monitoring devices is managed by ALARM policy language. In order to create the copy of network traffic, span ports are used at various interesting points. Administrative boundaries are considered as desired interesting points. Another technique cloudWatcheris proposed by (Shin & Gu, 2012) which also ensures that network packets are passing through security devices.

Programmability makes SDN more vulnerable for threats. It is common to apply security models on different applications which are installed on different devices in order to prevent from fraudulent rule insertion. (Canini, Venzano, Perešíni, Kostić, & Rexford, 2012) describes about to check the OpenFlow applications correctness with the aid of suitable models. Switch contains flow tables and flow tables consist of flow entries. It might be possible that intra-switch miss-configuration could be occurred which results wrong packet forwarding. To overcome this problem, Binary decision diagrams are proposed (Al-Shaer & Al-Haj, 2010).

To verify flow policies, Flover is proposed by (Son, Shin, Yegneswaran, Porras, & Gu, 2013). This proposed technique consists of assertions set and modulo theories. The FRESCO technique is proposed by (Shin et al., 2013), which explains about OpenFlow security application development framework. Network threats detection and reduction takes place by reusable module library. FRESCO includes FortNOX (Porras et al., 2012). FortNOx is considered as enforcement engine and it verify the flow rules. If new

rule is defined and if rule conflict occurs between the existing rule and the new rule, then new rule is accepted or rejected depending upon the defined security rules.

2.7.3 Control plane

Control plane is considered as the main plane of SDN and responsibilities of control plane are network monitoring, routing decisions and programming of physical network to define the routing path.

One of the solutions to the control plane attacks is SANE architecture (Casado et al., 2006). The proposed technique is more focused towards host authentication and policy enforcement which are the responsibilities of controller. ETHANE (Casado et al., 2007) advanced version of SANE which uses bit different concept to control the network. Centralized controller and ETHANE switches are used in this approach. Centralized controller enforces the globally defined flow rules or policies whereas switches are only the fibber which forward data packets according to global policies or flow rules defined in routing tables. The advantage of this architecture over SANE is that it requires less changes in original network. The disadvantage is the network policy might be compromised by application traffic.

The technique with the name Resonance (Nayak, Reimers, Feamster, & Clark, 2009) is proposed in which the network devices enforce the dynamic access control. The control is based on defined security rules.

In various types of traffic, flows are observed in order to detect the distributed denial of service attack (DDoS) (Braga, Mota, & Passito, 2010). In this technique, controller is used to monitor forwarding plane and detect the traffic which is malicious. To identify abnormal traffic, Self-Organizing Maps are used.

Several techniques can be used by attackers to find compromised nodes in order to attack or to get information of network. The technique with the name MTD (Jafarian, Al-Shaer, & Duan, 2012) gives the solution to prevent the IP addresses. Virtual IP addresses are assigned to hosts in order to hide real IP addresses. IP addresses are assigned randomly to hosts and virtual IP addresses pool is managed by controller. The advantage of this technique is that attacker cannot obtain the real IP address of host using different scanning techniques.

2.7.4 Control-Data Interface

Control-Data interface also known as southbound interface provides the communication channel between control plane and data plane. Considerable amount of studies exist for southbound interface security. (Kloti, Kotronis, & Smith, 2013) proposed STRIDE threat analysis technique to analyze the OpenFlow protocol in order to prevent from DoS attack and information leakage.

For mutual validation between controller and switches, transport layer security (TLS) is explained by OpenFlow switch specification (Mizrahi & Moses, 2013). As TLS is not commonly used and DoS attacks can be effective at control-data interface, so vulnerability assessment is discussed by (Benton, Camp, & Small, 2013).

(Kreutz, Ramos, & Verissimo, 2013) proposed various techniques with the name replication, diversity and secure components to prevent the network from various threats. These threats are usually because of the centralized controller as the whole network is dependent on controller which makes it more desirable for attackers.

In (Skowyra, Lapets, Bestavros, & Kfoury, 2013), the Verificare technique is discussed in order to make the OpenFlow network more scalable and secure. It provides model which can be used to check the network correctness. (Handigol, Heller,

Jeyakumar, Maziéres, & McKeown, 2012) discusses prototype network debugger.. This prototype is developers friendly and SDN developers can easily find the root cause by recreating the events those are cause of errors.

2.7.5 Data plane

Data plane also known as forwarding plane forward packets according to defined rules which are defined by controller. Switches on data plane contain flow tables with flow entries for forwarding packets. Various threat vectors can affect data plane to insert fraudulent flow entries. Monitoring system IDS is proposed by (Skowyra et al., 2013). For detection and response to network attacks in embedded mobile devices, this proposed technique uses the architecture of SDN.

(Goodney, Narayan, Bhandwalkar, & Cho, 2010) describes that Network IDS is on hardware based which can be configured by network administrator and used for deep packet inspection. The above two techniques detect abnormal traffic. (Mehdi, Khalid, & Khayam, 2011) is about the implementation of IDS using SDN in small networks like home or small office network.

Another technique proposed by (Naous, Stutsman, Mazières, McKeown, & Zeldovich, 2009) with the protocol name ident ++. This technique is proposed in order to avoid controller which is considered as bottleneck and to make device capable of forwarding packets. The proposed protocol demands end hosts additional information to verify that host is legal or fake.

2.8 Studies related to DoS attack on controller

To protect the controller from DoS attack, a lot of researches have been carried out. To detect the presence of DoS attack, technique with the name statistical method is proposed by (Nugraha, Paramita, Musa, Choi, & Cho, 2014), which uses the

characteristics of packets. To monitor and detect the attack, sFlow is used. However, the limitation of this attack is that it does not consider the time characteristics and it counts the malicious packets only. Another technique is proposed by (D. K. Lee, G. Y. Bang, & Choi, 2014), this technique is monitoring the DDoS attack using centralized monitoring system and detecting the DDoS by the use of snort. However, in this technique, the attack target is not controller and also the time factor is not included.

The technique with the name lightweight DDoS attack detection (Braga et al., 2010) is proposed. As the name of this technique represents that it uses less resources to detect DDoS attack compared to the other method. This method exploits the SDN programmability feature to manage the flow of traffic in switch. But it was not explained that how SDN controller will be effected by DDoS attack.

(Mousavi & St-Hilaire, 2015) proposed a technique for DDoS detection. This technique monitors the change in entropy in a way that if the destination IP addresses of packets for specific host will be increased, the value of entropy will be decreased. Threshold value is defined and if the entropy value is crossing the value of threshold, then DDoS attack is detected. But the limitation of this technique is not considering the time.

Another technique with the name Time-Based DDoS detection (Gde Dharma, Muthohar, Prayuda, Priagung, & Choi, 2015) is proposed. This technique proposes the detection of DDoS attack by monitoring the destination IPs for specifics hosts and the required time for which the traffic rate will be high. However this technique is not considering the mitigation of DDoS attack on controller in SDN environment.

2.9 Research Gap

The techniques closely related to our proposed technique are (Gde Dharma et al., 2015), which gave the idea of time-based DDoS attack detection based on entropy change and (Giotis et. al, 2014) which combines the OpenFlow and sFlow for mitigation of DDoS attack using whitelist table. However, In this proposed technique, the detection and mitigation of DoS attack is done using sFlow-rt and DoS policy respectively but working mechanism is different to benchmarked techniques. The DoS policy is more efficient in mitigating the DoS attack than whitelist table technique as DoS policy is simply installed on controller using pyresonance plateform and mitgate the attack in real time.

2.10 Summary

In summary, this chapter discuss the architecture and importance of SDN. It also explains the threat vectors affecting the SDN architecture. The existing SDN controller security related techniques are reviewed and limitations and significant issues are highlighted.

The key issue focussed is to detect the network attacks on controller in SDN environment which includes DoS attack and to mitigate the DoS attack. Eventually, it identifies the challenges involved to detect and mitigate the DoS attack in more convenient way which includes the real time detection and mitigation.

To meet challenges, this chapter give emphasis to the model which is more convenient for detection and mitigation of DoS attack. On the basis of review and critical investigation of current detection and mitigation techniques, recommendation is made that is starting point for real time detection and mitigation of DoS attack. The summary can be explained with the help of Figure 2.8.

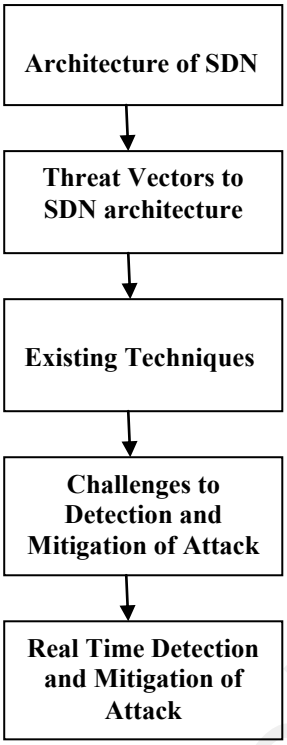


Figure 2.8: Summary

CHAPTER 3: FRAMEWORK FOR DOS ATTACK DETECTION AND MITIGATION

This chapter includes the technique which is proposed for DoS attack detection and mitigation on controller in SDN environment. First, it gives the scenario for DoS attack and the methodology. In the methodology, it explains how sFlow-rt is connected with controller to detect the DoS attack. Then it discusses the proposed controller and explains how DoS policy is implemented on controller for DoS attack mitigation. Finally, it gives the concluding remarks.

The chapter is categorized into four sections. Section 3.1 presents the DoS attack scenario. Section 3.2 explains the methodology which is proposed to detect and mitigate the attack. Section 3.3 explains how DoS policy works to protect the controller. Finally, section 3.4 gives the conclusive remarks.

3.1 DoS Attack Scenario

Figure 3.1 depicts the DoS attack scenario. The topology which we used consists of one open vSwitch, directly connected to the controller, and three simple switches which are considered as fiber only used for forwarding the packets. To launch the topology, MiniEdit in Mininet is used. The communication between switches and controller takes place through open vSwitch and remaining switches are connected to hosts in their own network. The communication process is step vise as first packets move towards sFlow collector and after completion of traffic analysis, these packets move towards controller. The entire process takes place only for new incoming packets which do not have flow entries in switch flow tables. On the contrary, some hosts in network may be compromised and can cause the DoS attack by generating packets that are malicious. Since these packets do not have any flow entry in switch flow table, these packets are forwarded to the controller for new flow entries. As a result, the controller may be

exhausted because of processing the same packets again and again and the controller may no longer be functional and may be unreachable to the remaining hosts.

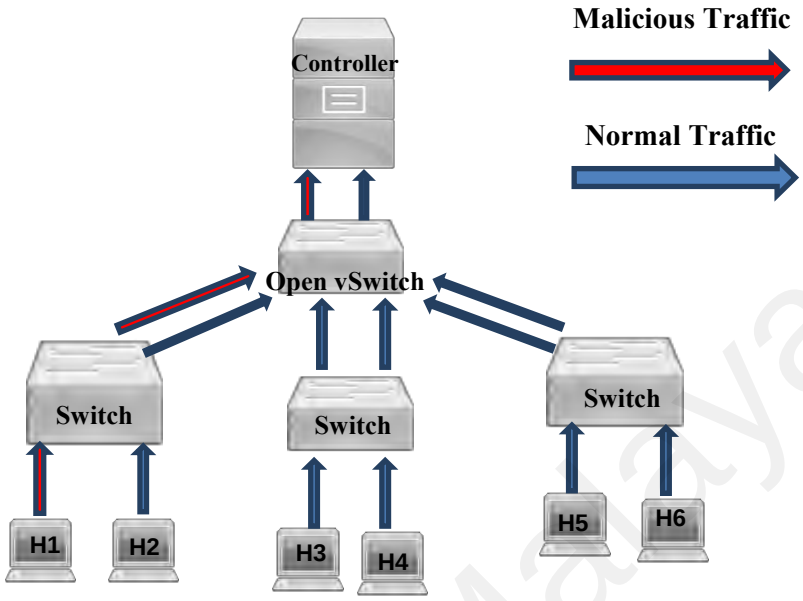


Figure 3.1: DoS Attack Scenario

3.2 Methodology

This section includes the methods to carry out the research. The Figure 3.2 depicts the flow of research. The research consists of five phases. It includes information gathering and then analysis of the information, proposed method, designing and implementation of system and evaluation.

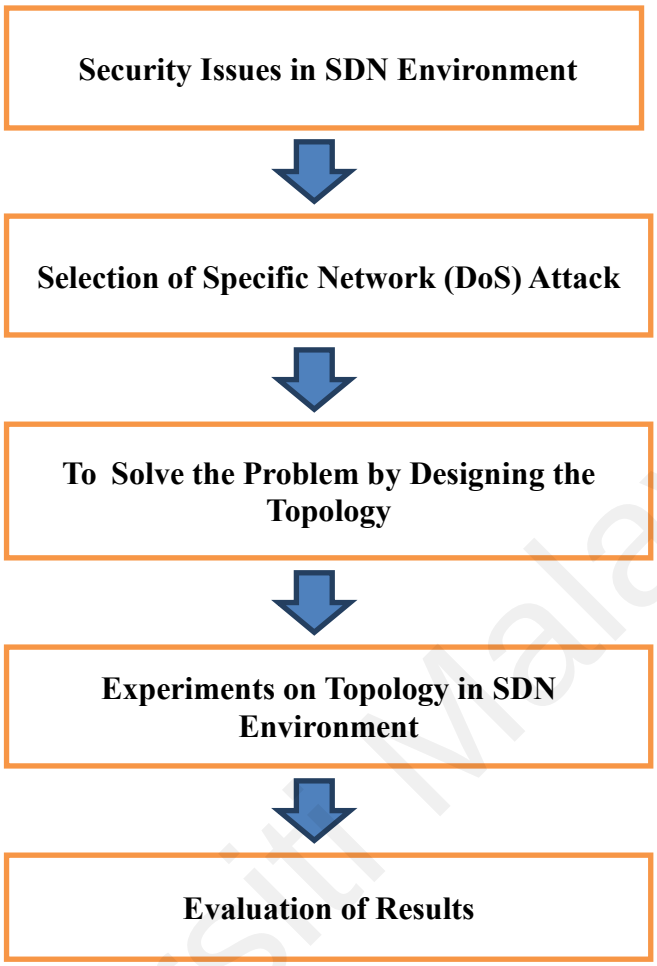


Figure 3.2: Methodology

In the methodology, the main objective is to protect the controller from DoS attack. The proposed technique consists sFlow-rt, Pyresonance and DoS policy and mitigates the attack in real time. The detection and mitigation process can be categorized into two steps:

i. Monitoring and detecting the network traffic using sFlow-rt.

ii. Activation of DoS policy after detection to mitigate the attack.

In first step, sFlow-rt is mentioned. All the network traffic is monitored by sFlow collector. In result, it differentiates the normal and abnormal traffic according to the defined rule.

In second step, real time mitigation is mentioned. This work is based on real time mitigation by blocking the malicious packets. To generate malicious packets for DoS attack on controller, ICMP packets are sent from compromised node. There exists no flow entry in flow table for these packets. These packets are forwarded to the controller. Since there is no rule defined against these packets at controller and also packets are moving towards same destination, the sFlow indicates the DoS attack as abnormal traffic rate is for higher time than the defined threshold time. At the indication of DoS attack, DoS policy is triggered which is installed by using Pyresonance platform. The DoS policy blocks the traffic coming from malicious node. Figure 3.3 illustrates the proposed architecture.

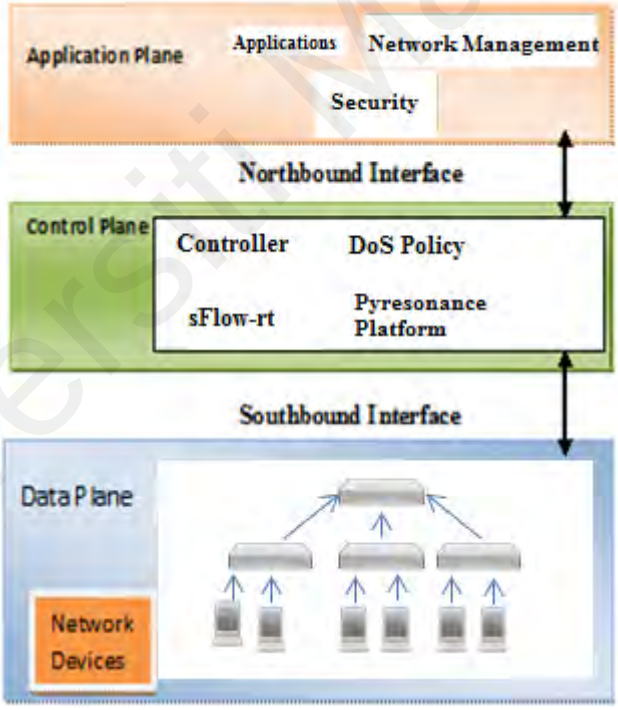


Figure 3.3: DoS policy architecture

3.2.1 sFlow-rt Collector

sFlow, stands for sampled flow, if added to the network results more reliable and enhanced performance of network. By the addition of sFlow in network, the traffic from all interfaces, on which sFlow is enabled, can be observed. This allows the network to work normal and protects the network from abnormal traffic.

There are several sFlow applications which make the network more reliable. These applications can be categorized as follow:

i. Detection of problem, identification and fixing of problem in a network.
ii. Handle the congestion in real time.
iii. Monitoring and observing the abnormal activities in the network and then Figure out the root cause of this abnormal activity.

Different features of sFlow can be categorized as follow:

i. Wide View of Network

sFlow can be reconfigured and its capacity of monitoring interfaces can be increased or decreased. As a result, the sFlow gives the complete network view from single point.

ii. Cheapest solution

It is considered as the cheapest solution in terms of system resources. For example, it does not require extra system resources such as memory or CPU during operation.

In SDN, sFlow can be implemented on switches or controller. In this work, it is installed on controller. Traffic monitoring mechanism of sFlow can be understood by Figure 3.4.

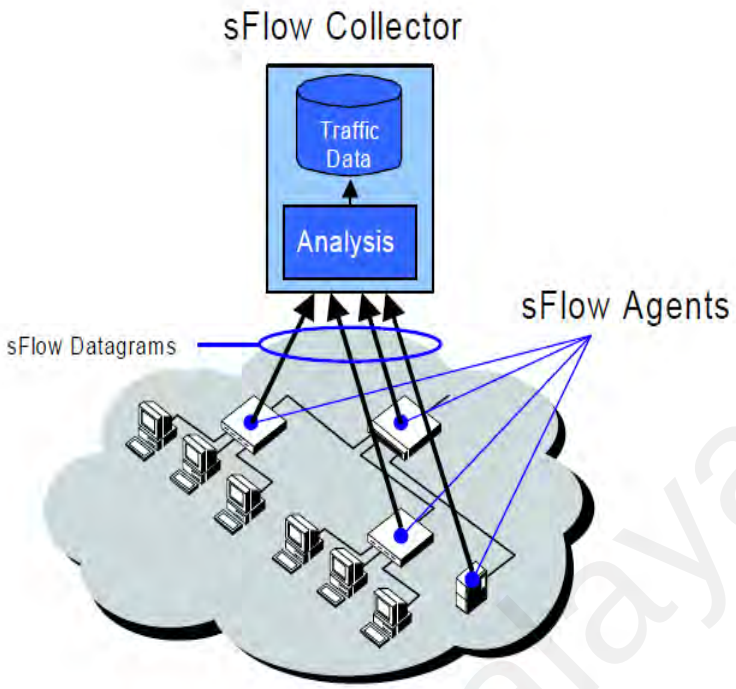


Figure 3.4: Traffic monitoring mechanism (Source: Online)

The Figure 3.5 depicts the elements of sFlow. All network traffic first goes to the sFlow collector and then to the controller.

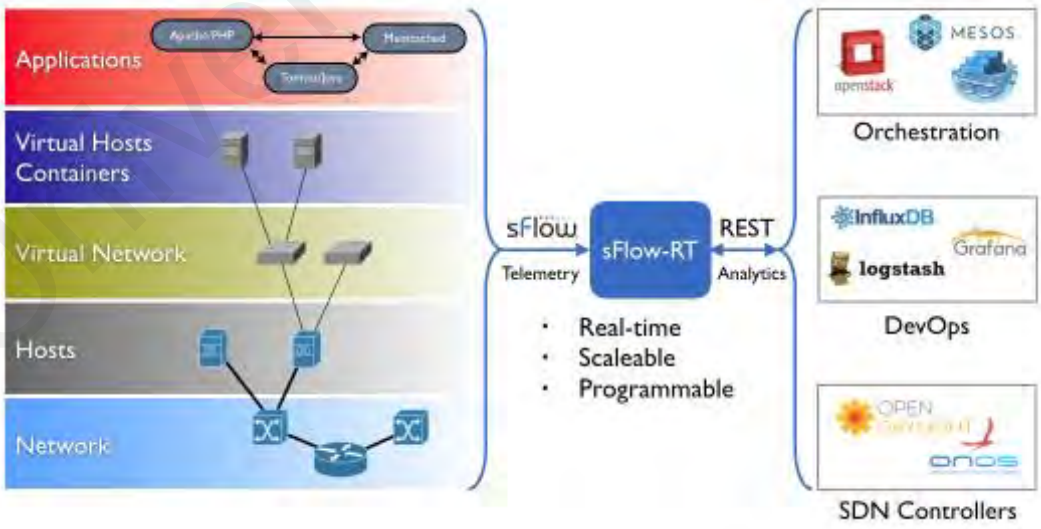


Figure 3.5: sFlow Components and working (Source: Online)

3.3 Controller

In SDN, the control plane and data plane are detached where data plane also known as forwarding plane consists of switches. The separation of control plane and data plane gives the orchestration concept by providing the view of entire network from single point. This single point controller is the main component of SDN architecture. It is also considered as the brain of a network. Any configuration change on controller may change in the behavior of entire network. Figure 3.6 shows simplest architecture of traditional network and Figure 3.7 depicts the simplest architecture of SDN.

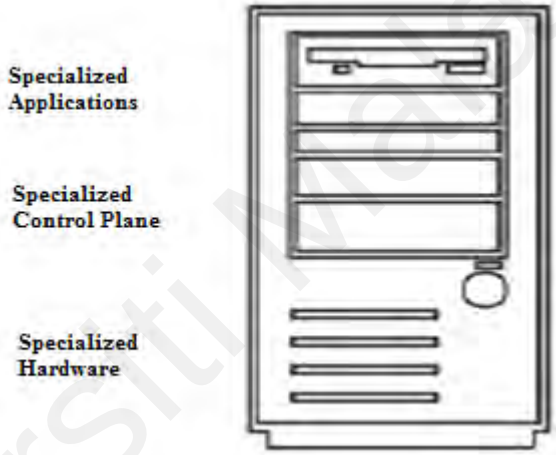


Figure 3.6: Traditional network architecture (Taha et. al, 2014)

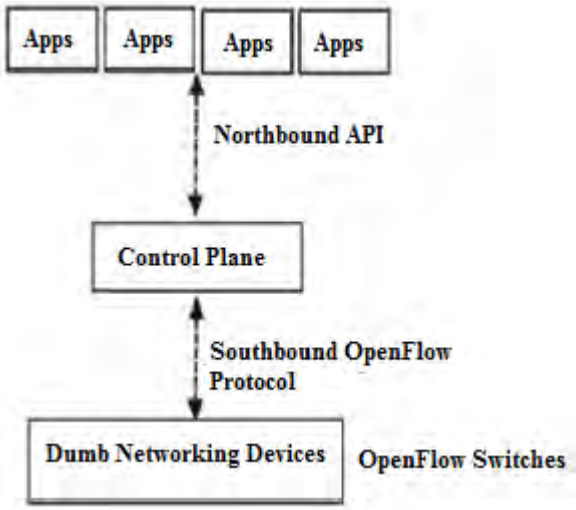


Figure 3.7: SDN architecture (Taha et. al, 2014)

In this research, POX controller is selected for application development in SDN architecture. It uses python language and several applications written in python can be run by POX controller. The communication among the plane (control plane and data plane) takes place via OpenFlow protocol. The Figure 3.8 illustrates that how communication takes place between controller and switches.

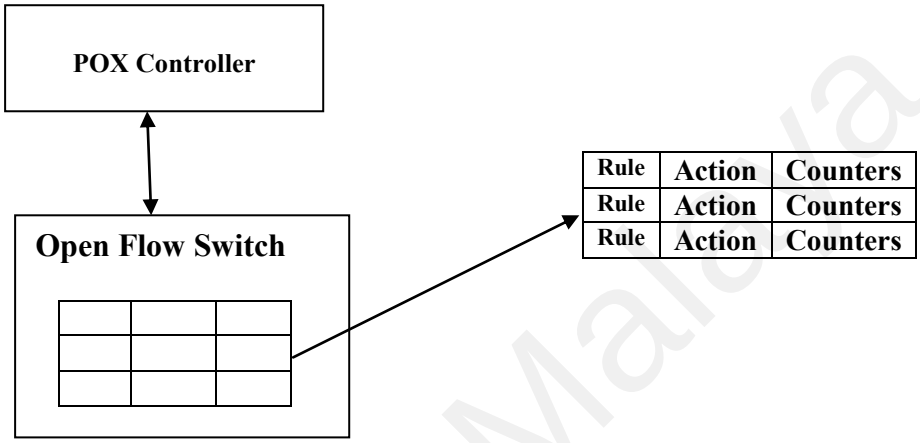


Figure 3.8: POX controller

First, the connection between controller and switch is established by turning on the switch. At the start, there is no flow entry in the flow table of switch. The packets first go to the control plane. The controller at the control plane set flow rules on switch for packets. All the flow tables in switch contain flow entries and flow entries consists of following three parameters:

i. Rule which is defined for packet
ii. Actions according to the defined rule.
iii. Counter, counts the number of packets.

Once the rule is defined in flow table for new packet, second time the packet is forwarded to the destination directly from the switch.

3.3.1 Pyretic

In programming languages of SDN, Pyretic is considered as the family member of frenetic. Pyretic is the combination of python and frentic. By the use of pyretic, a brief network application can be written by programmers. Pyretic is embedded in the python language and is programmer friendly. It also implements the program on switches.

Pyretic is installed on controller. It is installed by using "$ pyretic.py –m PQ Pyretic.modules.mac learner". Where "–m" represents the mode of operation of pyretic on runtime and "p" represents proactive scheme, as rules for incoming packets are already exists in switch flow table.

Pyresonance is one of the applications of SDN. Whenever a network event occurred, Pyresonance changes the configuration of network by the aid of state machine. If the word Pyresonance is split into two words, it will become Pyretic plus resonance. In other words, Pyresonance is the combination of Pyretic and Resonance.

SDN can be managed by a framework called Resonance. By the use of Resonance, the policy response to any network event can be defined by network administrator. These policies are known as FSM (Finite State Machine). The FSM have finite number of states. For example in proposed work, DoS policy is used which has two states. At the first state, the machine is working normal as there is no DoS attack. The second state is blocking the traffic coming from compromised host which is the cause of DoS attack. There are many other network events for which transitions of state occurs such as host authentication and intrusion detection system. In conclusion, by the use of Resonance, it is easy for the network operators to define the policies relative to different events of network.

Pyresonance is started by "$ pyretic.py pyretic.pyresonance.main –config=./pyretic/pyresonance/global.config – mode=manual"

3.3.2 DoS Policy

The main goal at this research is to protect the SDN controller from DoS attack. To achieve this goal, the DoS policy is used. The DoS policy is implemented on the controller by using Pyresonance platform. DoS policy is written in python language and works in a way that if network traffic is normal, which the sFlow-rt is monitoring continuously then the communication between hosts takes place. On the other hand, if there is any compromised host in the network and sending the malicious traffic, then DoS policy triggered and blocks the traffic coming from compromised host. Within few seconds it will detach the malicious host and after that, malicious host is not able to communicate with other hosts.

3.4 Concluding Remarks

This chapter explains how DoS attack targets the controller and then how this attack is identified with the help of sFlow-rt. After detecting attack, how DoS attack is mitigated by DoS policy which blocks the abnormal traffic and how Pyresonance works.

According to the best of findings, it is considered that this solution is efficient and uses fewer resources to secure the controller. The major advantages of the proposed technique which cannot be ignored are as follow:

i. Real Time Solution

This solution is real-time, as it mitigates the DoS attack within few seconds and also assures that network is functioning normal during mitigation.

ii. Flexible and Agile

By the use of Pyresonance, more than one policy can be applied and after detecting malicious host the remaining hosts in the network remains functional.

iii. Uses of Resources

The resources used in the proposed solution are limited. In general, controller is not hardware equipment. Infect it is software which is installed on system. A system could be any laptop or desktop. It is vendor flexible and controller can be designed according to the requirements.

The next chapter explains the simulation of proposed technique to detect and mitigate DoS attack.

CHAPTER 4: EXPERIMENTS AND RESULTS

This chapter describes the steps which are followed for conducting experiments and the evaluation of results of proposed real-time DoS attack detection and mitigation. First, it discusses network emulator which is used for experiments. Then, it explains the step followed to setup the experiment environments. Finally, it gives the evaluation.

The chapter includes four sections. Section 4.1 reports the significance of network emulator. Section 4.2 explains environment setup for experiments. Section 4.3 presents the implementation of steps to get the results from experiments. Finally, section 4.4 evaluates the results.

4.1 Network Emulator

In this research work, Mininet is used as emulator. Mininet is an environment where a realistic network topologies (virtual) can be easily created. These topologies are useful for development, teaching and research. Topologies may be complex or simple varies case to case. The use of Mininet emulator is same like working on the real network. By the use of Mininet, different topologies can be made and then network traffic can be tested on these topologies to obtain the results. After getting the desired results, these topologies and all steps of methodology can be applied to real network. Mininet can be installed on desktop or laptop having Linux and by taking the advantage of Linux platform, complex topologies with several nodes and with bandwidth in gigabits can be simulated. There are several other tools for network emulation but Mininet is considered as more efficient for SDN.

4.1.1 The advantages of Mininet

The use of Mininet emulator tool is considered advantageous in many aspects. The advantages are as follow:

i. In Mininet, it is really easy to make the network topology, topology can be made by using MiniEdit tool or users can write their own code using different programming languages. It is also efficient in terms of speed.

ii. The use of Mininet makes it simple for making custom network topologies.

iii. As Mininet is very powerful emulator and working on it is very similar to work on real network. As an example, the topology in the Mininet can have several hosts and the capability of these hosts is same like as real hosts on network. These hosts may run all the programs which the original hosts can run on Linux.

iv. Software Defined Networking is example of open source network and programming functionality makes the network more flexible and reliable. OpenFlow switches can be configured in Mininet, and after configuration, it can be connected to the controller. A controller could be virtual or real.

v. By using Mininet, it is not difficult to extend the network by adding more components which may participate in making the network more efficient such as addition of security devices.

4.1.2 Comparison of Mininet with Alternative Approaches

The Mininet advantages can be analyzed by comparing it with alternative designs. Different alternative designs and their comparisons with Mininet are explained as follow:

i. One topology forever

For experimentation, to use real switches and routers is not an efficient way as all the switches and routers need to be configure and new topology is required for different experiments. Mininet is the solution to the mentioned problem. Using Mininet, it is easy to make more than one topology and can be saved for future use which eliminates the need of reconfiguration.

ii. Virtualization

In this alternative to Mininet, it is considered that several instances or virtual machines may be running on a single system or laptop. The main drawback of this alternative is that it demands large system resources in term of memory and processing power. After fulfilling the resource requirements, again it is able to run limited number of virtual machines which is not reliable way. On the other hand, in Mininet, there is concept of virtualization and it is based on operating system. As a result, several hosts can run on the same virtual machine which demands fewer resources. This factor ultimately reflects the efficiency of Mininet.

4.1.3 Mininet Working

Figure 4.1 depicts the working of Mininet. It explains that '**mn**' is used to start the process of Mininet. There are several sub processes in the Mininet process and each sub process is representing the host of a network. In details, if there is need of more than one host in a network, then the Mininet starter starts two sub processes and it results two hosts. The metric (the host name and the Ethernet port) of each host is different. As the Ethernet port is different for each host, so the connection between the OpenFlow switch and the host takes place respectively to the assigned ports and the communication takes place without any confusion. After establishment of connection among the hosts and switches, the connection between the controller and switches takes place via OpenFlow protocol. Controller pushes the flow entries to the flow table of switch corresponding. This complete process takes place on the single virtual machine by the use of Mininet.

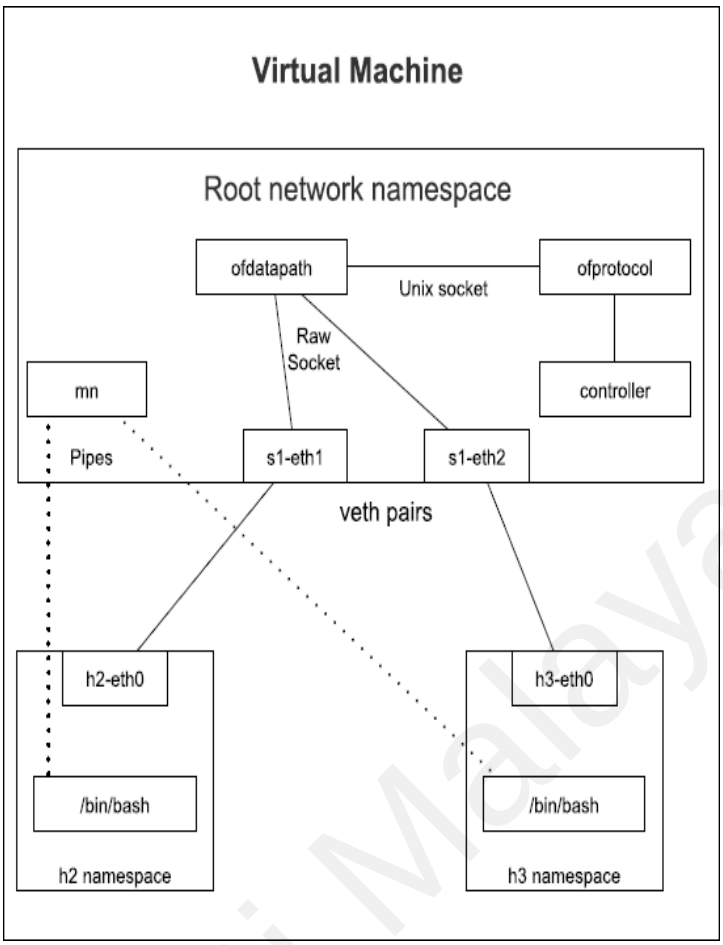


Figure 4.1: Mininet working

4.1.4 Mininet Workflow

To make the experimental setup, following steps are performed in Mininet.

i. Making a Network

To create and start the network, the following command is used in the console.

"Sudo mn –topo single,3 --mac – switchovsk –controller remote"

The command creates a network of 6-hosts; each host has different IP (internet protocol) address, one open vSwitch and one remote controller, as shown in Figure 4.2. All the hosts are connected to the switch and open vSwitch is connected to the controller.

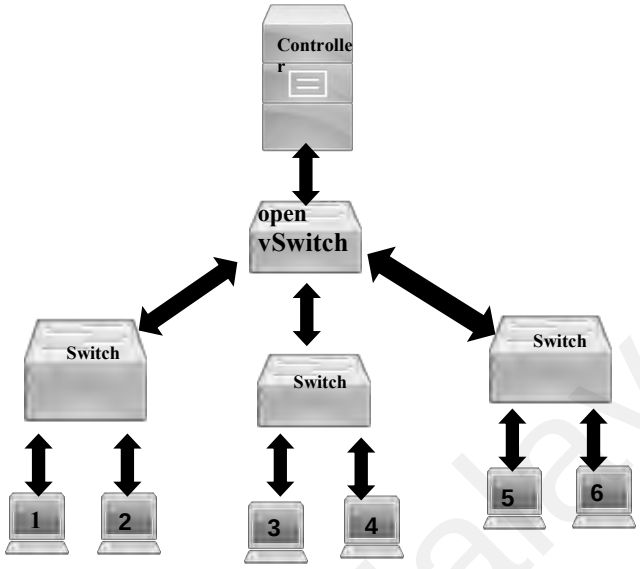


Figure 4.2: Network topology

The controller in this case is POX controller. Mininet uses the Linux platform and all the emulation is lightweight.

In order to check the connectivity, users can run the ping test. The command line interface is flexible for users and they can control the network by using one console and from the one console, the connectivity between two hosts can be checked by using the following command:

"Mininet> h1 ping h3"

It confirms that the hosts are connected or not, by showing the packet reachability to the destination.

ii. Physical Implementation

As it is mentioned earlier, that the virtual experimental setup can be implemented practically for real network. Before the implementation, there is needed to make sure that all the components of Mininet emulation must be working in a way according to the physical environment.

4.1.5 Packet Generation

To generate the DoS attack, the terminal of one host is used and the host is considered as compromised host. The command which is used to generate the DoS attack is as follow:

" $ sudo ping (destination host IP) –i .05"

It generates the packets after every 20^{th} of second and lead to a DoS attack which is used to evaluate the test bed for detection and mitigation. In DoS attack generating command, the IP address of destination host is used which results in sending all the packets to the host, who's IP address is mentioned in command.

4.2 Setting Up Environment for Experiments

The automated system is used as the detection and mitigation is taking place automatically without the network operator. The automated system consists of sFlow-rt, Pyresonance and DoS policy which works on defined threshold. For experiments the desktop system with 4GB RAM and 480GB hard drive is used. The processor was core i5 and operating system was windows 7. However on this system, virtual box and Ubuntu 14.04 64-bit is installed. The virtual operating system is allocated by 2GB RAM.

The required software to download in order to run the experiments is virtual box, Xserver and Mininet. The virtual box is used for virtualization of operating system. As

an example, it creates the instance of Ubuntu on physical machine or simply on the physical machine, another virtual machine can be installed.

Pyresonance is an application framework on top of which pyretic works, which is used to prevent the network from DoS attack.

sFlow-rt is used to monitor the network traffic and in this network, sFlow-rt is used to monitor the network traffic for preventing the SDN controller from DoS attack. In this work, the sFlow-rt is connected with controller.

4.3 Experiments

Creating Network

In this work, for four switches, one controller and 6-hosts, MiniEdit is used to make the network topology. After completing the topology on MiniEdit, the file is saved and then opened from the console by setting the path to the directory where this file exists for running the network.

To verify that the connection is working properly, ping test is performed. As mentioned in the previous section, all the software and components are installed and verified that these are working properly. The network is started in the terminal of VM.

In the second step, another terminal is opened to create the sFlow agent. The sFlow agent, as discussed earlier is connected to controller, monitors the network traffic to identify the DoS attack.

The third step is to start the sFlow to trace the network traffic. In the fourth step, implementation of DoS policy is performed which mitigates the DoS attack. To implement DoS policy, the DoS policy is copied into Pyresonance application directory by using the following command:

“ cp *.py/home/Ubuntu/pyretic/pyretic/pyresonance/apps/”

In the fifth step, third terminal is opened and started Pyresonance to mitigate the DoS attack. To accomplish this step, first the path of pyretic is set by using the following command:

“ $ cd/home Ubuntu/pyretic”

After that Pyresonance is started for mitigation of DoS attack. The following command was used to mitigate the attack:

“$./pyretic.py pyretic.pyresonance.main --config =./pyretic/pyresonance/global.config --mode=manual”

After implementing the above steps, the link connectivity is verified by pinging the hosts using pingall.

To differentiate the normal network traffic and then network traffic after attack, three test cases are considered. The host terminal is started using **xtermh1.** The host terminal is used to generate DoS attack as this host was considered as compromised host. Figure 4.3 summarizes all the steps:

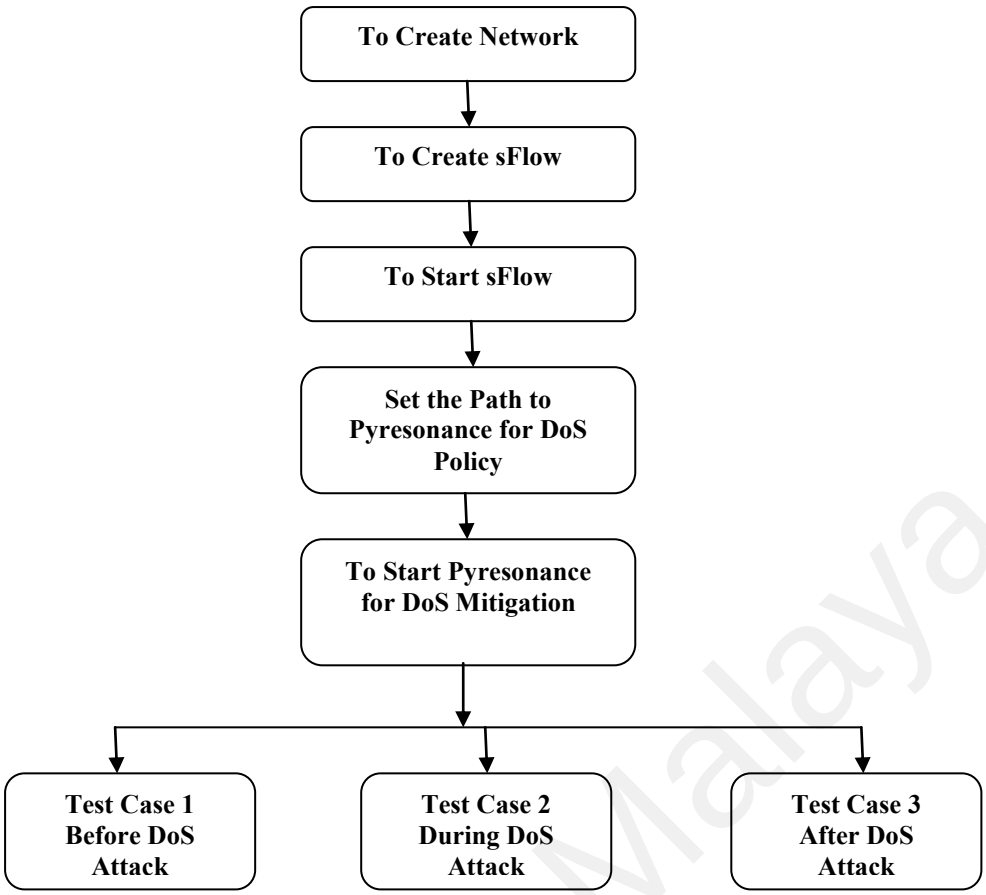


Figure 4.3: Steps for experiments

The first test case is before the launching of DoS attack which shows the normal network traffic. All the hosts are communicating with each other properly. The second test case is during DoS attack. It shows the presence of DoS attack in the network. The third test case is after DoS attack, which shows that malicious node is not able to communicate in the network. The three test cases are as below:

i. Test Case 1: Result before DoS attack

As there is no DoS attack initially, the traffic is normal as shown in Figure 4.4.

Figure 4.4: Result before DoS attack

ii. Test Case 2: Result with DoS attack

In this case, the terminal of host 1 which is connected to switch (Figure 4.2, Section 4.1.4) is used to generate DoS attack. The DoS attack is generated by using, sudo ping 10.0.0.3 –i .05. By running this command, the packets with infinite number are sent to the host 3, passing through controller. In short, the request for flow entries of same packets with greater frequency is captured by controller and as a result sFlow-rt determines the DoS Attack. In Pyresonance terminal, the messages of abnormal traffic could be observed as shown in the Figure 4.5 and the malicious node h1 is not able to communicate with target node h3 as shown in Figure 4.6. At this stage, the communication is stopped with the malicious node h1.

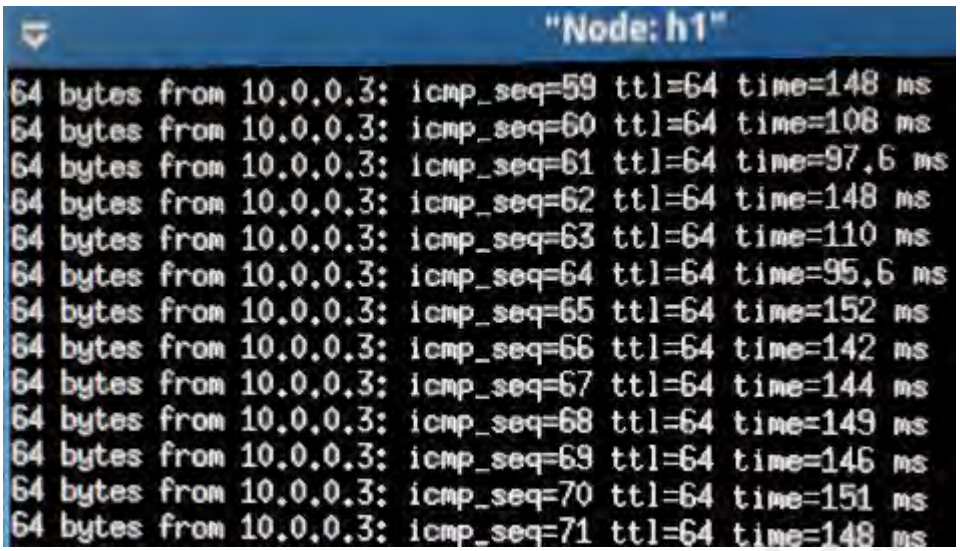


Figure 4.5: Result during DoS attack

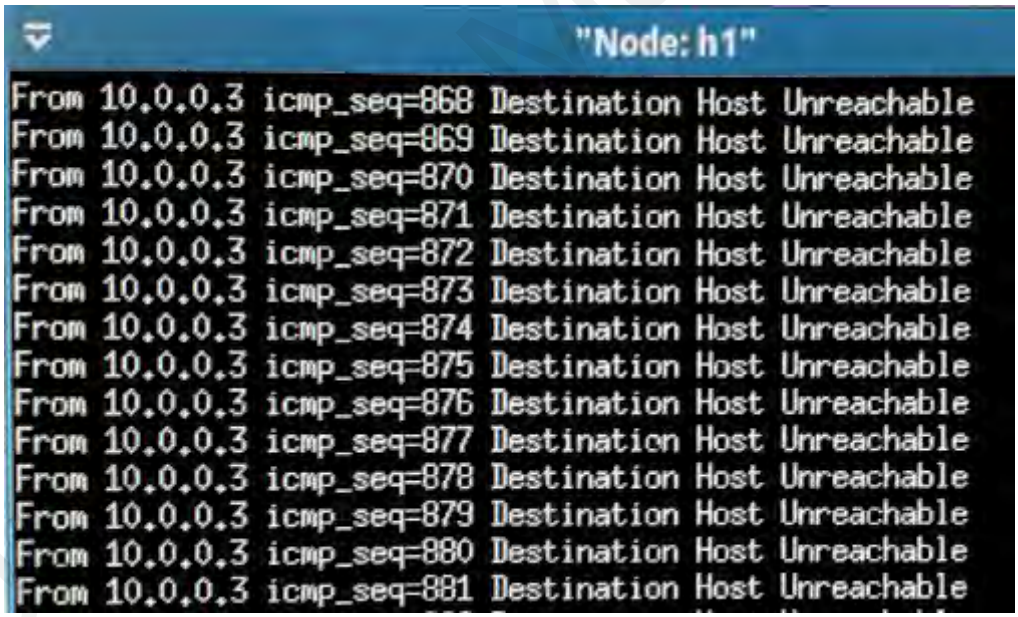


Figure 4.6: DoS attack mitigation

iii. Test Case 3: Result after DoS attack

In this case, it is verified that all the hosts are communicating with each other and only the traffic from malicious node 1 (node 1 Figure 4.2) is blocked as shown in Figure 4.7.

Figure 4.7: Result after DoS attack

From Figure 4.7, it can be observed that out of 30 packets, 22 packets are received. Each packet is approximately equal to 3.25% and total dropped packets are 8 which is approximately 26% (8*3.25=26). As all nodes are able to communicate with each other except h1 which shows that this system is mitigating the DoS attack by blocking the traffic coming from malicious node h1 and detaching the malicious node h1 from network. In addition to this, the proposed technique is for real time solution as only malicious host traffic is blocked and remaining host are communicating normally.

4.4 Evaluation

For the evaluation of this work, the usage of system resources is considered to observe the efficiency. As Pyresonance and sFlow-rt are connected with controller and determined that this designed system is lightweight. It uses the resources of system with no major difference as resources used by the normal tasks running on the system.

During experimentation, all the tasks running on the system are terminated except the simulation which was running on Ubuntu. To differentiate the resource usage without DoS attack and with DoS attack, the graph is added as shown in Figure 4.8 and 4.9 respectively.

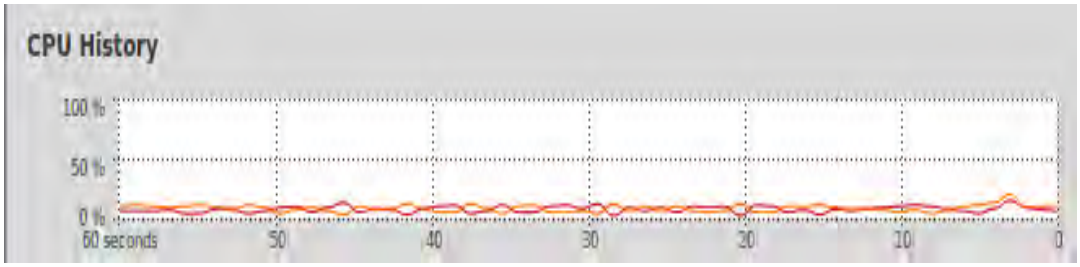


Figure 4.8: CPU usage before DoS attack

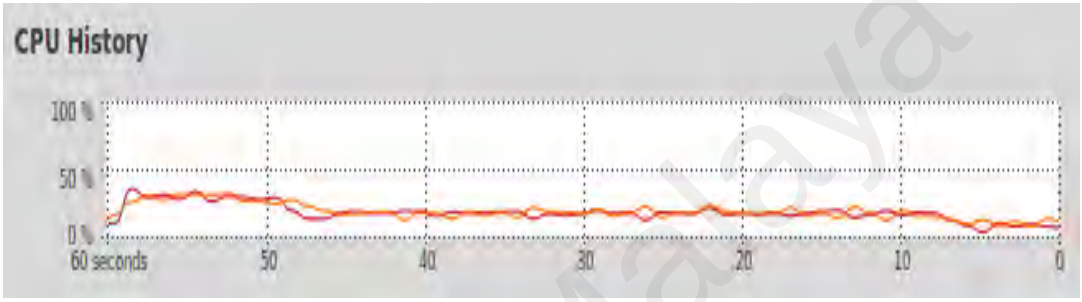


Figure 4.9: CPU usage during DoS attack

In Figure 4.9, the Y-axis which is representing the percentage of CPU usage is divided into 16 units (points) and each unit is equal to 6.25% (100/16=6.25). In Figure 4.9, the peak of CPU usage is approximately reaching to unit 6 on Y axis which means that the CPU usage is about 37.5% during DoS attack (6*units=6*6.25=37.5). At X-axis, one second is approximately equal to 3.20 units (points) which mean that total 60 seconds are equal to 192 units (60*3.2=192). The peak value is approximately at unit 4 on X-axis when the CPU usage is 37.5% during DoS attack and it is approximately equal to 1 second (3.20 units=1 second). It shows that attack appeared at 59th second when the resource usage is approximately 37.8% and removed approximately at 49th second when the resource usage becomes normal. It shows that the time taken is 10 seconds for mitigation of DoS attack.

In summary, (Gde Dharma et al., 2015), gave the idea of time-based DDoS attack detection and (Giotis et. al, 2014) proposed the combining of OpenFlow and sFlow

techinique, which is based on entropy change, to detect the DDoS attack. However, we proposed a solution which uses sFlow-rt for detection and DoS policy, defined on controller, for mitigation of DoS attack.

CHAPTER 5: CONCLUSION

This chapter gives the conclusion of the entire work. First, it explains the achievement and discusses all the chapters briefly. Finally, it discusses the future work.

This chapter is categorized into two sections. Section 5.1 gives the concluding remarks and describes our research contribution. Section 5.2 labels the limitations and suggest the future work.

5.1 Achievements

This research work presents the network security challenges to the SDN controller and then proposes a solution for detection and mitigation of the attacks. The objectives of this research work are discussed in section 1.3 of chapter 1. The summary of the achievement of objectives is as given below.

The detailed literature was reviewed in the domain of SDN security and then selected the articles which were most related to SDN controller security. The different security techniques to prevent the controller, which were discussed in selected articles, were thoroughly reviewed. The challenges, critical aspects and issues in those articles were highlighted and finally the gap in the research was identified.

The real-time detection and mitigation of DoS attack on controller in SDN environment was proposed as a solution. The architecture and the components of proposed technique were discussed in Chapter 3.

To implement the DoS attack detection mechanism on SDN controller, the sFlow-rt was connected to the controller, which notify the occurrence of attack. To mitigate the attack, DoS policy was defined on the controller using Pyresonance framework. After the occurrence of attack, the DoS policy was activated which block the traffic coming

from malicious node. The Dos policy also detached the malicious node from the network. Experiments were conducted in series for proposed technique evaluation. The network functioning was observed normal during when the DoS attack occurred which confirmed the real-time mitigation of attack. The time-based DDoS attack detection model was selected as a reference. This technique proposed the idea of detection of DDoS attack. The proposed technique is derived from this idea and implemented to get the results. The proposed technique also mitigates the DoS attack. The research contribution can be reviewed as:

i. To review the current state of the art techniques related to DoS attack detection and mitigation on controller in SDN.
ii. To design an automated system which will detect the DoS attack on controller in SDN environment and also mitigate the attack on controller in order to prevent it and block the traffic coming from malicious host in real time by using DoS policy.
iii. The implementation of the designed automated system by the use of Mininet emulator.

5.2 Future Work

In this research work, some prospective research directions are found. The research direction could be extended as a future work.

In proposed solution, one open vSwitch and three simple switches were used. The open vSwitch was connected to the controller and the normal switches were connected to the nodes. Only the open vSwitch includes the flow tables and simple switches forward the packets according to the open vSwitch instructions. It was observed that the nodes connected to the same simple switch can communicate with each other without communicating with controller and can bypass the defined DoS policy. This issue could be investigated and could be preceded for future work.

REFERENCES

Al-Shaer, E., & Al-Haj, S. (2010). *FlowChecker: Configuration analysis and verification of federated OpenFlow infrastructures.* Paper presented at the Proceedings of the 3rd ACM workshop on Assurable and usable security configuration.

Anwer, B., Benson, T., Feamster, N., Levin, D., & Rexford, J. (2013). *A slick control plane for network middleboxes.* Paper presented at the Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.

Ballard, J. R., Rae, I., & Akella, A. (2008). *Extensible and Scalable Network Monitoring Using OpenSAFE.* Paper presented at the INM/WREN.

Benton, K., Camp, L. J., & Small, C. (2013). *Openflow vulnerability assessment.* Paper presented at the Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.

Braga, R., Mota, E., & Passito, A. (2010). *Lightweight DDoS flooding attack detection using NOX/OpenFlow.* Paper presented at the Local Computer Networks (LCN), 2010 IEEE 35th Conference on.

Canini, M., Venzano, D., Perešíni, P., Kostić, D., & Rexford, J. (2012). *A NICE way to test OpenFlow applications.* Paper presented at the Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12).

Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., & Shenker, S. (2007). *Ethane: taking control of the enterprise.* Paper presented at the ACM SIGCOMM Computer Communication Review.

Casado, M., Garfinkel, T., Akella, A., Freedman, M. J., Boneh, D., McKeown, N., & Shenker, S. (2006). *SANE: A Protection Architecture for Enterprise Networks.* Paper presented at the Usenix Security.

D. K. Lee, G. Y. Bang, & Choi, D. (2014). *A DDoS Blocking System Based on SDN Using Centralized Traffic Monitoring*. Paper presented at the The 3rd International Conference on Smart Media and Applicatione.

Dao, N.-N., Park, J., Park, M., & Cho, S. (2015). *A feasible method to combat against DDoS attack in SDN network.* Paper presented at the 2015 International Conference on Information Networking (ICOIN).

Dharma, N. G., Muthohar, M. F., Prayuda, J. A., Priagung, K., & Choi, D. (2015). *Time-based DDoS detection and mitigation for SDN controller.* Paper presented at the Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific.

Fayazbakhsh, S. K., Sekar, V., Yu, M., & Mogul, J. C. (2013). *Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions.* Paper

presented at the Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.

Feamster, N., Rexford, J., & Zegura, E. (2014). The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review, 44*(2), 87-98.

Gde Dharma, N., Muthohar, M. F., Prayuda, J., Priagung, K., & Choi, D. (2015). *Time-based DDoS detection and mitigation for SDN controller.* Paper presented at the Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific.

Giotis, K., Argyropoulos, C., Androulidakis, G., Kalogeras, D., & Maglaris, V. (2014). Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. *Computer Networks, 62*, 122-136.

Goodney, A., Narayan, S., Bhandwalkar, V., & Cho, Y. H. (2010). *Pattern based packet filtering using NetFPGA in DETER infrastructure.* Paper presented at the 1st Asia NetFPGA Developers Workshop, Daejeon, Korea.

Handigol, N., Heller, B., Jeyakumar, V., Maziéres, D., & McKeown, N. (2012). *Where is the debugger for my software-defined network?* Paper presented at the Proceedings of the first workshop on Hot topics in software defined networks.

Jafarian, J. H., Al-Shaer, E., & Duan, Q. (2012). *Openflow random host mutation: transparent moving target defense using software defined networking.* Paper presented at the Proceedings of the first workshop on Hot topics in software defined networks.

Kloti, R., Kotronis, V., & Smith, P. (2013). *OpenFlow: A security analysis.* Paper presented at the Network Protocols (ICNP), 2013 21st IEEE International Conference on.

Kreutz, D., Ramos, F., & Verissimo, P. (2013). *Towards secure and dependable software-defined networks.* Paper presented at the Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE, 103*(1), 14-76.

Lerner, M., Vanecek, G., Vidovic, N., & Vrsalovic, D. (2006). *Middleware Networks: Concept, Design and Deployment of Internet Infrastructure* (Vol. 18): Springer Science & Business Media.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., . . . Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review, 38*(2), 69-74.

Mehdi, S. A., Khalid, J., & Khayam, S. A. (2011). *Revisiting traffic anomaly detection using software defined networking.* Paper presented at the Recent Advances in Intrusion Detection.

Mizrahi, T., & Moses, Y. (2013). Time-based Updates in OpenFlow: A Proposed Extension to the OpenFlow Protocol. *Technion-- Israel InsYtute of Technology, technical report, CCIT Report, 835*.

Mousavi, S. M. (2014). *Early Detection of DDoS Attacks in Software Defined Networks Controller.* Carleton University Ottawa.

Mousavi, S. M., & St-Hilaire, M. (2015). *Early detection of DDoS attacks against SDN controllers.* Paper presented at the Computing, Networking and Communications (ICNC), 2015 International Conference on.

Naous, J., Stutsman, R., Mazières, D., McKeown, N., & Zeldovich, N. (2009). *Delegating network security with more information.* Paper presented at the Proceedings of the 1st ACM workshop on Research on enterprise networking.

Nayak, A. K., Reimers, A., Feamster, N., & Clark, R. (2009). *Resonance: dynamic access control for enterprise networks.* Paper presented at the Proceedings of the 1st ACM workshop on Research on enterprise networking.

Nugraha, M., Paramita, I., Musa, A., Choi, D., & Cho, B. (2014). Utilizing OpenFlow and sFlow to Detect and Mitigate SYN Flooding Attack. *Journal of Korea Multimedia Society, 17*(8), 988-994.

Nunes, B. A., Mendonca, M., Nguyen, X.-N., Obraczka, K., & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys & Tutorials, IEEE, 16*(3), 1617-1634.

Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M., & Gu, G. (2012). *A security enforcement kernel for OpenFlow networks.* Paper presented at the Proceedings of the first workshop on Hot topics in software defined networks.

Prasad, B. (2014). *Security Issues in Software Defined Networking.* Jadavpur University Kolkata.

Qazi, Z. A., Tu, C.-C., Chiang, L., Miao, R., Sekar, V., & Yu, M. (2013). *SIMPLE-fying middlebox policy enforcement using SDN.* Paper presented at the ACM SIGCOMM Computer Communication Review.

Salman, O., Elhajj, I. H., Kayssi, A., & Chehab, A. (2016). *SDN controllers: A comparative study.* Paper presented at the 2016 18th Mediterranean Electrotechnical Conference (MELECON).

Shin, S., & Gu, G. (2012). *CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?).* Paper presented at the Network Protocols (ICNP), 2012 20th IEEE International Conference on.

Shin, S., Porras, P. A., Yegneswaran, V., Fong, M. W., Gu, G., & Tyson, M. (2013). *FRESCO: Modular Composable Security Services for Software-Defined Networks.* Paper presented at the NDSS.

Skowyra, R. W., Lapets, A., Bestavros, A., & Kfoury, A. (2013). *Verifiably-safe software-defined networks for CPS.* Paper presented at the Proceedings of the 2nd ACM international conference on High confidence networked systems.

Son, S., Shin, S., Yegneswaran, V., Porras, P., & Gu, G. (2013). *Model checking invariant security properties in OpenFlow.* Paper presented at the Communications (ICC), 2013 IEEE International Conference on.

Vanecek, G., Mihai, N., Vidovic, N., & Vrsalovic, D. (1999). Enabling hybrid services in emerging data networks. *IEEE Communications Magazine, 37*(7), 102-109.