

**EXTENDED TOPOLOGY ANALYSIS OF A  
DETECTION MECHANISM IMPLEMENTATION  
AGAINST BOTNET BASED DDOS FLOODING  
ATTACK IN SDN**

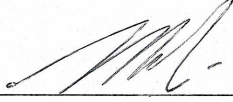
**A Thesis Submitted to  
the Graduate School of Engineering and Sciences of  
İzmir Institute of Technology  
in Partial Fulfillment of the Requirements for the Degree of  
MASTER OF SCIENCE  
in Computer Engineering**

**by  
Emre KARAKIŞ**

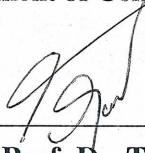
**June 2019  
İZMİR**

We approve the thesis of Emre KARAKIŞ

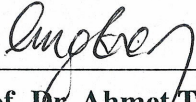
Examining Committee Members:



**Assoc. Prof. Dr. Yusuf Murat ERTEN**  
Department of Computer Engineering, İzmir Institute of Technology



**Assoc. Prof. Dr. Tolga AYAV**  
Department of Computer Engineering, İzmir Institute of Technology



**Assoc. Prof. Dr. Ahmet Tuncay ERCAN**  
Department of Computer Engineering, Yaşar University

28 June 2019



**Assoc. Prof. Dr. Yusuf Murat ERTEN**  
Supervisor, Department of Computer Engineering  
İzmir Institute of Technology



**Dr. Emrah TOMUR**  
Co-Supervisor, Master Researcher  
Ericsson



**Assoc. Prof. Dr. Tolga AYAV**  
Head of the Department of  
Computer Engineering

**Prof. Dr. Aysun SOFUOĞLU**  
Dean of the Graduate School of  
Engineering and Sciences

## ACKNOWLEDGMENTS

I would like to express my gratitude to my thesis supervisor Assoc. Prof. Dr. Yusuf Murat Erten and my co-supervisor Dr. Emrah Tomur for their guidance and extraordinary supports. This thesis work completely becomes distinctive thanks to them. Without their precious insights, this work would not be completed as it is.

Besides my thesis advisors, I am also thankful to other professors in IYTE whom I have taken course during my graduate study for providing their valuable knowledge to make the thesis study concrete.

Furthermore, my special thanks go to my colleagues Murat Öz, Ali Nehrani, Fatih Türkmen, Didem Genç, Kemal Kayahan and Research Assistant Emre Şahin at Pamukkale University for their continuous contributions, patience and valuable advices.

Additionally, I am especially thankful to Prof. Dr. Mehmet Saltan for his guidance and spiritual supports.

Last, but not least, I must thank my gratitude to my family who always supported me throughout my education.

# ABSTRACT

## EXTENDED TOPOLOGY ANALYSIS OF A DETECTION MECHANISM IMPLEMENTATION AGAINST BOTNET BASED DDOS FLOODING ATTACK IN SDN

When SDN comes up as a new technology, while it also brings many benefits such as high availability, scalability and performance, it also brings us new vulnerabilities that is targeted by attackers. Botnet Based DDoS Flooding Attacks have been one of the major problems for service provider networks who encountered these repeatedly since the first DDoS came into existence in the early 2000's. In this thesis, we mainly concentrate on the source-based detection approach against Botnet Based DDoS Flooding Attack by combining the strength of SDN and s-Flow-RT technology.

The main purpose of this research is to detect Botnet Based DDoS Flooding Attack that can also be performed in distributed SDN environments by using a similar approach with an available detection mechanism which is not implemented previously on an extended network with more network elements in order to observe whether the obtained successful results on the small network are compatible with a result obtained on this research. This study also includes a detection application using previously studied detection approach based on statistical inference model. The detection application is tested on virtual environments by organizing a Botnet Based DDoS Flooding Attacks on a predefined source node and then test results show that the mechanism could effectively detect the attack.

## ÖZET

### YAZILIM TANIMLI AĞLARDA BOTNET TEMELLİ DAĞITIK HİZMET DIŞI BIRAKMA SALDIRILARINA KARŞI BİR TESPİT MEKANİZMASININ GENİŞLETİLMİŞ TOPOLOJİ ANALİZİ

Yazılım Tanımlı Ağ yeni bir teknoloji olarak ortaya çıktığında, yüksek kullanılabilirlik, ölçeklenebilirlik ve performans gibi pek çok avantaj getirirken, aynı zamanda da saldırganların hedef aldığı yeni güvenlik açıklıklarının da beraberinde getiriyor. Botnet Temelli Dağıtık Hizmet Dışı Bırakma Saldırıları, 2000’li yılların başında ilk Dağıtık Hizmet Dışı Bırakma Saldırısının ortaya çıkmasından beri bunlarla tekrar tekrar karşılaşan servis sağlayıcı ağlar için önde gelen siber suçlardan biri olmuştur. Bu tezde ağırlıklı olarak Botnet Temelli Dağıtık Hizmet Dışı Bırakma Saldırılarına karşı Yazılım Tanımlı Ağ ve s-Flow-RT teknolojisinin güçlerini birleştirerek kaynak temelli tespit yaklaşımına odaklanıyoruz. Bu araştırmanın temel amacı, daha küçük bir ağ üzerinde elde edilen başarılı sonuçların, bu çalışmada elde edilen sonuçlarla uyumlu olup olmadığını görmek amacıyla, Dağıtık Yazılım Tanımlı Ağ ortamlarında da uygulanabilen Botnet Temelli Dağıtık Hizmet Dışı Bırakma Saldırılarını, daha fazla ağ elemanı ile genişletilmiş bir ağ üzerinde daha önce uygulanmamış olan mevcut bir tespit mekanizması ile benzer bir yaklaşım kullanılarak tespit etmektir. Bu çalışma aynı zamanda istatistiksel çıkarım modeline dayanan daha önce çalışılmış tespit yaklaşımını kullanan bir tespit uygulaması içermektedir. Tespit uygulaması sanal ortamlarda önceden tanımlanmış bir kaynak düğümünde Botnet Temelli Dağıtık Hizmet Dışı Bırakma Saldırısı düzenleyerek test edilir ve test sonuçları, mekanizmanın saldırıyı etkili bir şekilde algılayabildiğini gösterir.

# TABLE OF CONTENTS

LIST OF FIGURES .....	ix
LIST OF TABLES .....	xi
LIST OF ABBREVIATIONS .....	xii
CHAPTER 1. INTRODUCTION .....	1
1.1. Aim of the Thesis and Objectives .....	3
1.2. Organization of Thesis .....	4
CHAPTER 2. RELATED WORK .....	6
2.1. A Feasible Method to Combat against DDoS Attack.....	6
2.2. DDoS Blocking Application .....	7
2.3. FlowTrApp .....	7
2.4. Easy Defence Mechanism Against Botnet Based DDoS Flooding Attacks .....	9
CHAPTER 3. BACKGROUND .....	11
3.1. Software Defined Network .....	11
3.2. OpenFlow Switch .....	14
3.2.1. OpenFlow Messages .....	16
3.3. SDN Architecture .....	17
3.4. Security Threats .....	19
3.4.1. Spoofing .....	20
3.4.2. Tampering .....	21
3.4.3. Repudiation and Non-repudiation .....	22
3.4.4. Accountability .....	23
3.4.5. Information Disclosure.....	23
3.4.6. Denial of Service Attacks (DoS) .....	25
3.4.7. Distributed Denial of Service Attacks.....	27
3.4.8. Elevation of Privilege .....	29

CHAPTER 4. ENVIRONMENT REQUIREMENTS .....	31
4.1. SDN Controllers Overview .....	31
4.2. ONOS .....	32
4.3. Mininet .....	33
4.3.1. Mininet Advantages .....	33
4.4. s-Flow & s-Flow-RT .....	34
4.4.1. Interoperability of Applications .....	35
4.4.2. RESTflow .....	35
4.4.3. Network Wide Visibility of sFlow .....	38
4.5. Bash Script Application .....	38
4.6. ONOS Cell Mechanism .....	39
4.7. Hping3 .....	40
4.7.1. TCP/IP Communication .....	40
4.7.2. Hping Arguments .....	41
4.7.2.1. Hping3 Modes .....	41
4.7.2.2. Hping3 Flag Options .....	41
4.7.2.3. Hping3 IP Related Options .....	42
4.7.2.4. TCP/UDP Related Options .....	42
4.7.2.5. Common Options .....	43
CHAPTER 5. IMPLEMENTATION .....	46
5.1. Detection Method .....	46
5.1.1. Calculation of $\delta$ Parameter .....	47
5.1.2. Calculation of $\rho$ Parameter .....	48
5.1.3. Example Calculation of $(\delta, \rho)$ .....	48
5.1.4. $\lambda$ Parameter Calculation of Poisson .....	51
5.1.5. Threshold $\delta$ Calculation .....	52
5.1.6. Threshold $\rho$ Calculation .....	52
5.1.7. The Detection Procedure .....	52
5.2. Design of the Application .....	53
5.3. Cluster Communication Methods .....	55
5.3.1. GRE Tunnels .....	56
5.3.2. SDN-IP Application .....	58

CHAPTER 6. EXPERIMENT AND ANALYSIS .....	60
6.1. Distributed Network Generation.....	60
6.2. Experiment with Multiple Controllers running on Two Physical Computers .....	73
6.2.1. Normal Traffic Test .....	73
6.2.2. Attack Traffic Test.....	77
6.3. Experiment with Large Network on the Single Controller .....	79
6.3.1. Initial Experiment .....	79
6.3.2. Normal Traffic Experiment .....	80
6.3.3. Attack Traffic Experiment .....	83
 CHAPTER 7. CONCLUSION .....	 85
 REFERENCES .....	 86
 APPENDICES	
APPENDIX A. INSTALLING ONOS CONTROLLER .....	89
APPENDIX B. INSTALLING AND USING MININET .....	92
APPENDIX C. S-FLOW-RT RELATED CONFIGURATIONS .....	96
APPENDIX D. BASH SCRIPT APPLICATION AND ONOS CELL CONFIGURATION .....	99
APPENDIX E. HPING3 INSTALLATION, USAGE AND DDOS ORGANIZATION .....	100
APPENDIX F. AN EXAMPLE OF CONSTRUCTING GRE TUNNEL .....	104



# LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
Figure 1.1. DDoS Attack Structure-1 .....	3
Figure 2.1. Distribution Collaboration Two Dimension Plot .....	10
Figure 3.1. A Simple SDN Network with OpenFlow Switches .....	13
Figure 3.2. Flow Processing in OpenFlow Protocol .....	14
Figure 3.3. SDN Architecture .....	18
Figure 3.4. DDoS Attack Structure-2 .....	27
Figure 4.1. Flow Cache Embedded on the Switch .....	36
Figure 4.2. Packet Export and Flow Export Mechanism .....	37
Figure 4.3. s-Flow-RT on SDN Architecture .....	37
Figure 4.4. Bash Script Application .....	39
Figure 4.5. Three-way-handshake Process .....	41
Figure 5.1. A Sample Graph of Nodes with $(\delta, \rho)$ Calculation .....	49
Figure 5.2. Communication Structure for Application .....	54
Figure 5.3. Deployment Scenario for Application .....	55
Figure 5.4. Encapsulated Payload Packet .....	57
Figure 5.5. A simple Two Switch Topology .....	57
Figure 5.6. SDN-IP Architecture .....	58
Figure 6.1. Running ONOS Controller .....	61
Figure 6.2. ONOS CLI .....	62
Figure 6.3. Mininet CLI .....	62
Figure 6.4. Mininet Python File .....	63
Figure 6.5. Initial Topology View on First Controller .....	63
Figure 6.6. Generating Traffic on ONOS .....	64
Figure 6.7. Pingall Execution in Mininet .....	65
Figure 6.8. Connecting Other Machine with SSH .....	65
Figure 6.9. Initial Topology View on Second Controller .....	66
Figure 6.10. Ping Test on Second Controller .....	67
Figure 6.11. PingAll Test on Second Controller .....	67
Figure 6.12. Cluster Combination .....	68
Figure 6.13. Combined Topology View .....	68

Figure 6.14. Installing sFlow Agents .....	69
Figure 6.15. Flows By Time Graph of sFlow .....	70
Figure 6.16. Normal ICMP Traffic Visualization on sFlow .....	70
Figure 6.17. Attack Traffic Visualization on sFlow .....	71
Figure 6.18. Flow Definition Parameters on sFlow .....	71
Figure 6.19. Observing Flow as Kbps value on ONOS .....	72
Figure 6.20. Observing Flow as Mbps value on ONOS .....	72
Figure 6.21. Observing Flow as Gbps value on ONOS .....	73
Figure 6.22. Distributed ONOS Clusters Combined Using GRE Tunnel .....	74
Figure 6.23. Result In Normal Situation on Multiple Clusters .....	75
Figure 6.24. DCD Graph For Normal Situation on Multiple Clusters .....	75
Figure 6.25. TCP Normal Flow Observation on TCP Flow Cache .....	76
Figure 6.26. Poisson Probabilities For Normal Situation on Multiple Clusters .....	76
Figure 6.27. Result In Attack Situation on Multiple Clusters .....	77
Figure 6.28. DCD Graph For Attack Situation on Multiple Clusters .....	78
Figure 6.29. Poisson Probabilities For Attack Situation on Multiple Clusters .....	78
Figure 6.30. Single Controller Topology .....	80
Figure 6.31. Result In Normal Situation On Single Controller .....	81
Figure 6.32. DCD Graph For Normal Situation On Single Controller .....	82
Figure 6.33. Poisson Probabilities For Normal Situation On Single Controller .....	82
Figure 6.34. Result In Attack Situation On Single Controller .....	83
Figure 6.35. DCD Graph Attack Situation On Single Controller .....	84

# LIST OF TABLES

<u>Table</u>		<u>Page</u>
Table 3.1.	Packet Header Fields on Flow Tables .....	15
Table 3.2.	Actions of OpenFlow Switch .....	15
Table 3.3.	Counter Statistic Information .....	16
Table 4.1.	Hping3 Basic Arguments .....	42
Table 4.2.	Hping3 Modes .....	42
Table 4.3.	Hping3 Flag Options .....	43
Table 4.4.	Hping3 IP Options .....	43
Table 4.5.	Hping3 TCP/UDP Related Options .....	44
Table 4.6.	Hping3 Common Options .....	44
Table 5.1.	Flow Table of Nodes on Figure 5.1 .....	50

## LIST OF ABBREVIATIONS

SDN	Software Defined Network
DDoS	Distributed Denial of Service
API	Application Programming Interface
IP	Internet Protocol
ONOS	Open Networking Operating System
GRE	Generic Routing Encapsulation
REST	Representational State Transfer
URL	Uniform Resource Locator
HTTP	Hyper-Text Transfer Protocol
DBA	DDoS Blocking Application
CAPTCHA	Completely Automated Public Turing Test to tell Computers and Humans Apart
DCD	Distribution Collaboration Degree
RDB	Master Resource Database
OSS	Open Source System
IPS	Intrusion Prevention System
IDS	Intrusion Detection System
TLS	Transport Layer Security
ONF	Open Networking Foundations
OFPT	OpenFlow based Parallel Transport
NIC	Network Interface Card
VM	Virtual Machine
ARP	Address Resolution Protocol
MAC	Media Access Control
ARM	Address Resolution Mapping
SSL	Secure Socket Layer
DNS	Domain Name Server
PKE	Public Key Encryption
NAT	Network Address Translation
VPN	Virtual Private Network
DoS	Denial of Service
ACK	Acknowledgment

TCP .....	Transport Control Protocol
CIAC .....	Computer Incident Advisory Capability
UDP .....	User Datagram Protocol
IoT .....	Internet of Things
UDP .....	Open System Interconnection
RBAC .....	Role Based Access Control
IBM .....	International Business Machines
HP .....	Hewlett Packard
NRO .....	Network Resource Optimization
BNC .....	Big Network Controller
CLI .....	Command Line Interface
GUI .....	Graphical User Interface
OSGi .....	Open Service Gateway Initiative
SSH .....	Secure Shell
ToS .....	Type of Service
ICMP .....	Internet Control Message Protocol
GNOME .....	GNU Network Object Model Environment
MTU .....	Maximum Transmission Unit
FTP .....	File Transfer Protocol
SYN .....	Synchronize
AS .....	Autonomous System
BGP .....	Border Gateway Protocol
BYON .....	Build Your Own Network
IPv4 .....	Internet Protocol Version 4
SSD .....	Solid State Drive
CPU .....	Central Processing Unit
RAM .....	Random Access Memory
HDD .....	Hard Disk Drive
MacOS .....	Macintosh Operating System
JSON .....	JavaScript Object Notation
NULL .....	Nation Under Lethal Limitations
TTL .....	Time To Live

# CHAPTER 1

## INTRODUCTION

Recently, with the emergence of new networking environments such as cloud computing and internet of things environments, the networking paradigms have inevitably been updated. The dynamic nature of networking environments has always been open to change in time. This condition leads to increase in the complexity of today's networks and it requires to simplify the network management and aggregate the management in one center. At that point, when a new networking technology evolves, a new requirement appears to make the network more scalable and dynamic to be able to manage network devices in a good way. These requirements can be handled with programmable networks such as Software Defined Networks. SDN is also a new networking approach decoupling control plane from the data plane. The control plane is logically centralized and responsible for forwarding decisions. At one hand, all devices in traditional networks make forwarding decisions themselves, on the other hand all devices in SDN environment are dumb. SDN separates control plane from the data plane. This separation requires the implementation of new network services such as traffic engineering, access control, bandwidth management etc (Murtuza and Asawa, 2018). The data plane composed of network devices and these devices have some flow entries on their flow tables. In order for devices to communicate with the controller, there is an OpenFlow protocol to be able to provide a secure communication. Each flow entry has matching rules and actions. When the packet arrives at the network element, on the condition that the rules are matching with the incoming packet header fields, the packets are allowed to pass through the network element. Otherwise, packets are forwarded to the controller for further processing. After that, the controller sets a flow rule for the corresponding packet header fields. The flow rule is installed on the flow table of the network devices.

Compared to traditional networks, SDN consists of three layers which are Application Layer, Control Layer and Infrastructure Layer. Control Layer has network services and SDN controller and network manager. In Application Layer, there are many business applications such as firewalls, IDS and IPS that can communicate with the Control Plane. This communication is handled by the Northbound API of the SDN controller.

In Infrastructure Layer, network devices such as routers, switches and hosts take place. The communication among devices and controller is handled with the Southbound API generally known as OpenFlow protocol. There are also Eastbound and Westbound APIs of the SDN network. These APIs perform controller to controller communication.

The central location of the controller has been a vulnerability for different security attacks. One of the major security concerns for this condition is the distributed denial of service flooding attacks on SDN environments. While the controller can be a target for DDoS attacks, the network devices of SDN can also be a target point. The first scenario for such attack is to exhaust the communication channel between the controller and the network devices. When the attack packets come up to the switches of the SDN network, a new flow rule needs to be installed on the switch, as a result, switches use memory resources and processing power for the large packets of the attack. This situation leads to the disruption of memory and the overflow of the flow tables. Consequently, the network element can not serve the arbitrary packets coming from the other locations. When the scenario for the controller is considered, the controller also uses the processing power and the memory resources. Large packets for the attacks can not be handled by the controller in terms of handling new flow entries for non-existing flows on the switches when the attack happens. The most common characteristics of DDoS attack is to disrupt a legitimate user's connectivity or services by exhausting limited resources, including network resources and server resources such as bandwidth, router processing capacity, memory and CPU (Lu and Wang, 2016). When a DDoS attack is organized on a certain destination, the attack is generally performed by some malicious users hiding themselves behind the scene. DDoS attacks may be performed in different ways. The one way is to use the methods which spoofs the vulnerable server resources, then floods a targeted victim with huge amount of internet traffic.

The other method is to use Botnet Based methods. Attackers generally seize a set of computers which are unaware of what is happening by using a group of methods such as worms, Trojan horses or back-doors. After that, attackers continuously send requests to orchestrate the bots with regard to redirect the traffic to a certain victim address. The packets going to the victim at a certain time are turned into the huge flows. By that way, victim server can not fulfill the traffic expectedly. This result leads the server to be unavailable. While DDoS attacks have been a subject for traditional networks for two decades, it is also possible for SDN networks to organize DDoS attacks in a local network inasmuch as these attacks might usually come from outside of the network, however, the

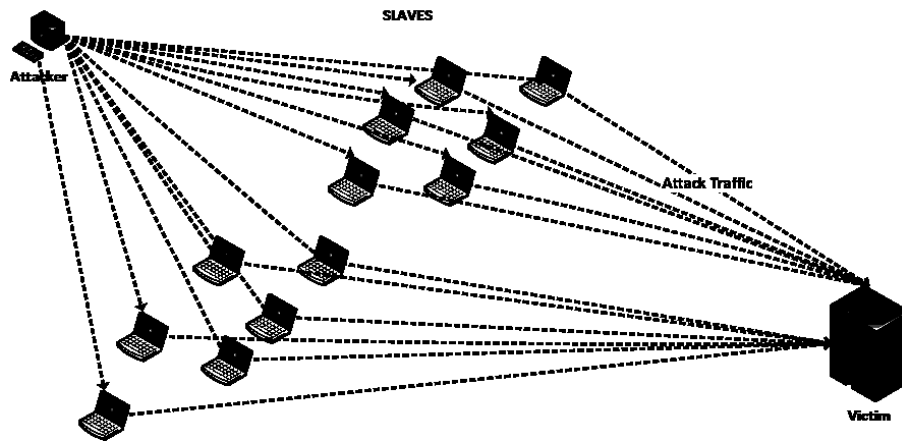


Figure 1.1. DDoS Attack Structure-1

attack may also stem from the inside for the networks composed of the multiple tenants, specially malicious tenants (Lu and Wang, 2016). Due that reason, it is quite important to perform DDoS attack study on distributed SDN networks by detecting the attack effectively with regard to shorten the required time to mitigate the attack. While detecting the attack, it is not only block the source IP addresses causing to the attack, but also it is important to prevent the spread of the attack by separating the attack traffic from the legitimate traffic (Murtuza and Asawa, 2018).

## 1.1. Aim of the Thesis and Objectives

This study mainly aims at detecting the DDoS attacks by using sflow technology on the distributed SDN environments. According to the information obtained through the literature search, we searched out an article which is specified on the references part. We decided to go through with the article of (Lu and Wang, 2016). We mainly concentrated on the future work of this article which is not implemented previously as an extended network with more network elements created by the Mininet emulation tool in order that the available successful results on the small network performed by the (Lu and Wang, 2016) are compatible with a result that will be obtained on this research. The general purpose of the research is to detect Botnet Based DDoS Flooding Attack that can also be implemented in distributed SDN environments by using the same approach with the available detection mechanism based on the approach of aggregation of flows. Addition-



ally, we also searched out another article called FlowTrApp developed by (Buragohain and Medhi, 2016) distinguishing the attack flows from the legitimate traffic and depending on the per flow based detection approach. Each flow is categorized as high rate or low rate and long lived or short lived attack flow according to two per flow based parameters like flow rate and flow duration (Buragohain and Medhi, 2016). This approach also uses the statistical sampling approach provided by s-Flow RT analytics engine. The algorithm supported by authors of (Lu and Wang, 2016) and used in this research focus on the approach of aggregation analysis of flows instead of categorizing individual flow. In addition to this, the other aim is to construct a distributed network architecture with a recent technology called ONOS in order one cluster to communicate with the other. After providing the communication of two ONOS clusters by setting the GRE tunnels for the specified switches of different clusters. It is essential to implement a DDoS Flooding Attack from the network under the first cluster to the network under the other cluster. There are a group of method to the communication of clusters. Two of them will be illustrated on the section 5.3 in Chapter 5. As an additional purpose, we have planned to observe the corresponding flows of each victim node by using s-Flow RT technology. Both ONOS and s-Flow-RT have REST APIs to be able to talk with external applications. As a final aim, we will write an application getting the topology information from Northbound API of ONOS and the flow based information from the s-Flow-RT REST API. In this application, we have inspired by the article (Lu and Wang, 2016) which implements a modular application.

## **1.2. Organization of Thesis**

For the remainder of this thesis, Chapter 2 covers the related works on targeting the DDoS Flooding detection approaches on SDN environments and recent studies related to such attacks are summarized. In Chapter 3, the needed background information related with SDN and OpenFlow architecture and some familiar threats that can be performed on SDN networks and its countermeasures are discussed according to our preliminary research based on the survey of (Alsmadi and Xu, 2015). In Chapter 4, several environment requirements are explained and indicated how it is installed and configured on Linux based environments for DDoS attack organization and the detection on the distributed SDN environments. In Chapter 5, several communication methods of distributed clusters

are discussed with regard to be able to send traffic from one cluster to the other cluster. The detection algorithm composed of the available analysis of (Lu and Wang, 2016) are mentioned on that part. Lastly, the design of the application is mentioned and emphasized how it is communicating with other applications such as ONOS and sFlow-RT. Chapter 6 presents how distributed networks are constructed on two physical machines and covers the comparison of obtained simulation results with the study performed on local network on mentioned article and the information about the experiment is given and evaluated on that part. Chapter 7 highlights some future works and concludes the paper.

## CHAPTER 2

### RELATED WORK

In traditional network environments, exhaustive solutions against DDoS attacks have been proposed for decades. There are fairly many approaches for the detection and countermeasure methods to detect or overcome such attacks. In this chapter, we have examined a group of works performed in SDN environments in terms of the detection and countermeasure methods.

#### 2.1. A Feasible Method to Combat against DDoS Attack

SDN environment is quite vulnerable to DDoS Flooding attacks. Inasmuch as the controller-switch communication channel is a potential target for the attackers (Dao et al., 2015). As a result of sending large amount of traffic, controller-switch channel can be shutdown for the communication of network components with controller. (Dao et al., 2015) proposed a feasible approach based on the IP filtering technique to combat against DDoS attacks in SDN environment. The method analyzes the user traffic to detect and prevent the attack. When collecting and analyzing the traffic, authors performed an experiment at University of Auckland network for the period of one month. According to the result of the analysis, approximately 90% of the frequent users send at least 5 packets to each destination. On the other hand, abnormal users transmitted less than 5 packets per connection. For two weeks period, 60% IP addresses appeared on only one day. Depending on the traffic analysis which is obtained by the experiment, the method aims at generating a table T for holding the IP addresses and their statistic counters for the redirected packets by the switches on the controller. Each unique IP address has a counter value of  $c_i$  to specify how much times a request sent by the particular IP address. When the attack is performed, a new packet is forwarded to the controller if it has not matching header fields with the existing flow entries. The controller initially knows that the coming packet from the switch may be a possible attack packet. A specific entry is created by the controller by using *idle\_timeout* and *hard\_timeout* variables to limit the lifetime of the flow. Then, the source IP address is updated on the table T and the counter parameter is

increased by one. When  $c_i$  variable reaches the boundary of  $k$  which is admitted as an average connection of the frequent users by the experiment, average number of packet counter statistic  $s$  is taken by the switch. Then,  $s$  counter statistic parameter is compared with the minimum average number of packets per connection which is defined as  $n$ . If  $s$  variable is smaller than  $n$ , the traffic is assessed as a malicious traffic. Because, according to the observation by the experiment, abnormal users have less packets per connection than the frequent users. Otherwise, it can be classified as a frequent user. After the user classification has been done, then they are blocked by defining a drop rule issued by the controller.

## 2.2. DDoS Blocking Application

SDN can be utilized to handle the difficulty of DDoS attacks. (Lim et al., 2014) proposed a technique utilizing the power of SDN on the network elements and using URL redirection methods against HTTP based flooding attacks. Paper suggests that SDN can be used as a key factor on its configurability on the network devices to develop a defence mechanism against botnet based DDoS attacks protecting the server accommodating in SDN network. (Lim et al., 2014) developed a DBA application running on the POX controller and protecting a specified server connected to an OpenFlow enabled switch. There are  $n$  number of legitimate hosts and  $k$  number of illegitimate hosts outside of the SDN network. Bots particularly target the server address which is specified as  $D$ . DBA application generates a set of IP addresses in order to use for a redirection of the bots to the fake server address instead of redirecting on the real server address  $D$  under an attack condition. DBA retrieves the flows and processes them and the server observes possible attack metrics, the server notifies the DBA application to produce CAPTCHA coded address from the pool.  $D'$  address is suggested to the bots. In condition that,  $D'$  is recognized by the malicious side,  $D''$  is suggested by the DBA application. The proposed algorithm just emphasis on the protection scheme assuming that the detection is performed by the server. For the point of the classification, a threshold value is determined for a client requesting a certain IP address. For a specific client, when the request number of client exceeds the threshold value, the client is classified as a bot, and then the corresponding packets are dropped. When the attack is realized, the bots are redirected to  $D'$  whereas the normal user can access the real  $D$  address of the server.

### 2.3. FlowTrApp

FlowTrApp is an SDN framework for data centers which performs DDoS detection and mitigation depending on two per flow based traffic parameters such as flow rate and flow duration of a flow. The application uses the OpenFlow protocol and s-Flow technology. s-Flow technology is a collector technology gathering flow statistics from the network elements. The application separates the attack flow from the normal flow by categorizing each flow from high rate to low rate and long lived to short lived. The application requires specific L7 constraint defined by the application admin. For example, nowadays many web application allows user to perform the operation in a single transaction at a time (Buragohain and Medhi, 2016). In order to work the application well, the application admin should define the two per flow based parameters according to the behavior of a legitimate user pattern. A legitimate user pattern is determined by observing the user behavior to understand how much traffic rate is sent by a legitimate user to perform the operation and how long it takes. FlowTrApp gets the threshold values from the application admin and it constructs a traffic tuple composed of minimum flow rate, maximum flow rate, minimum flow duration and maximum flow duration. Then, it classifies a traffic flow as attack traffic or normal traffic. Each flow has a flow rate and flow duration to compare the incoming flow with the boundary values. If two per flow based parameters of an incoming flow falls into the interval which is defined with a traffic tuple, then this flow can be categorized as a legitimate flow. (Buragohain and Medhi, 2016) proposed significant categories for attack flows. If an incoming flow has greater than the upper limit and has less duration than the lower limit, then the attack is called as High Rate Spike Attack. If the incoming flow has greater than the upper limit and has less duration than the upper limit of duration, then the attack is called as Short Lived High Rate Attack. If the incoming flow has greater than the upper limit and has bigger duration than the upper limit of duration, then the attack is called as Long Lived High Rate Attack. If the incoming flow falls into the legitimate range for the flow rate and has bigger duration than the upper limit of duration, then the attack is called as Idle User Attack. If the incoming flow has smaller than the lower limit and has bigger duration than the upper limit of duration, then the attack is called as Long Lived Low Rate Attack. (Buragohain and Medhi, 2016) states that for the other possible areas excluding the mentioned categories, the attack can not be detected in real time. For the mitigation process, the application uses a malicious user list to contain the user having malicious behavior and a malicious counter variable to count

how many times a specific user behaves malicious activities based on the categorization. Each user has a limited access count to the system. Thus, if the malicious count variable is greater than legitimate access count, then the source address is blocked by setting the hard timeout variable as maximum and dropped from the malicious user list.

## **2.4. Easy Defence Mechanism Against Botnet Based DDoS Flooding Attacks**

This study proposes a source-based defence mechanism against Botnet Based DDoS attacks. The mechanism depends on the statistical inference model. The sample flows are obtained from the sFlow-RT collector tool of InMon. The mechanism mainly aims at combining the power of SDN and the traffic collector. A detection algorithm and a response scheme proposed by (Lu and Wang, 2016) to detect the attack in source and mitigate the attack as soon as possible. In order to detect the botnet based attacks, (Lu and Wang, 2016) emphasis on defining a metric to measure the botnet based attack specifications. This metric is defined as a traffic tuple which is a two dimension metric called distribution and collaboration degree (DCD) of destination flow. A destination flow (d-flow  $j$ ) is considered as all the packets having the same destination IP address. The first dimension  $\delta$  quantifies a degree referring to how many nodes sending request to a destination. The other dimension  $\rho$  quantifies the intensity of a flow in another dimension.  $(\delta, \rho)$  tuple is represented on the x-y coordinate. The condition of sending packets to a destination for normal traffic or attack traffic could be defined as a random process and a discrete random function is treated as Poisson Process for a certain time interval. For a specific time  $t$ , the probability of  $k$  users together sending requests to a destination is defined as a Poisson function. Threshold  $\delta$  is determined by the Poisson distribution function. There is another threshold value for  $\rho$  variable. The threshold  $\rho$  is decided according to the maximum exit bandwidth of a certain node. After threshold values are determined, x-y coordinate plane is separated into the category A, B, C and D respectively as indicated in Figure 2.1. If the the point  $(\delta_j, \rho_j)$  of d-flow  $j$  falls into the area A meaning that the d-flow  $j$  is a normal flow due to the fact that  $\delta_j$  and  $\rho_j$  is lower than threshold values. If the point  $(\delta_j, \rho_j)$  of d-flow  $j$  falls into the B meaning that the d-flow  $j$  is normal, however, high speed Internet service is accessed by the users and the bandwidth is used a little bit more like downloading films or files having large sizes (Lu and Wang, 2016). If the the

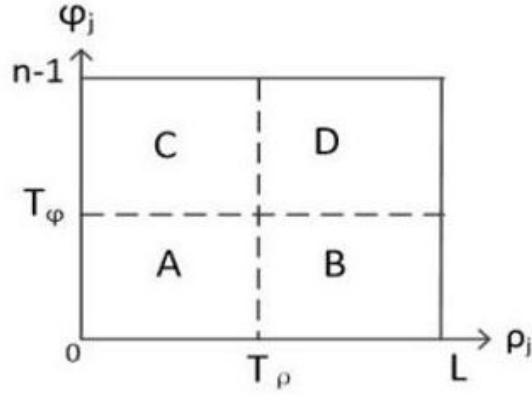


Figure 2.1. Distribution Collaboration Two Dimension Plot (Source: Lu and Wang, 2016)

point  $(\delta_j, \rho_j)$  of d-flow  $j$  falls into area C and D, these flows are considered as an attack flow and these flows are dropped and moved to the blocking queue for the mitigation. For the attack flows, sliding window queue response is taken and the queue size is controlled by a function to force the situation back to the area A or B. All drop actions and queue actions are installed on the corresponding switches on the SDN network according to the response scheme. This study includes a modular application communicating both Floodlight SDN controller and sFlow-RT collector by using REST APIs of both controller and traffic collector. Each module of the application performs the task itself. The application consists of 6 module which are topology, collector, detection, response, log and update. Topology gets the topology information from the SDN controller, and pushes the topology information to the collector module. Collector schedules the sFlow agents on each switch to get the traffic information from each switch on the network. After getting the topology information and traffic data from the network, the detection module categorizes the traffic according to the detection algorithm. Then, response scheme is executed in response module to take drop actions and install flow entries on the switches. For every second, topology and traffic information is updated and recalculated. Hence, the historical data is hold on the log module and the parameters related to the detection method is updated according to the historical values. For this thesis, we have used the detection algorithm of this approach in terms of determining the traffic detection parameters to be able to perform the detection mechanism with more network elements as specified on the future study of this research.

# CHAPTER 3

## BACKGROUND

In this chapter, SDN, its architecture, OpenFlow Protocol, OpenFlow Switch and OpenFlow messages used by the controller and network elements of the SDN environment are explained. Moreover, security concerns on the SDN environment are also explained in detail.

### 3.1. Software Defined Network

With the advance of technological improvements, SDN has emerged as a new networking paradigm for cloud services, social networks and modern networking environments. It is proposed as a future communication network architecture. Some increasing requirements and specifications in the traditional network architecture such as frequent change of bandwidth, adaptation difficulties of the topology and routing information change and the inadequacy of storage and capability has played as a main role for the existence of the SDN. SDN decouples the control plane and data plane where a logically centralized controller performs forwarding decisions on behalf of switches. Switches in data plane performs just forwarding process decided by the controller. A logically centralized software is used for the management of the whole network. In SDN, there are also vulnerabilities unique to SDN in addition to the existing one. Most of the security issues stem from the communication bottleneck between control plane and data plane. The followings are the major definitions related to SDN architecture.

**Data Plane:** It represents hardware and infrastructural elements such as switches and routers. The whole forwarded data such as packets are also represented with the data plane concept.

**Control Plane:** It represents all logic and devices that would make forwarding decisions. Control instructions are sent from control plane to data plane via protocols like OpenFlow. It interacts with infrastructure layer via OpenFlow protocol. Controller programs the forwarding rules for network devices. There is one or more controller in control plane that provides a global view of the network.



**Traditional Networks vs SDN:** SDN differs from the traditional network because in traditional networks, switches and routers perform forwarding decisions themselves by using routing protocols. While in traditional networks network elements are not aware of the global view of the network, in SDN environment controller is knowledgeable about the global network. In traditional networks, forwarding decisions are IP address based whereas in SDN, decisions are taken according to flows. The separation of control plane and data plane overcomes many security issues in traditional networks even though new vulnerabilities unique to SDN appear.

**OpenFlow Protocol:** It is the most popular communication protocol in SDN networking. It allows the communication between infrastructure layer and control layer and provides the programmability of devices. OpenFlow controller installs the forwarding rules proactively or reactively for network devices. These devices make header matching with associated patterns. Controller performs all forwarding rules and switches only make forwarding with predefined actions. It enables an external entity like controller to the manipulation of the network flow packets (Mousavi, 2014). OpenFlow enabled switches have the flow tables representing the ingress and egress paths of a packet for a specific switch. OpenFlow protocol enables controller to access flow tables. This access is provided over a secure communication channel.

**Infrastructure Layer:** In infrastructure layer, there are network devices such as hubs, bridges, routers and switches. There are network resources in infrastructure layer. These resources are stored in RDB. RDB is considered as a storage of the whole resources. Agents in infrastructure layer are responsible for executing SDN controller instructions. Coordinator is dependent on the OSS. OSS allocates resources for network elements by means of coordinator and it determines policies for the management of these elements. Network devices are instructed by the SDN Controller by Southbound Interface. According to these instructions, forwarding actions are performed.

**Control Layer:** It contains core logical programming of packet forwarding, network switching and routing. The core instructions and decision-based information are sent to the data plane of the network devices. The control layer has Control Layer Agents which are responsible for connecting the Application Layer and Control Layer with some programmable APIs.

**Application Layer:** It consists of various applications like IPS, IDS, security systems, traffic monitoring services, load balancing, bandwidth management, quality of service etc. It also contains access applications being able to access network devices.

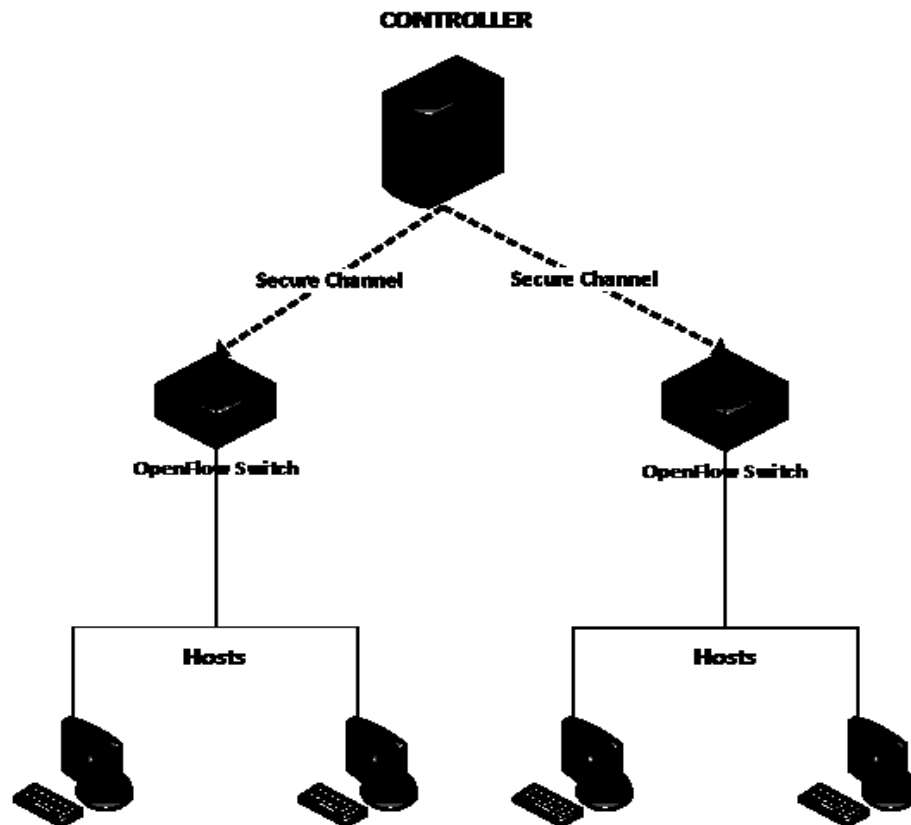


Figure 3.1. A Simple SDN Network with OpenFlow Switches

Figure 3.1 exemplifies a simple SDN network. In SDN architecture, network devices are distributed on the data plane. These devices are managed by the controller. Due that reason, controller gains a complete visibility on the data plane elements by utilizing some network functions and the APIs. When a particular packet comes into the switch, initially flow table is controlled for the flow rule whether the specific packet has reached previously or not. If there is a match on the flow table, the switch will process the flow rule. Otherwise, the incoming packet headers are forwarded to the controller. The controller begins looking for the flow entries, in condition that the controller does not discover the flow rule inside the SDN network, the packets are accepted as unfamiliar packets, and are dropped. If the controller finds the matching fields, then it will send the flow rules to the switches and then install the flow rules on the corresponding switches. After that, the specified switch executes the actions in the flow rule according to the path between the source and the destination given in the flow table. Figure 3.2 illustrates the flow processing in the OpenFlow protocol.

**Flow:** A flow is a group of packets going from one point to another point on the managed

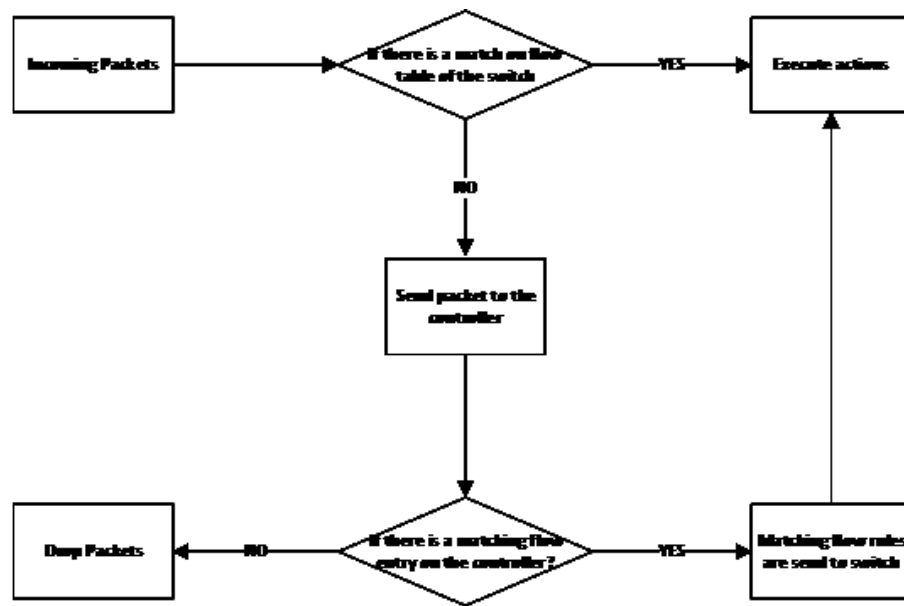


Figure 3.2. Flow Processing in OpenFlow Protocol (Source: Murtuza and Asawa, 2018)

networks and having a temporary lifetime. The packets inside the flow should have the same matching structure within its lifetime. A flow carries on the statistical parameters like received bytes, received packets and flow duration.

**Received Bytes:** It represents how many bytes received with a specific flow.

**Received Packets:** It represents how many packets fitting the same matching structure of the flow entry on the flow table of the switch.

**Flow Duration:** It represents how much time a flow exists in a flow table. Flow duration is controlled by the *idle\_timeout* and *hard\_timeout* fields. These parameters are set when a flow entry is installed on the flow table. These parameters do not change the modification of the flow entries. During this interval time, if there is no observation for the traffic related to the corresponding flow entry, *idle\_timeout* will expire. The purpose of setting *hard\_timeout* is to set a bound for the lifetime of the flow.

### 3.2. OpenFlow Switch

OpenFlow switch contains one or more flow tables and a secure communication channel with controller. OpenFlow protocol performs the secure communication over TLS enabled channel. Each table composed of flow entries having a match field, some

counter information and a group of instructions. The basic OpenFlow switch consists of 10 tuple for matching operation. Table 3.1 illustrates packet header fields on the flow tables. For each table, there is a metadata field as a register information to transfer header information from one table to another. When the packet is searched for matching, the metadata field is used to packet transfer between the multiple tables. If a match is found, counter statistic information will be updated and the corresponding action is assigned to the related entry. Counter statistic information refers to the current numerical value of real time variables like per flow entry, per table, per port and per queue (Mousavi, 2014). These parameters are shown in the Table 3.3. The actions can be look up for in other tables, forwarding packet or drop packets, rewriting of the header fields according to the OpenFlow specifications which is published by ONF. Supported actions are given in the Table 3.2. For this thesis research, OpenFlow version 1.3.0 is used for the testing purposes.

Table 3.1. Packet Header Fields on Flow Tables

Switch Port	Mac Src	Mac Dest	Eth Type	VlanId	IP Src	IP Dest	IP Port	TCP sport	TCP dport
-------------	---------	----------	----------	--------	--------	---------	---------	-----------	-----------

Table 3.2. Actions of OpenFlow Switch

Action	Description
Output	Output to switch port
Set VLAN VID	Set the IEEE802.1q VLAN ID
Set VLAN PCP	Set IEEE802.1q header
Strip VLAN	Strip the IEEE802.1Q header
Set Ethernet source address	Set Ethernet source address
Set Ethernet destination address	Set Ethernet destination address
Set IP source address	Set IP source address
Set IP destination address	Set IP destination address
Set IP ToS	Set IP Type of Service (ToS) service
Set TCP/UDP source port	Set TCP/UDP source port
Set TCP/UDP destination port	Set TCP/UDP destination port
Enqueue	Output packet to a queue

Table 3.3. Counter Statistic Information

Counter Statistics	Parameters
Per-table	Active Entries,Packet lookups,Packet matches
Per-flow	Received packets, Received bytes, Duration (seconds),Duration (nanoseconds)
Per-port	Received packets, Transmitted packets, Received bytes, Transmitted bytes, Receive drop, Transmit drops, Receive errors, Transmit errors, Receive frame alignment errors, Received overrun errors, Receive CRC errors, Collisions
Per-queue	Transmit packets, Transmit bytes, Transmit overrun errors

While sending the packet to the controller, OFPT\_PACKET\_IN message is created by the OpenFlow switch in order to report the arrival of the packet to the controller. The controller should give response for the incoming message to the corresponding switch. The packet is processed on the controller. Then the controller replies OFPT\_PACKET\_IN message with OFPT\_PACKET\_OUT by setting required actions for the incoming packet. These actions can be one of the mentioned actions previously.

### 3.2.1. OpenFlow Messages

OpenFlow messages can be categorized into three message types which are asynchronous messages, symmetric messages and controller-to-switch messages.

**Controller-To-Switch Messages:** Controller-to-switch messages are initiated by the controller and used to configure the switch and query its state in order to get statistical information from the switch and manage the flow table of the switches. The functionality of OFPT\_PACKET\_OUT messages is to set the packets generated the controller to the data plane. The purpose is to install flow entries directed by the controller. Flow Mod message is another controller-to-switch messages sent from the controller to switch to be able to add, delete, modify the flow entries on the switch (Murtuza and Asawa, 2018).

**Asynchronous Messages:** Asynchronous messages are the messages that sent from the switch to the controller to be able to inform the controller about the network state. One example of these messages is OFPT\_PACKET\_IN messages. When a new packet comes to the switch, if there is no matching on the flow table for incoming packets, it is necessary for OFPT\_PACKET\_IN message to be sent to the controller. OFPT\_PACKET\_IN message can contain the entire parcel or just a piece of the got packet. As a response to

this message, the controller responds with the OFPT\_PACKET\_OUT message.

**Symmetric Messages:** Symmetric messages are used to ensure the activity of communication between data plane and the control plane. These messages can be sent from both controller and switch. Echo request and hello messages can be given example for these messages.

### 3.3. SDN Architecture

Software Defined Networking is a new network architecture emerging with the needs in modern networking environments such as data centers, social networks and cloud services. It is designed to overcome the network requirements based on the frequent changes of bandwidth, topology and routing information, storage and scalability needs of these environments and enables network administrator to manage the network environment centrally via programmatically configured controller. The main purpose is to separate the control plane (traffic control) from the data plane (network hardware) in order to provide flexibility of network components and facilitate the management.

SDN is closely associated with OpenFlow application which is one of the most well-known protocol and an interface between data layer and control layer providing access to network devices such as routers and switches and it is a communication protocol to interact with data plane of network devices by using the functionality of the controller. OpenFlow protocol is basically used to analyze the traffic flow and communicating with the hardware to forward the traffic to an appropriate direction. SDN architecture is composed of different partitions with regard to be shared various tasks. These partitions are indicated in Figure 3.3.

**Southbound API:** Southbound API can be considered as a bridge between infrastructure layer and control layer. It is a way to communicate network elements with controller. OpenFlow protocol is one example of Southbound section.

**Northbound API:** Northbound API is another interface that provides communication between the controller and applications or services running over the network. REST API can be considered as one example of that interface which does not offer an optimal and secure method due to the lack of the management of authorization. On the other hand, Northbound API must support the various applications, therefore it is a flexible and transparent component in terms of compatibility of different services.

**Eastbound and Westbound API:** These are used for the same purpose of managing the distributed SDN architecture. In a distributed architecture, it is necessary for different instances of the controllers to communicate and have management and control information. There are different type of a distributed SDN architecture. One type of architecture is the vertical or hierarchical architecture including the main controller being able to have different low-level controller which is responsible for management, control and monitoring. The other type includes controllers which may have different functionalities.

The most important purpose of SDN architecture is to obtain an open standard based, decoupled, dynamic, centrally managed, adaptable, directly programmable and flow based manageable, and cost effective structure.

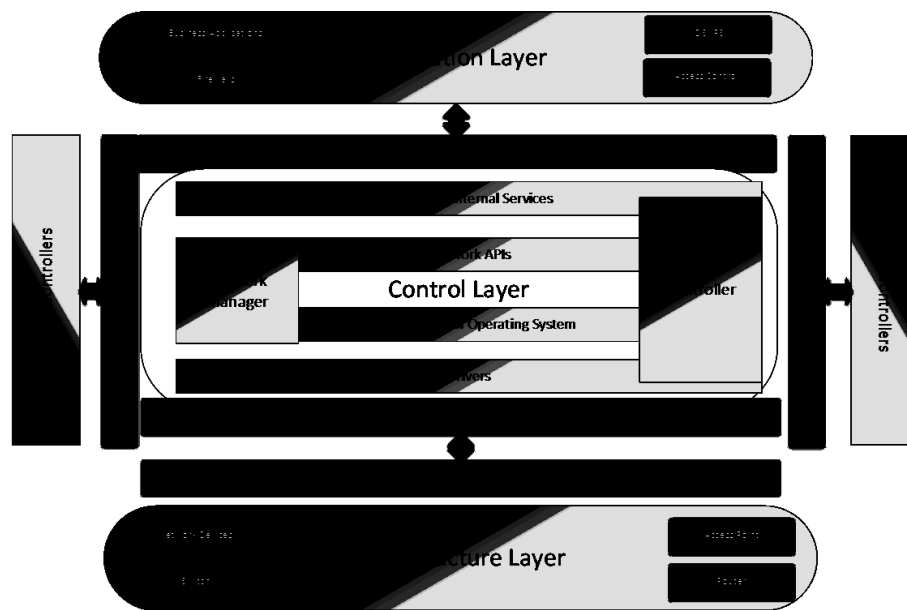


Figure 3.3. SDN Architecture

**Dynamic:** While hardware configurations and reconfigurations are difficult to implement, the software can be programmed in order to keep pace with the sudden changes dynamically for necessities of modern applications and devices.

**Open Standard Based:** In SDN architecture, many developers can develop applications called middleboxes, these applications provide communicating with the controller and network switches. In that condition, controller platforms are considered as open source.

**Centrally Managed:** With this characteristic of SDN infrastructure, network elements are managed in one central point which is controller. In this design logic, the effects of dynamically changed network traffic and network elements are controlled.

**Decoupled:** It means that separating the data plane from the control plane or physical network is separated from the virtual network.

**Directly Programmable:** It is possible for the controller to be programmed by user level applications or middleboxes. This programmability gives rise to be accessed to the controller by middleboxes. By this way, network administrators can easily audit the network.

**Flow Based Management:** Flow-based management is to decide the direction of the traffic depending on each flow whereas IP based management concentrates on the IP address of each host. Flow table rules in switches or firewalls reside in each flow. These flows affect the whole network. In flow-based architectures, there are different kinds of messages used for communication between switches and controller such as OpenFlow messages, controller to switch messages, switch to controller notification messages, flow statistics, datapath flows and symmetric messages. The symmetric messages are used with regard to performing the information exchange in the traffic between controller and switches and to be decided by extracting the incoming packets according to flow table rules by controller and switches. In addition to normal switches, there are also virtual switches that process the traffic between Network Interface Card (NIC) and VMs . Because of the processing of that traffic by the virtual switches, VMs (Virtual Machines) are able to treat as if they are real hosts. A central controller works with one of the virtual switches with respect to central and logical management. SDN switches are collectively managed by the central controller instead of performing local control of switches. However, some of the switches may lose communication with the central controller. In that condition, switches may be set to different modes such as interactive mode and passive mode.

### 3.4. Security Threats

As technological advances occur, new issues like security issues of the new technologies start to come into existence. SDN can be evaluated as one example of that condition. SDN typically depends on both software and hardware, thus there are so many vulnerabilities and attacks on SDN architecture. Due that reason, the security of SDN also should be controlled. Although the vulnerabilities of SDN exist, there also exist some sort of security control applications across these vulnerabilities such as IDS/IPS, firewalls, policy management, auditing and access control so as to deal with these issues of SDN architecture. Security threats may concentrate on both different assets and



resources of SDN architecture and also the various components of the architecture. To illustrate, while some attacks can be focused on switches' flow tables including some significant information such as routing and management information, some attacks can be centralized to the controller which is the core of the management and control.

### 3.4.1. Spoofing

Spoofing is a general attack type that the attacker uses the fake network information by hiding the real network information such as IP address and MAC address and ARP tables consciously. Spoofing may be used to perform DDoS attacks by using fake addresses in a zombie network.

**ARP Spoofing:** ARP Spoofing attack is one of the spoofing attacks where the attacker's MAC address is assigned to a valid IP address. The traffic that the original receiver has is captured by the attacker and the receiver is broken out of the network. In order to determine the ARP Spoofing attack, IP to MAC mapping table can be tracked (Alsmadi and Xu, 2015). A prevention method used against ARP Spoofing attack is ARM module which resides in the controller. Only MAC addresses of authorized users or hosts are traced with that module. After that, the controller looks up to this ARM module and eliminates all of the ARP responses which are not confirmed by this module. ARP poisoning is another issue that can happen between controller and switches if the SSL encryption is not used. ARP cache poisoning implies that the intruder is placed with the victim to the same subnet and by that way, the ARP tables of other components are poisoned by the forged information. In that situation, the attacker might use a scanner in order to listen to the traffic. In 2014, Al Shabibi has developed an anti-ARP cache poisoning switch application in the controller to bring a solution for such attacks (Alsmadi and Xu, 2015). When assessed the overall attack detection methods, these methods are separated into two as low-resolution methods and high-resolution methods. Low-resolution methods are inspected from the flow whereas the high-resolution methods are analyzed at the packet level. In order to overcome the low-level attacks it is necessary to be known the information detail of the flows. On the other hand, high-level resolution attacks might be solved by the detail of the information at the packet level. That is to say, ARP spoofing and cache poisoning attacks can be analyzed in packet level in order to be able to detect or take precautions because of the fact that it is one of the high-level resolution attack.

**IP Spoofing:** IP Spoofing is generally implemented inside the other security attacks such as DNS tampering or amplification. IP Spoofing aims at changing the attackers' IP address to a fake IP address in order to hide the information belonging to the sender and introduce the attacker to the other systems with a spoofed IP address by generating a legitimate Internet Protocol packets. In DNS tampering attacks, IP Spoofing can be used to reroute the traffic to illegitimate websites while the attacker is tampering the DNS directory. It can also be part of other security attacks. All spoofing attacks purpose to the redirection of the traffic to illegal hosts. One way of preventing the spoofing attacks is to use strong authentication methods which are important for countering the unauthenticated intrusion. The other way for countering IP spoofing is to use the IP Address Validation methods. Source Address Validation Improvement modeled by the Internet Engineering Task Force confirms the addresses of packets to determine whether the packets have a valid binding (Alsmadi and Xu, 2015). There is another module called Virtual Address Validation Edge extended from Source Address Validation Improvement. This module existing in the controller confirms the address of the outgoing packets not having any record in the flow table. There is a decision mechanism for the flows depending on the validation module and a dataset of white-lists to be allowed or dropped. Flows are analyzed according to the validation module and a dataset of white-lists and then are allowed or dropped. Another approach is to extend the Virtual Address Validation Edge module features by using OpenRouter (Alsmadi and Xu, 2015). In this scenario, each router realizes the assignment and routing information of the whole network. Software Defined Filtering Architecture is another countermeasure for IP based or router based spoofing attacks. Flow rules are constructed, collected and added if the spoofing is detected. The nature of networking information is dynamic because of its frequent changing characteristics. This information can be exemplified such as IP Address, MAC Address, routing, and topology information. An additional approach is offered as a moving target defense approach by (Al-Shaer et al., 2012). The position of each host should always be modified in order to be guarded against the attacker's fury.

### **3.4.2. Tampering**

Tampering is intentionally unauthorized access to the network information in order to destruct the information. To illustrate that in this type of access, flow rules can

be changed by the attacker and this can bring about network misbehavior. An attacker can determine a flow rule which refuses the legal hosts and accepts the illegitimate hosts. The other example is that an attacker may hijack the traffic from one point to another direction. The most significant point in SDN architecture is the security of the channel between different controllers. In 2012, a new security issue was defined as the issue of the dynamic flow tunneling depending on the conflicts during the interpretation of the rules by (Porras et al., 2012a). Firewall or flow table rules might seem to be a goal of security threats. While the individual flow does not break the firewall rules, a set of rules can be executed collectively by the intruder and then this execution can lead to the violation of the rules. Those conflicts between the flows and firewall rules depending on the integration of the incoming flows were controlled by (Porras et al., 2012a). For the prevention of unintended tampering at a flow level, SDN can be useful to obstruct that kind of attack. Tampering can be degraded by auditing and monitoring at some point of the network. If the attack is across a determined point, the rest of the network is evidence with regard to the detection of tampering. For the protection from the tampering attack, the control of encryption methods and legitimate connections should be managed. In virtual environments, in addition to external tampering, there is also internal modification of network information which affects the level of correctness and integrity. Existing researches indicate that different VMs in the same field can access the resources from each other due to the fact that these machines use common physical resources.

### **3.4.3. Repudiation and Non-repudiation**

Repudiation refers to the negation of the actions of one entity residing in a communication. There is another concept which is called as non-repudiation implying that if one entity refuses the actions which is performed, it is necessary to be proven that the considered actions are performed by this entity. In a typical communication between two parties, if the sender sends a packet to the receiver, the receiver wants to confirm that packets have come from the sender. For the verification of non-repudiation, encryption methods such as TLS and SSL are used to communicate in a secure channel and to be authenticated the actual sender and receiver. PKE and digital certificates can be a solution to the repudiation. Signal chaining can be another solution to provide non-repudiation due that reason an audit system should contain the sequence and route steps so that it

can be understood that a packet only can go over the channel between the sender and receiver. The other verification method for non-repudiation is the auditing and logging methods. All activities can be observed and traced from the flow tables. Those methods can be accepted as a proof for traffic activities. However, it is essential to select what exactly audits inasmuch as continuously auditing gives rise to some performance issues. Logging and auditing methods may also be objected to security attacks. A flow-based authentication system called as Fort-Nox offered by (Porras et al., 2012b) is designed to handle non-repudiation verification. It is an auditing tool that uses some information such as application Id, privilege level, flow time and date which is required to the investigation of the instant events if any unusual condition occurs. In order to identify the activity of the sender, the Accountable Internet Protocol is suggested as a new protocol to replace with Internet Protocol (IP) which is offered by (Andersen et al., 2008) in 2008. The main purpose is to bring additional information to existing information with regard to the identification of the sender. Additional location-based identification of hosts is offered in addition to IP address related to Public Key Encryption of the host.

#### **3.4.4. Accountability**

Accountability refers to the responsibility of each controller from its switches in SDN architecture. It is not allowed for different controller residing in a different domain to exchange data. Even though local controllers drop the packets belonging to the switches in other controllers' networks, there is necessity of communication of different controllers via well-defined interfaces in terms of not obstructing to each other and preventing some security problems. Identifying applications are also important because the several security controls such as proxy system and NAT restrict the identity of hosts. Accordingly, firewalls cannot detect the traffic source. In order to overcome this issue, Flow Tags depending on the flow information is suggested to identify the applications. Applications add the flow tag information and FlowTag module proposed by (Seyed et al., 2013) recognizes the packet header information including the flow tag in order to determine where the packet actually comes from.

### 3.4.5. Information Disclosure

Information disclosure attacks does not aim at directly corrupting the network resources, however, the desire to observe these resources can be accepted as the main goal of this type of attack. An attacker attributes to sniff the network resources to seize the communication details of hosts. Network switches are controlled by the controller application. When captured the controller application, it is possible for the attacker to have access more than it should be. In traditional networks, data is integrated with the control whereas in SDN architecture data is separated from the controller, switches' flow table carries the flow information. In that condition, if an attacker accesses those switches via the controller, the traffic may be directed to illegitimate directions by modifying the flow tables. At this point, it is also possible for the attacker to participate inside the network as an observer. Man in the Middle Attack is an example of information disclosure attack that aims at the data inside the channel. OpenFlow architecture can be exposed to this type of attacks thanks to the applications on the Northbound section having possible vulnerabilities.

**Scanning Countermeasures:** Scanning methods are used to detect the vulnerabilities or weaknesses stemming from the information disclosure attacks in the network. The most considerable way to resist scanning attack is the encryption methods which can be exemplified as SSL/TLS encryption methods. Active security methods may also be used for the detection of scanners. Flow information is used for the traffic anomaly detection such as the detection of the worms. Due that reason, the suspicious traffic is extracted initially and then analyzed with the help of OpenFlow based detection system. To prevent from OpenFlow network scanning, some methods have been developed. While some focus on responding the scanner with incorrect traffic, others perpetually alter the hosts' identification information. NAT, VPN , proxy systems conceal the identification of hosts even if the actual purpose does not become like that. For that problem, an OpenFlow anonymization service called as AnonyFlow is developed by (Mendonca et al., 2012). When the translation becomes between the virtual address and actual IP addresses, special anonymity IDs that are visible to other hosts are used instead of IP Addresses which are visible to other hosts.

**Information Disclosure Countermeasures:** The protection of private information requires some actions such as whitelisting and blacklisting which are used for filtering the traffic. In traditional networks, filtering is done according to the IP and MAC Addresses.

However, this filtering scheme which can hinder some attacks may also be used for OpenFlow networks. In order to decrease the effect of information disclosure attack in the OpenFlow network, a significant suggestion is articulated by (Kloeti, 2012). This suggestion depends on intelligent rules providing time out randomization. These rules generate some obstacles for the sniffer to the disclosure of network patterns. To illustrate, the difference in response time is so valuable information for OpenFlow networks that fake response time can be comprehended to take action to this type of attack.

### **3.4.6. Denial of Service Attacks (DoS)**

DoS Attacks are the most dangerous attacks targeting to decrease the network performance and eliminate the legal packets. This type of attacks can lead to the suspension of the whole network or stop the operations on the network. In OpenFlow networks, this attack can also be applied to interrupt the network activities between controller and switches having a continuous communication. This interruption can be achieved by injecting flow to this channel. The required information for the detection of such attack is the flow level information. Flow based attacks such as DoS, worms, botnets and scans are generally based on the flow header. The common characteristics of these attacks are to have a large and unbalanced traffic. Additionally, these attacks aims at using several widely used port numbers. These widely used ports can also be used to detect the DoS attacks. Extracted information from the flow headers are so useful for Dos detection. The most suitable way to detect DoS and flooding attacks are to use the methods being able to determine the large traffic. One of the methods concentrates on the volume change between incoming and outgoing traffic. If the traffic goes from one direction to the other and contains multiple response packets, this can be considered as a flooding type. In order to occupy one direction multiple SYN packets can be sent to the receiver, in that case periodically sent multiple ACK packets can be generated by the receiver when the sender does not care. DNS Amplification Attack is a type of DDoS attack depending on the use of reachable DNS Servers. Attacker use these servers to flood a target system with DNS response traffic. This attack results in the Internet corruption. In order to overcome such attacks, monitoring and tracking recursive DNS queries can be used to detect DNS Amplification. Another risky problem is the loop occurrence in the network leading the packets not to reach their destination. This condition is another reason for DoS attacks.

In SDN architecture, a large amount of traffic leads to the change of the flow attributes. Hence, traffic generators can be used to modify the attribute values. There are two goals for DoS attack in SDN. One of the purpose is to flood the switch flow tables and then reorganize these tables with illegal rules. The second purpose is to provide the controller to stay busy by directing a large traffic. Strong and reliable encryption methods provides a secure communication between switches and controller. However, there is still a problem with DoS and flooding because of being sent the large traffic to OpenFlow networks. Due that reason, Avant-Guard Systems are developed by (Shin et al., 2013) to overcome DoS attacks and get rid of their negative effects.

**Detection of Denial of Service Attacks:** The detection of DDoS attacks depends on the several metrics such as Average Number of Packets in per flow, Average of Bytes per flow, Average of Duration per flow, Percentage of Pair-flows and Growth of Single flows (Alsmadi and Xu, 2015). The monitoring of these metrics gives rise to the degradation of the performance of the controller. The controller performance issue is solved by adding an extra controller working for the large amount of data arising from the monitoring process. The most considerable method to detect such attack is to determine the volume of traffic. This volume can be compared with the threshold value to be able to comment on the size of the traffic. By this way, it would be possible to discover abnormal traffic on the network. When the threshold value is violated, a new rule is added to reject the packets by the controller. TCP connections can be targeted by the DoS attack or flooding attacks. In such a condition, ACK messages are the indicator of the existence of the communication between the sender and receiver. While some studies focus on an algorithm to manage TCP ACK packets, some others concentrate on a monitoring interface to inspect the communication between switches and controller. On the other hand, there also exists a DDoS detection system for the OpenFlow network. This system evaluates the metrics in terms of the possibility of the occurrence of DDoS attacks. However, it is possible to observe the false positive alarms from time to time.

**Countermeasures of DoS:** Effective and dynamic response methods can be used to counter DoS attacks. Monitoring and restricting the traffic by the controller are the basic countermeasure for DoS attacks. Protection mechanisms for DoS attacks require to instantly recover the negative effects of the flooding. Moreover, legitimate and illegitimate traffic should be separated from each other. Flow rules optimizations or rule-merging is another countermeasure method for DoS or flooding attacks. Flow tables can be targeted to flood with rules dynamically. However, buffers and switch memories can be exceeded.

This situation leads to the denial of service for legal traffic. Continuously, evaluating flow table rules should be a characteristic of switches needing to be added to have a dynamic ability.

### 3.4.7. Distributed Denial of Service Attacks

DDoS attacks have been recognized as an Internet network security problem by the network security research communities since the mid-1980s. It is one of the major threats exhausting server resources. DDoS attacks have been firstly recognized since the CIAC announced the first DDoS attempt in the summer of 1999 (Zargar et al., 2013). DDoS attack is an endeavor to make a machine or system asset inaccessible to the intended users. Most of the attacks purpose the victim resources unavailable, leading to revenue losses, increased costs of mitigating the attacks and repairing and restoring the devastating services. In order to lunch a DDoS attack, an attacker compromises a set of computers by using a set of well-known methods like back-doors, Trojan horses, etc. As indicated in Figure 3.4, after captured hundreds or thousands of computers, the attacker sends commands to handler computers to orchestrate the attack simultaneously. The handler sends simultaneous commands to the compromised hosts as usually known zombie network. Each bot generates traffic to the common destination point and the requests sent from the bots convert to a giant flow going on the same destination. A victim has some limited processing power and capacity to handle a limited amount of service requests. Due that reason, the victim can not accomplish giant flows, as a result, the service will be unavailable for the intended users.

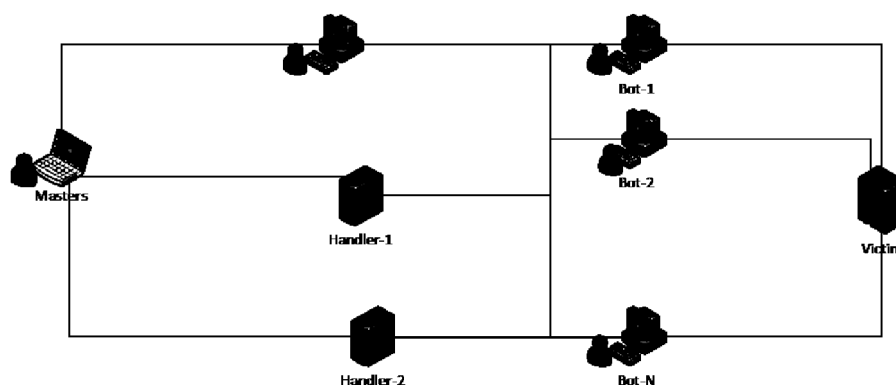


Figure 3.4. DDoS Attack Structure-2



DDoS attacks has been studied for two decades by the security researchers. This threat is not only an issue for traditional networks, but also it threatens the SDN network architecture in terms of several reasons such as bandwidth depletion and exhaustion of resources. There are also other methods to perform DDoS attacks like memcached based DDoS attack, however, in this study, we concentrated on the Botnet Based DDoS Attacks as the category of TCP/SYN Flood. To exemplify the experienced DDoS activities in reality, the first DDoS attack and the biggest ever attack is given below.

**First DDoS Attack in History:** The first DDoS attack in the history is performed in August 1999 to disable the University of Minnesota's network. A hacker used a tool called Trinoo composed of compromised machines called masters and daemons. An attacker send command to Daemons to send UDP flood. It was made no effort to hide the daemons IP addresses. Due that reason, attack owners could easily be diagnosed in this attack (Radware, 2017).

**Biggest Ever DDoS Attack:** Github has experienced with the biggest DDoS attack for the five minutes in the history of 28 February 2018. It peaked at the record 1.35 Tbps (Newman, 2018). There were no botnets in this attack, because it uses memcached based amplification approach. Amplification Based DDoS attack leverages the amplification effect of a popular database caching system known as memcached and works by sending a forged request to the targeted memcached server (Newman, 2018). This type of attacks are more dangerous than the botnet based attacks because of producing larger amount of traffic. The other biggest DDoS attack was targeted to bombarding the Dyn DNS servers by using Mirai botnets composed of IoT devices in 21 October 2016. Dyn is a DNS service provider company giving service to a wide range of popular companies like Netflix, Twitter, Paypal, Spotify, Amazon and HSBC etc. This attack is also considered as one of the biggest DDoS attack affecting its customers and peaked at 1.2 terabits per second according to the Akamai Research.

**Common Types of DDoS:** Despite of the fact that the categorization of DDoS attacks can be analyzed comprehensively in different ways, according to OSI Layer, the most general categorization based on the three layers of OSI Layer which are Transport Layer, Network Layer and Application Layer. A wide range of DDoS attack types has been observed on the aforementioned layers for decades. When assessed the highest rate attack types on these layer, the most encountered attack types can be grouped into 3 which are UDP Flood, TCP/SYN Flood and ICMP Flood Attacks. The most popular flooding attack types are briefly explained below.

- **UDP Flood Attack:** UDP Flood Attack is known as Transport Layer attack. It is a type of DDoS attack that sends a huge amount of UDP packets to the victim node. Large packets are directed to random ports on the destination node. The randomization causes the victim machine to look for the specific application on the random port numbers coming from the zombie network. As a response to large UDP packets, Destination Unreachable response packets are sent to the bots in condition that any application is not found on the specified ports of attack packets.
- **ICMP Flood Attack:** ICMP Flood Attack is known as a Network Layer attack. It is generally also known as a ping flood attack which is performed by sending a huge amount of ICMP Echo Request packets to the target machine. The packets generated by the zombie network are sent simultaneously, as a consequence, the target machine tries to reply to the incoming packets. However, the nature of simultaneity of the attack causes the victim machine to slow down and eventually, it will be unavailable.
- **TCP/SYN Flood Attack:** TCP/SYN Flood Attack is known as a Transport Layer DDoS attack. It exploits the weakness of three-way handshake process of TCP connection. In TCP connection, if a sender sends a SYN request to the receiver, the receiver should reply the packet with SYN/ACK response packets. When a large volume of SYN packets is sent to the victim by the zombie network, the victim node unnecessarily waits for ACK packets never responded by each compromised host of the zombie network. The process of waiting ACK packets by the victim leads to the use of resources by the victim machine. Consequently, the victim will be inaccessible by its intended users.

### 3.4.8. Elevation of Privilege

Elevation of Privilege attack aims at enhancing the access priority of the attacker to the resources on the system by gaining access permissions. The detection of the Elevation of Privilege attacks contains powerful and intelligent auditing methods. A system called Pedigree developed by (Ramachandran et al., 2009) is to track the executed applications with the help of the tagging information. The most considerable issue is the scalability of auditing and logging methods which give rise to the needs of huge amounts

of storage and memory. Privileges reside in Access Control Module. Escalation attacks aim at getting this information on these modules to modify the content by the attacker. Distributing permissions to hosts in a fair way is a major responsibility. With regard to handling this type of attack, RBAC system depending on the flow based authentication is offered to distribute the permissions in a flow basis instead of a user or host basis. This system neither always admits any user nor refuses, thus this reduces the privilege escalation problem. This system also provides the isolation of the controller from other flows. Internal flows and external flows are separated from each other. Switches have some rules that depend on the signature to confirm to be given permission, each flow is evaluated and if such signature is not included in the flow table, this flow deserves the lowest permission level. However, there is still an unusual condition even if the authentication failed, the least privilege approach allows the flows needed to be dropped ordinarily. Many attacks initially have the least privilege and the privilege of the victim is used to attack or disrupt. Privilege or permission system can be integrated into an individual module. This module can observe the applications in order to detect the sudden changes in the network. Some researches have been performed on the access control management systems having different access levels. Some applications include a few amounts of levels including administrator while some others can have a sufficient amount for the prevention of privilege escalation. Increasing number of access levels makes access of attacker difficult to have sufficient authority for performing such attacks.

## CHAPTER 4

### ENVIRONMENT REQUIREMENTS

This chapter covers the required applications for this thesis to perform the experiments on SDN.

#### 4.1. SDN Controllers Overview

A controller is the core of the SDN architecture. It is the central point of the SDN architecture because it handles the communication of applications and network devices in the control layer. There are many SDN controllers developed as the open source while some others are developed by the commercial fields. A lot of research on SDN controllers have been purposed on improving the controller characteristics such as availability, scalability, cost-effectiveness, and intelligence (Sakellaropoulou, 2017). To be able to work on SDN, it is necessary to research on SDN controllers. Nowadays, a large number of SDN controllers have been developed by special communities such as Linux Foundation and ONF. For example, the first SDN technology supporting the OpenFlow protocol is known as Nox controller developed by Nicira network in 2008. Pox is another SDN controller that appears as a sister project and advocates the python based SDN. Beacon is also another SDN Controller which is developed by David Erickson at Stanford University. It is a Java-based OpenFlow controller handles events and multithreaded operations. OpenMUL Controller is a lightweight SDN controller written in mostly C programming language and designed for tying the gap between virtual resources and physical resources and getting high performance and reliability. The main characteristic of OpenMUL is to facilitate deployment easier for mission-critical networks (Saikia and Malik, 2014). There are also commercially developed SDN controllers by different vendors such as Cisco, IBM , HP and Juniper. These vendors have preferred to give a perspective of their own ideas. However, open-source approaches are mostly used and researched by different communities. On April 8, 2013 OpenDayLight is proposed as an open source controller developed by Linux Foundation. It is improved on the Beacon design. It is the most deployed controller compared to others. The design criteria of OpenDayLight is to perform several use cases

such as operational efficiency, network programmability, Automated Service Delivery, NRO etc. Floodlight is also a popular OpenFlow controller initially implemented on the Beacon design. It is designed to be a platform for a wide variety of applications. The architecture of Floodlight has a modular structure based on the BNC (Sakellaropoulou, 2017). It is designed as a concurrent system to handle the throughput among the data centers and enterprise networks. Floodlight has a characteristic of running as a network plugin, this enables it to be integrated with OpenStack. This feature provides lots of researchers to dynamically visualize network resources.

## 4.2. ONOS

ONOS is a scalable SDN platform for service provider networks. It provides a control plane for SDN, it can manage network components and can communicate with clusters. The main characteristic of ONOS controller is that it is highly available and fault-tolerant inasmuch as multiple clusters can be on different machines. In condition that one of them is down, the other SDN controller takes over its responsibility and devices that it manages. ONOS also enables developers and community members to use CLI and GUI. ONOS consists of many modules and there are also lots of applications working with ONOS controller. These applications can be activated or deactivated depending on the requirements on the CLI and GUI. With the help of its southbound interface, it can communicate with network devices such as switches, routers, and hosts. ONOS can run on multiple servers in a distributed manner. ONOS services and its applications have been written in Java, it also depends on Apache Karaf OSGi container to allow module installation and bundling of multiple applications dynamically. ONOS is an open source project supported by a large number of companies that are a member of ONF. Many developers all around the world within the community try to perform different tasks and future targets.

ONOS platforms can be deployed by using different technologies and applications like Docker containers and Vagrant. It is also possible to deploy on physical machines. These physical machines can act as SDN controllers. In terms of machine behaviors, three definitions that are target machine, management machine and cluster are given below.

**Cluster:** Cluster is a set of target machines working together as distributed systems. By this way, multiple clusters has shared some roles to work in better scale, this improves

performance. Sharing of roles can be defined as load balancing.

**Target Machine:** Target machine is a single machine running on ONOS that is deployed physically, virtually or as a container.

**Management Machine:** Management machine is a central machine that can deploy ONOS on multiple target machines. This machine is generally used by operators. Thus, it is optional.

In this thesis, ONOS controller is used for the management of the network devices on the data plane. Installation processes of ONOS are fairly complex and difficult to follow the instructions. Thus, we have given the whole set of installation processes in the Appendix A.

### 4.3. Mininet

Mininet is a virtual realistic network emulator or orchestration system creating network elements on SDN environment. It runs a collection of end hosts, switches and links on a single Linux Kernel (Lantz et al., 2018). Thanks to Mininet, a single system can be converted to a complete network by its virtualization mechanism. A mininet host can act like a real host. It is possible to setup SSH connection between them and also uses any of the programs installed on the underlying Linux operating system. Each device created by the Mininet has different Ethernet interfaces as if they are real devices. Mininet virtual hosts, switches, links, and controllers can be considered as actual devices created by the software instead of hardware. This means that any topology created in real hardware network can also be created by Mininet on the emulation environment. Mininet provides a CLI to interact with network devices. It is an experimental application to experiment with OpenFlow and SDN systems with Mininet. To experiment with SDN, in this study, we used Mininet to create custom network topologies on the SDN controller. In Appendix B, we have given a detailed installation and command line usage examples of Mininet on the Linux operating system. It is highly required to carry out different attack scenarios in the SDN environment. Mininet also allows the researchers to construct their own topologies by creating custom topology files. After creating these files, it has a mechanism to reference a remote controller to create different topologies on the SDN controller. In addition to all of these features of Mininet, a simple linear and custom topology examples have been exemplified and explained in detail in Appendix B.

### 4.3.1. Mininet Advantages

In this section, a group of advantages of Mininet is briefly explained.

- **Fast:** A previously configured network can be started up with Mininet within a few seconds.
- **Custom Topologies:** It enables to create your own network by its extensible python API.
- **Run Real Program:** The system programs installed on the real machine can be used on generated virtual hosts.
- **Platform Independent:** Mininet can be run on a laptop, a server, in a virtual machine or in the cloud environment.
- **Easy to use:** It is possible to create and run experiments by Python script with a few command.
- **Open Source:** It is an open source project, it is possible to pull it from GitHub and analyze the codes of Mininet how it is implemented and contribute by fixing bugs or errors or share information by editing its documentation.
- **Customize Packet Forwarding:** Custom networks designs performed on Mininet can easily be transferred to a realistic hardware OpenFlow switches for line-rate packet forwarding.

### 4.4. s-Flow & s-Flow-RT

**sFlow:** s-Flow is an industry standard developed by InMon to monitor network flows going through the network. It gives a complete visibility of the packets against the security attacks. It is a Layer 2 technology identifying how much bandwidth used by which network element.

**s-Flow-RT:** s-Flow-RT is real-time and scalable platform and a traffic collector to provide real-time visibility to SDN. It is sampling technology to analyze and interpret the flows depending on time. s-Flow-RT analytics engine retrieves the continuous telemetry stream

from the sflow agents deployed in the network elements. The telemetry streams are converted to actionable metrics which is accessible from its RESTFlow APIs. It permits to retrieve network parameters from the devices and set threshold values for the abnormal traffic.

**s-Flow Agent:** sFlow agent is an application which is embedded within the network devices. Agents can communicate with the collector. Agents get the real time incoming and outgoing packets passing through themselves. The traffic information is sent from the network device to the traffic collector. In order to handle the network visibility, it is necessary to install sflow agents on the switches. In order to exemplify the installation of agents on a switch, we have given a detailed example in Appendix C.

#### **4.4.1. Interoperability of Applications**

After getting installed and running ONOS, Mininet and sFlow-RT, it is necessary to enable a group of applications of ONOS by using its GUI. These are the applications called as OpenFlow Provider Suite, Network Config Host Provider, Network Config Link Provider and Segment Routing (Phaal, 2018). Instead of manually installing and enabling sFlow agents, s-Flow-RT provides a mechanism to create agents automatically by using its sflow.py file ships with the application packets. In order to perform this process, it is required to use sflow.py file of s-Flow-RT application. On the other hand, Mininet can also be executed with sFlow application to create agents automatically while the network devices are creating. In order to exemplify the executing command of both of Mininet and s-Flow, we have given a command line example to perform this process in Appendix C. This mechanism exemplifies how Mininet, s-Flow-RT, and ONOS are working together and how agents send the packets to a central collector with the help of sflow.py file of the sFlow-RT application. A previously defined Mininet topology file can be executed with the sflow.py file. The desinged topology is created on the remote ONOS controller by the topology file. Moreover, the topology file sends the topology information to the ONOS controller whereas the sflow.py script enables sFlow monitoring of the network elements and send the topology to sFlow-RT collector. Besides, ONOS has a segment routing application using equal cost multi-path routing strategy, by that way the traffic might be directed to alternative paths on the generated topology.



## 4.4.2. RESTflow

RESTflow is a method of sFlow for transmitting flow records from OpenFlow switch through sFlow agents to sFlow collector.

**Flow Record:** A flow record is defined as a series of packets that share common attributes. A flow record contains a group of fields composed of ingress interface, source IP address, destination IP address, IP protocol, source TCP/UDP port, destination TCP/UDP port, IP ToS , start time, end time, packet count and byte.

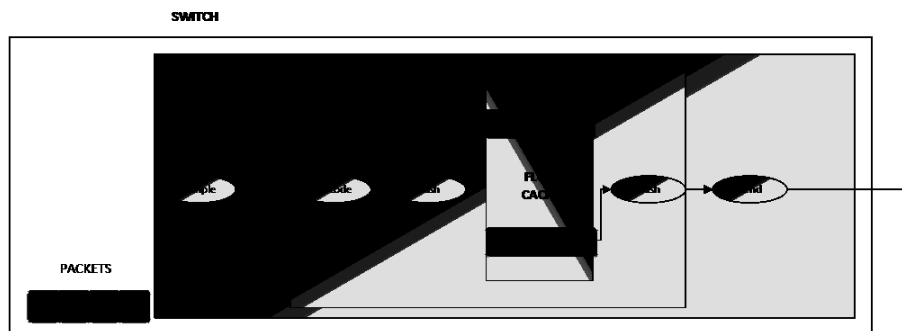


Figure 4.1. Flow Cache Embedded on the Switch (Source: Phaal, 2013)

Figure 4.1 illustrates the steps carried out by the switch for constructing flow records. Initially, packet streams come to the switch and each packet is sampled on the switch and packet header fields are decoded to extract the key fields. A hash function is executed on the key fields to search for a specific flow record in the flow cache (Phaal, 2013). If an existing flow appears in the switch for the incoming packet, its values are updated according to the new parameters coming with the incoming packet. Decoded and hashed records are flushed after processing of each sampled packet, then they are sent to the traffic analysis tool. As the switch operates on each incoming flow, a flow cache collects the whole packets coming from the switches.

Figure 4.2 illustrates the packet export and flow export mechanism. Thanks to sFlow monitoring, the decode, hash, and flush operations needing to be performed on the switch are not implemented on the switch (Phaal, 2013). Switches send the packets immediately to the collector and these operations are performed by the traffic analysis tool. Each switch periodically sends interface counter information to the traffic analysis application.

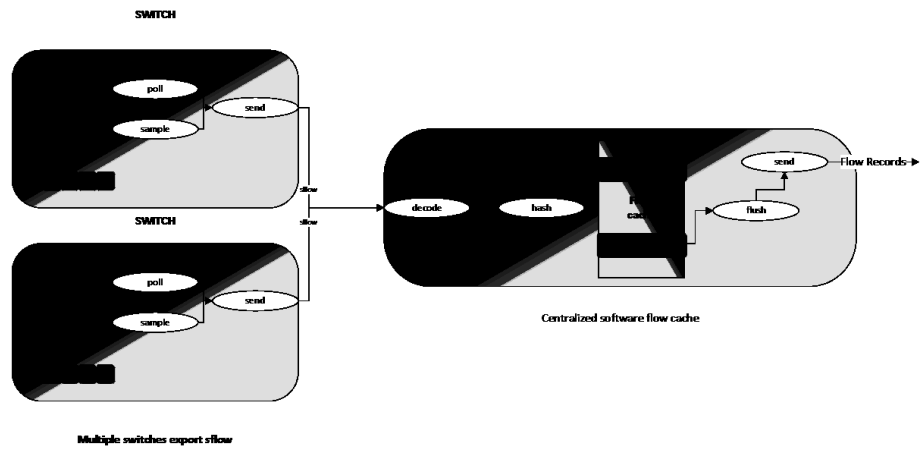


Figure 4.2. Packet Export and Flow Export Mechanism (Source: Phaal, 2013)

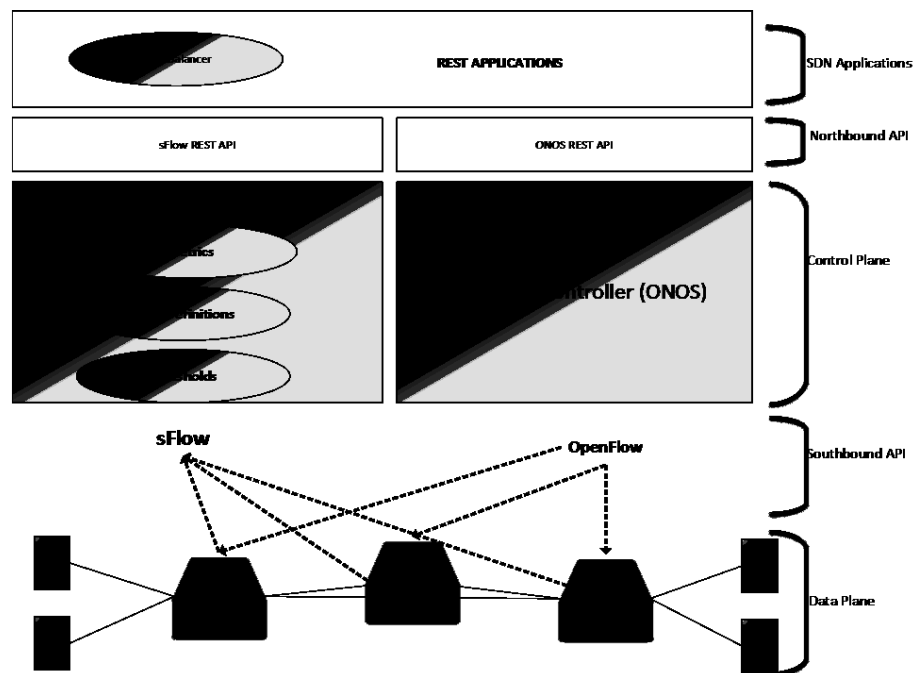


Figure 4.3. s-Flow-RT on SDN Architecture (Source: Phaal, 2013)

Figure 4.3 indicates that sFlow-RT resides in which layer on the SDN architecture and gives an overview to collect flow samples from data plane by the sflow protocol which is a southbound API. While OpenFlow controller provides a complete visibility on the network elements, sFlow protocol has also global view of the network devices on the data plane. The generated traffic on the network devices are collected by the sflow agents on the data plane and then forwarded to the sFlow-RT collector for the traffic analysis. sFlow-RT collector takes place at the control plane to collect and analyze the real-time traffic metrics and flows coming from agents. At the top of Figure 4.3, a wide range of security control applications can be run as an external applications by using the Northbound APIs of both ONOS and s-Flow-RT.

**Defining Flow Cache:** sFlow-RT analytics engine requires to define a flow cache to collect the packets from the data plane. For instance, different flow caches can be defined for different flows such as TCP flows, ICMP flows or UDP flows etc. By using s-Flow-RT RESTAPI, flow caches can be either defined programmatically or manually from the Linux terminal. During the execution time of sFlow-RT, s-Flow-RT caches allow obtaining the whole flow records generated on the network. In order to define flow cache, we have given a set of instructions for creating flow cache on sFlow-RT collector in Appendix C. Additionally, Appendix C also includes a content of a python file automatically defining a TCP cache on s-Flow-RT. The mentioned code pieces in Appendix C is used to get the whole real-time flow records inside the flow cache and it also illustrates the output of the execution.

#### **4.4.3. Network Wide Visibility of sFlow**

The most significant specification of sFlow is to monitor the entire networks composed of many switches and routers. sFlow agents are installed on every network element to transfer flows to traffic analysis tool. Enabling sflow agents on every access switch provides complete visibility of flows on the data plane. All ports on the network switches are listened by the flow analyzer. By this way, a large network on data centers can be controlled effectively. sFlow can also be implemented on both hardware and software switches belonging to different vendors (Phaal, 2009).

## 4.5. Bash Script Application

In order to automate and execute the multiple applications synchronously, we have written a bash script application to run local or distributed applications with one move. Application requires Guake terminal installed on the machine. Guake is top-down terminal for GNOME . It provides us to open multiple terminal windows and make the operations inside the windows separately and automatically. In order to experiment different kinds of functionality of the applications and generate different topologies on the ONOS controller, application can make user 10 different selections which are Distributed Network, SDN-IP Tutorial, SDN-IP Basic, Distributed SDN-IP, Individual Network, GRE-Distributed Network, ONOS Installation, Distributed sFlow Test for institute configuration and Distributed sFlow Test for home configuration and also Single SDN Domain for Large Network respectively. Figure 4.4 indicates an overview for the menu of the bash script application before executed. For the configuration of the bash script application, it is given in Appendix D.



```
MAIN - MENU
1. Distributed Network
2. Sdn Ip Tutorial
3. Sdn Ip Basic
4. Distributed Sdn-IP
5. Individual Network
6. GRE-Distributed Network
7. Onos Installation
8. Distributed sFlow Test(institute)
9. Distributed sFlow Test(home)
10. Single SDN Domain For Large Network
```

Figure 4.4. Bash Script Application

## 4.6. ONOS Cell Mechanism

Cell consists of a group of environment variables that are operated by the scripts shipping with ONOS. It can be used for the customization of packaging operations. Cell files make it easy to set ONOS environment variables by defining bash snippets inside

it. Additionally, before the ONOS controller is executed, it allows to be specified the required application needing to be activated with the execution of ONOS. With regard to illustrate a cell definition, Appendix D exemplifies how to create a cell definition and execute on the Linux environment.

## **4.7. Hping3**

hping3 is a network scanning tool which is known as a free packet generator and assembler for TCP/IP protocol. It is developed and documented by Salvatore Sanfilippo. It is a command line oriented TCP/IP packet generator. It does not only support ICMP echo requests, but also has ability to generate TCP, UDP, and IP protocol. It can work on different operating systems like Windows, Linux, MacOS etc. hping3 has been mostly used in network security field such as firewall testing, advanced port scanning, advanced traceroute operations of supported protocols, TCP/IP stack auditing, Remote OS fingerprinting, Path MTU discovery and network performance testing. hping3 has ability to generate random normal traffic for a network or perform security attack testing with different supported protocols.

### **4.7.1. TCP/IP Communication**

Before explaining how to generate normal traffic and performing DoS attack on the network by using hping3, it is required to examine the TCP/IP basics. TCP/IP communication basically depend on the connection establishment among the sender and receiver. The connection establishment is handled with the process known as three-way handshake. To illustrate, suppose that there is a client and a server on a network. Client and server wants to communicate with each other. This communication can be a web request sending to a server. In this case, the application layer protocol is HTTP protocol using the TCP communication on the Transport Layer. Another example may be given for the file transfer operation. A client can request a file from the server. In this case, the application layer protocol is FTP serving on a different port number. This communication also requires TCP connection on the transport layer. All examples given above requires three-way-handshake for connection establishment between the client and the server.

**Three-way-handshake:** Figure 4.5 illustrates the three-way handshake communication method of TCP protocol to establish a connection. Firstly, for different reasons like web page request, sending mail or file transfer, the client wants to communicate with the server. As a first step, the client sends a SYN packet to the server telling that the client wants to initiate a connection with the server. Then, the server responds the received SYN packet with SYN/ACK packet saying that the server is available to communicate with the client. After that, the client sends ACK packet to complete the connection establishment with the server. These steps are known as the three-way handshake of TCP protocol. After the completion of the connection establishment, the server can send the requested packets by the client. Due to the fact that the three-way-handshake method is used in TCP communication in Transport Layer, TCP protocol is accepted more reliable than UDP protocol because of the connection establishment.

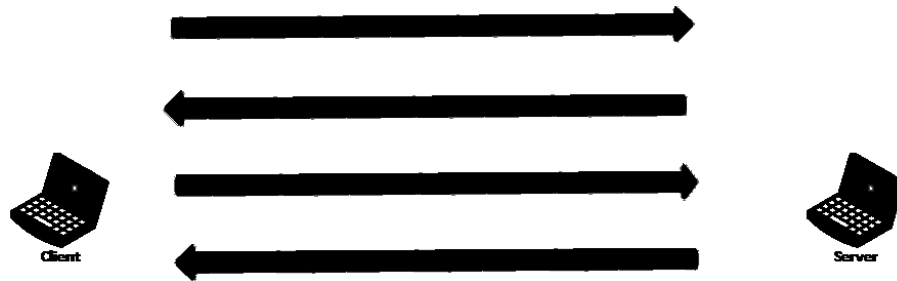


Figure 4.5. Three-way-handshake Process

## 4.7.2. Hping Arguments

This section lists a set of hping3 arguments and explains what they mean. This part includes mostly used specifications of hping3 instead of giving all of them. The basic arguments are demonstrated in Table 4.1.

### 4.7.2.1. Hping3 Modes

If no option is used, hping3 works on TCP mode. However, there are a group of different modes in hping3 for different usage as shown in the Table 4.2.

Table 4.1. Hping3 Basic Arguments (Source: Sanfilippo, 2006)

<b>Hping3 Options</b>	<b>Explanation</b>
hping3	It is the name of the application binary.
-count(-c)	The packet count to send is defined numerically.
-interval(-i)	This attribute waits for the specified number of seconds or micro seconds between sending each packets. -interval X statement waits X second before sending each packet or -interval uX statement waits for X micro seconds before sending each packet.
-fast	10 packets are sent for every second.
-faster	It is faster than -fast attribute, however, it can not send packets as fast as the computer can perform according to the signal-driven-design of the computer.
-flood	It sends packets as fast as possible without considering to indicate the reply packets.
-interface(-I)	It specifies the interface name to use. Otherwise, the default routing interface is used.

Table 4.2. Hping3 Modes (Source: Sanfilippo, 2006)

<b>Usage of Modes</b>	<b>Meaning</b>
-0(-rawip)	Raw IP mode
-1(-icmp)	ICMP mode
-2(-udp)	UDP mode
-8(-scan)	Scan Mode
-9(-listen)	Listen Mode

#### 4.7.2.2. Hping3 Flag Options

While working in TCP mode, it is possible to specify which flag will be sent to the destination. There are a set of flags to be used in TCP mode as indicated in the Table 4.3.

#### 4.7.2.3. Hping3 IP Related Options

Hping3 also allows us to specify IP options for several reasons such as hiding the IP address information of the source node, sending from random source addresses to a destination and sending packets from one source to different destinations as explained in Table 4.4.

Table 4.3. Hping3 Flag Options (Source: Sanfilippo, 2006)

Flag Options	Flag Type
-S	Set SYN Flag
-A	Set ACK Flag
-R	Set RST Flag
-F	Set FIN Flag
-P	Set PUSH Flag
-U	Set URG Flag
-X	Set XMAS Flag
-Y	Set YMAS Flag

Table 4.4. Hping3 IP Options (Source: Sanfilippo, 2006)

IP Options	Explanation
-spoof(-a)	Hide the source IP address, this guarantees the target will not know the real source IP address.
-rand-source	This option enables the random source mode. hping3 will send packets from random source address.
-rand-dest	This option enables the random destination mode. hping3 will send the packets to random destinations according to specified rule like 10.0.0.x. When using this mode, it is required to use -interface option to specify the outgoing interface.

#### 4.7.2.4. TCP/UDP Related Options

Hping3 also has ability to set TCP/UDP options for some reasons like defining different TCP/UDP parameters such as port information, sequence number information and TCP window size as explained in Table 4.5

#### 4.7.2.5. Common Options

Hping3 also has some common options covering the whole modes like setting packet size. Each of the options are given on the Table 4.6. While generating traffic on the network, it is required to set packet size for each outgoing packet. This study uses to set packet body size to generate the network traffic by using hping3 application. Hping3 has also other options such as using file content, setting file name and setting TCP ACK etc.



Table 4.5. Hping3 TCP/UDP Related Options (Source: Sanfilippo, 2006)

<b>TCP/UDP Options</b>	<b>Explanation</b>
-s –baseport	Specify a source port. The default is assigned randomly.
-p –destport	Set destination port number, the default is zero. If + sign precedes with the port number like +1024, this implies that the destination port will be increased with each reply received whereas if double + sign exists, this implies that the destination port will be increased for each packet sent.
-w –win	Set TCP window size. The default is 64.
-o –tcpoff	Set fake TCP data offset. Normal data offset is one quarter of TCP header length.
-M –setseq	Set the TCP sequence number.
-L –setack	Set TCP ACK.
-Q –seqnum	This option collects the sequence number generated by the target.
-b –badchksum	Send packets with a bad UDP/TCP checksum
–tcp-timestamp	Enable the TCP timestamp option. Try to guess the timestamp update frequency and the remote system update.

Table 4.6. Hping3 Common Options (Source: Sanfilippo, 2006)

<b>Option</b>	<b>Explanation</b>
-d –data data size	Set packet body size.
-E –file filename	Use filename contents to fill packet's data.
-e –sign signature	Add signature.
-j –dump	Dump received packets in hex.
-J –print	Dump received packets printable characters.
-L –setack	Set TCP ACK.
-B –safe	Enable safe protocol, by using this option, lost packets are resent.
-u –end	In condition that –file filename option is used, tell you when EOF is reached.

Consequently, hping3 is used for this thesis to generate both random normal traffic and abnormal traffic originating from different hosts of SDN network. Additionally, several usage examples, installation steps of this application and organization of mostly known attack types is given in detail on Appendix E.

In this section, we mainly explained the environment requirements of the thesis and how different kinds of tools are used in this thesis, what they do and what additional information required to the usage of the tools. Additionally, installation processes are included for each tool to deploy an SDN network on the ONOS platform. For the automation of the tasks, our bash script application was briefly mentioned. And also we explained how to create a virtual realistic network for SDN environment by using Mininet. Then, we mentioned about the sFlow traffic analyzer tool to install s-flow agents on each switch and forward the packets to the external flow cache. In addition to this, we mentioned about hping3 known as a packet generation tool in detail. Then, different options and explanations of hping3 are given on this section. Lastly, performing a DDoS or DoS attack with the help of hping3 is explained in detail.

# CHAPTER 5

## IMPLEMENTATION

As stated in chapter 1, for this thesis, we have not put forward a novel study for the detection algorithm. In the future works of (Lu and Wang, 2016), the authors discuss the problem of the experiment limitations of their study, they have implemented their detection and response schemes on the local area network within the single SDN domain. Additionally, their approach has not been tested on the big enough network. As an aim of this study, we designed a little bit large network containing bots and by using the same detection mechanism, we have tested it on the emulation network by using Mininet for our designed large network. Then, we compared the obtained result with their results. The distributed design of a large network containing 63 hosts and 13 switches is explained in detail in chapter 6. In this chapter, we first mention the detection mechanism of the article. In addition to this, we also explained how to extend the distributed network of ONOS clusters by using GRE Tunnels. There are also other mechanisms for combining networks of one SDN controller with the networks of the other. We also briefly explained how different approaches related to the combination of the multiple SDN controllers can be applied.

### 5.1. Detection Method

According to the detection mechanism of (Lu and Wang, 2016), in order to detect the DDoS attacks on SDN environment, it is significant to determine a metric to measure the behavior of an attack or normal traffic. The method depends on a two dimension metric called DCD. This metric consists of two parameters which are  $\delta$  and  $\rho$ .  $(\delta, \rho)$  can be considered as a two dimension tuple.  $\delta$  quantifies how many nodes sending request to a destination for a destination flow. A destination flow of d-flow  $f_j$  is defined as all the packets having the same destination IP address (Lu and Wang, 2016). The other metric  $\rho$  quantifies the intensity of a flow for determining another dimension. Suppose that there are n number of nodes in a network. This can be defined mathematically as  $C = \{C_i | i = 1, 2, 3, \dots, n\}$ . Here,  $C_i$  represents the  $i$ -th node on the network. Each node

on the network has some destination flows to other nodes excluding itself. Therefore,  $C_i$  is also a set containing destination flows and it is defined mathematically as  $C = \{f_l | l = 1, 2, 3, \dots, m\}$ .  $f_l$  is defined as the  $l$ -th d-flow of  $C_i$ .  $|C_i| = m$  implies that there are  $m$  number of destination flows of the  $i$ -th node  $C_i$  (Lu and Wang, 2016).

### 5.1.1. Calculation of $\delta$ Parameter

In order to compute  $\delta$  value of d-flow  $f_j$ , the Equation (5.1) and the Equation (5.2) are used.

$$\text{Suppose : } A = \bigcup_{i=1}^n C_i \quad (5.1)$$

$$\forall f_j \in A : \delta_j = \sum_{k=1}^n J(f_j \in C_k) \quad (5.2)$$

$$\text{where : } J(f_j \in C_k) = \begin{cases} 1 & f_j \in C_k \\ 0 & f_j \notin C_k \end{cases}$$

At this point,  $\delta_j$  implies that how many nodes in this network are sending packet to a destination (Lu and Wang, 2016). This metric can bring out the behavior difference of Botnet-based DDoS attack flow and random normal flow (Lu and Wang, 2016). Suppose that  $X_j$  is a discrete random variable and  $t$  is a time parameter.

$$X_j(t) = \delta_j(t) - 1 \quad (5.3)$$

The Equation (5.3) evaluates the  $\delta_j$  value for every  $t$  instant. The subtracting one from  $\delta_j(t)$  implies that having destination flow to a node itself is theoretically ignored (Lu and Wang, 2016). However, in practice, a node can send a request to itself. Here, this condition is passed over. A random process is a random variable that varies with space or time. In accordance with the definition, we have time-varying parameter  $\delta$  for our case. Due to the fact that  $X_j(t)$  is based on the time and according to the theory of random process definition,  $X_j(t)$  can act as a Poisson Process (Lu and Wang, 2016). The possible values of  $X_j(t)$  is  $0, 1, \dots, n - 1$ . Here, it is required to find out the probability of  $k$  users sending request to a destination within  $0 - t$  time interval. This probability could be represented as Equation (5.4)

$$P\{X_j(t) = k\} = \frac{\lambda^k e^{-\lambda}}{k!} \quad (5.4)$$

### 5.1.2. Calculation of $\rho$ Parameter

$\rho_j$  is another dimension to evaluate the intensity of a destination flow.  $\rho_j$  parameter can be considered as a total value of a specific destination flow of a node. For the detection process, the  $\delta_j$  dimension by itself is insufficient, inasmuch as if a group of users send normally request to a destination, for instance, suppose that a thousands of users to learn the military status from electronic state application, these users can send request to application at the same time, then the application can not serve its users, for this kind of situation, users can not be considered as bot and the requested flows are also not considered an attack flow. Thus, it is required to determine the another dimension to measure the intensity of a flow to ensure whether the destination flow is an attack flow or not. Using sflow technology, it is possible to get all destination flows passing over the switches or routers on the network. We can get all flows for a certain time period. Similarly, the controller operating system allow us to obtain the topology information related with the network design.

For a specific node  $i$ , a function is defined to get the packet per second value of a destination flow as indicated in Equation (5.5). Here  $p$  is a packet per second value of a flow.

$$F_i(f_i) = p \quad (5.5)$$

In order to exemplify the situation for the real world application, it can be supposed that  $C_1$  node has an IP flow from 10.0.0.1 to 10.0.0.2. Here, 10.0.0.1 is the IP address of the  $C_1$  node considered as source node. 10.0.0.2 is the IP address of the node  $C_2$  accepted as the destination node. This flow can be defined as  $F_1(f_1) = 100$  packet-s/sec. For a specific destination flow which is denoted as d-flow  $f_j$ , the intensity value  $\rho_j$  is evaluated with the following equation.

$$\rho_j = \sum_{k=1}^n F_k(f_j) \quad (5.6)$$

By the summation represented in Equation (5.6), it is possible to get the total intensity value for each node considered as a destination node. With this calculation, we obtain a tuple data structure which is a key value parameter  $(\delta_j, \rho_j)$ .

### 5.1.3. Example Calculation of $(\delta, \rho)$

Figure 5.1 illustrates a graph to get more detail understanding of  $(\delta, \rho)$  calculation. Table 5.1 lists the related flow values consists of flow number, source IP address, destination IP address, flow value, source node and destination node.

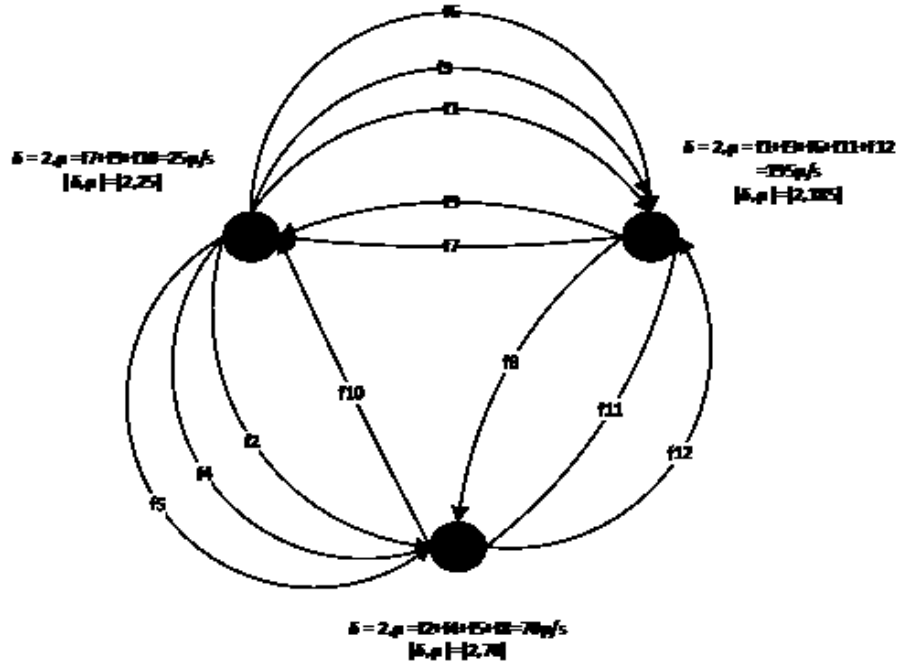


Figure 5.1. A Sample Graph of Nodes with  $(\delta, \rho)$  Calculation

According to the Figure 5.1, we observe that  $C_1$  node has 6 destination flows, however, in practice we consider  $f_1, f_3, f_6$  are the same flows with different values appeared in any time  $t$  instant. We treat these flows as if they are the same flows by adding individually. Then, this flow is considered as a destination flow  $f_j$  from  $C_1$  to  $C_2$  which are  $f_1 = f_3 = f_6 = f_j$ . In addition to this,  $C_1, C_2$  and  $C_3$  nodes have flows coming from at most 2 source addresses. Thus,  $\delta$  value of each node on the network is equal to 2. However, we will calculate it by executing the equations.

Table 5.1. Flow Table of Nodes on Figure 5.1

Flow No	Source Ip Address	Destination Ip Address	Flow Value	Src	Dest
$f_1$	10.0.0.1	10.0.0.2	100 p/s	$C_1$	$C_2$
$f_2$	10.0.0.1	10.0.0.3	20 p/s	$C_1$	$C_3$
$f_3$	10.0.0.1	10.0.0.2	50 p/s	$C_1$	$C_2$
$f_4$	10.0.0.1	10.0.0.3	15 p/s	$C_1$	$C_3$
$f_5$	10.0.0.1	10.0.0.3	25 p/s	$C_1$	$C_3$
$f_6$	10.0.0.1	10.0.0.2	10 p/s	$C_1$	$C_2$
$f_7$	10.0.0.2	10.0.0.1	5 p/s	$C_2$	$C_1$
$f_8$	10.0.0.2	10.0.0.3	10 p/s	$C_2$	$C_3$
$f_9$	10.0.0.2	10.0.0.1	15 p/s	$C_2$	$C_1$
$f_{10}$	10.0.0.3	10.0.0.1	5 p/s	$C_3$	$C_1$
$f_{11}$	10.0.0.3	10.0.0.2	10 p/s	$C_3$	$C_2$
$f_{12}$	10.0.0.3	10.0.0.2	15 p/s	$C_3$	$C_2$

According to the information obtained from Table 5.1,  $\rho$  value of  $C_1$  is calculated by adding the whole flows having the 10.0.0.1 destination IP address and we observe that  $f_7, f_9, f_{10}$  flows have this destination IP address and then we sum all flows having this destination address. Then,  $\rho$  value of  $C_1$  is calculated as  $f_7 + f_9 + f_{10} = 25$  p/s. Similarly, the value  $\rho$  value of  $C_2$  is calculated as  $f_1 + f_3 + f_6 + f_{11} + f_{12} = 185$  p/s. Eventually, the value  $\rho$  value of  $C_3$  is calculated as  $f_2 + f_4 + f_5 + f_8 = 70$  p/s. If we execute the Equation (5.1) and Equation (5.2) for the sample network given above, then we obtain the following equations.

$$A = \bigcup_{i=3}^3 C_i = C_1 \cup C_2 \cup C_3$$

$$C_1 = \{f_l | l = 1, 2, 3, 4, 5, 6\} \cup C_2 = \{f_l | l = 7, 8, 9\} \cup C_3 = \{f_l | l = 10, 11, 12\}.$$

Owing to the consideration of having the same destination IP address, the below flows are accepted as the same flows despite of the fact that they might have different source IP address and different flow value.

$$f_7 = f_9 = f_{10}, \quad f_1 = f_3 = f_6 = f_{11} = f_{12}, \quad \text{and} \quad f_2 = f_4 = f_5 = f_8$$

$$\begin{aligned}
\delta_{1|3|6|11|12} &= \sum_{k=1}^3 J((f_1 \in C_k)) \\
&= J((f_{1|3|6|11|12} \in C_1) + J((f_{1|3|6|11|12} \in C_2) + J((f_{1|3|6|11|12} \in C_3)) \\
&= 1 + 0 + 1 = 2
\end{aligned}$$

Therefore,  $\delta$  value of  $f_1 = f_3 = f_6 = f_{11} = f_{12}$  destination flows would be equal to each other. Here, we can judge that both of a node and a destination flow should have a  $(\delta, \rho)$  tuple. After the calculation of  $\delta$  and  $\rho$ , we can add two columns to Table 5.1 to present the  $\delta$  and  $\rho$  value of destination flows.

For  $f_7 = f_9 = f_{10}$ ,

$$\begin{aligned}
\delta_{7|9|10} &= \sum_{k=1}^3 J((f_7) \in C_k) = J((f_{7|9|10}) \in C_1) + J((f_{7|9|10}) \in C_2) + J((f_{7|9|10}) \in C_3) \\
&= 0 + 1 + 1 = 2
\end{aligned}$$

Finally for  $f_2 = f_4 = f_5 = f_8$ ,

$$\begin{aligned}
\delta_{2|4|5|8} &= \sum_{K=1}^3 J((f_{2|4|5|8}) \in C_k) \\
&= J((f_{2|4|5|8}) \in C_1) + J((f_{2|4|5|8}) \in C_2) + J((f_{2|4|5|8}) \in C_3) = 1 + 1 + 0 \\
&= 2
\end{aligned}$$

#### 5.1.4. $\lambda$ Parameter Calculation of Poisson

In order to find the probability  $k$  users together sending request to a destination, it is required to determine  $\lambda$  parameter of Poisson. For a specific time interval, it is possible to obtain  $\delta_i$  value of any node. The  $\delta_i$  variable is a parameter varying over time. Time is also varying parameter from  $t = 1$  to  $t = s$ . If we observe the variable during  $s$  second or within any time parameter. Then, we can calculate  $\lambda$  parameter for each node with the following Equation (5.7).

$$\hat{\lambda}_i = \frac{\sum_{t=1}^s \delta_i(t)}{s} \quad t = 1, 2, \dots, s \quad (5.7)$$



At that point, we used maximum likelihood estimation to determine  $\lambda$  of any node  $i$  at a certain time interval. This estimator represents the mean value of  $\delta_i$  within this time interval.

### 5.1.5. Threshold $\delta$ Calculation

After the  $\lambda$  value is determined, it is possible to choose Threshold  $T_\delta$  according to Poisson Probability function as indicated in Equation (5.8) (Lu and Wang, 2016).

$$H(x) = P(X \leq x) = \sum_{k=0}^x \frac{\lambda^k e^{-\lambda}}{k!} \quad (5.8)$$

After threshold value is chosen, if  $\delta_j > T_\delta$ , then we could judge this d-flow  $j$  is an attack flow with confidence level  $\alpha$ .

### 5.1.6. Threshold $\rho$ Calculation

Suppose that the maximum exit bandwidth of a protected node limited by Internet Service Provider is defined as  $L$  and this can also be a maximum service capacity of a node in the network (Lu and Wang, 2016). Then, the maximum service capacity unit is the same unit with the parameter  $\rho$ .  $\theta$  is defined as a sensitivity factor for the detection. Then, threshold  $\rho$  is evaluated as  $T_\rho = \theta L$ . After calculating  $(\delta_j, \rho_j)$  tuple, this point can be visualized on the Figure 2.1. If the point  $(\delta_j, \rho_j)$  falls into area A, then this flow is considered as a normal flow. If the point  $(\delta_j, \rho_j)$  falls into area B, then it is possible to say that a group of users are fully utilizing the bandwidth like downloading film. The situation B is not considered as an attack. If the point  $(\delta_j, \rho_j)$  falls into area C or D, then d-flow  $j$  is considered as an attack flow.

### 5.1.7. The Detection Procedure

The pseudo-code of the detection algorithm is demonstrated as below. Algorithm 1 calculates the  $(\delta_j, \rho_j)$  tuple for each destination flow of each node. Then, it tries to perceive the destination flow falling into C and D regions according to the boundary of

Threshold  $\delta$  and Threshold  $\rho$ . In condition that destination flow falls into Area D, the approach of the algorithm is that the destination flow is an attack flow with possible normal. For the Area C, the approach is that the destination flow is normal flow because of having a normal  $\rho$  parameter. However, the large number of source addresses are related to this flow. Due that reason, it is possible for this flow to be an attack. On the contrary, Algorithm 1 collects and returns the  $(\delta_j, \rho_j)$  tuple as a set of B for the normal situation.

---

#### Algorithm 1 Detection

---

```

1: procedure DETECTION( $C_1, C_2, \dots, n$  : d-flow set of each node )
2:    $C = \{\forall C_i, | \exists f_j, j = 1, 2, \dots, mi, j \in N\}$ 
3:    $A = \cup_{i=1}^n C_i$ 
4:    $B = \emptyset$ 
5:    $m = |A|$ 
6:   for  $i=1$  to  $m$  do
7:      $\delta_i = 0$ 
8:      $\rho_i = 0$ 
9:     for  $j=1$  to  $n$  do
10:      if  $f_i \in C_j$  then
11:         $\delta_{i+} = 1$ 
12:         $\rho_{i+} = F_j(f_i)$ 
13:      end if
14:    end for
15:    if  $\delta_i \geq T_\delta \wedge \rho_i \geq T_\rho$  then
16:      The flow is the attack flow with possible normal (Area D);
17:    end if
18:    if  $\delta_i \geq T_\delta \wedge \rho_i \leq T_\rho$  then
19:      The flow is normal flow with possible attack (Area C);
20:    else
21:       $add(\delta_i, \rho_i)$  to  $B$ 
22:    end if
23:  end for
24:  return  $B$ 
25: end procedure

```

---

## 5.2. Design of the Application

By using distributed clusters of SDN environment, northbound APIs of ONOS controller and sFlow-RT collector, it is possible to deploy a software application to execute the detection algorithm on the designed network. The application uses the ONOS RESTAPI to get the global topology information from ONOS controller and also uses the sFlow-RT RESTAPI to obtain the traffic information from agent software deployed on the

switches as indicated in Figure 5.2

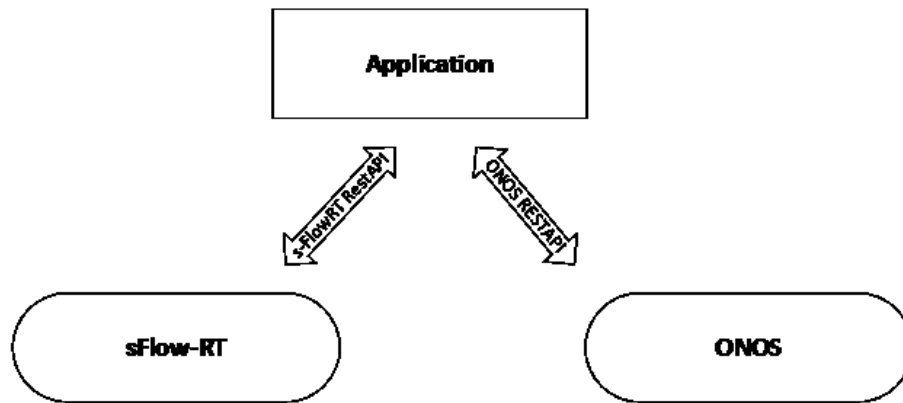


Figure 5.2. Communication Structure for Application

As compared to the application of (Lu and Wang, 2016), our application is not a modular application. However, the application consists of a console application written in Python performing the tasks of the whole modules excluding response module because of being not interested in the response scheme for this thesis project. The application initially gets the topology information such as the information of all nodes in the network. Then, the topology information is pushed to the sFlow-RT collector. sFlow-RT collector begins to sniff the whole sflow-RT agents deployed on the switches to get the all flows from the points that agents are installed on the network. Then, the obtained flows are separated into the destination flows of each node. After the separation process, every individual node has the destination flows in condition that this related node has the flow traffic from the corresponding node's IP address to the other destination IP addresses. After that, all flows and all nodes are send to the detection method. The detection method analyzes the flow and topology records by using the detection algorithm to produce the current  $(\delta, \rho)$  tuple of each destination flow. By analyzing destination flows,  $(\delta, \rho)$  tuple of each node is determined. Besides, the detection method execution is repeated for every second, then it produces current metric value containing both the historical  $(\delta, \rho)$  record stored for every second and the observation frequency information for different  $\delta$  values. According to the produced metric values, the unknown parameter  $\lambda$  of Poisson distribution is estimated by using the mean value of  $\delta$  within a certain time interval. According to the number of active nodes, the probability of k users sending request to a destination is calculated and then presented graphically on a probability distribution plot. Furthermore, after metric values are analyzed, the distribution collaboration plot is drawn for the total traffic sniffed within

a certain time interval. This plot gives the detection result. Eventually, the frequency of different  $\delta$  values are given with 3D plot to observe at which frequency the  $\delta$  value is obtained for a specific time  $t$ .

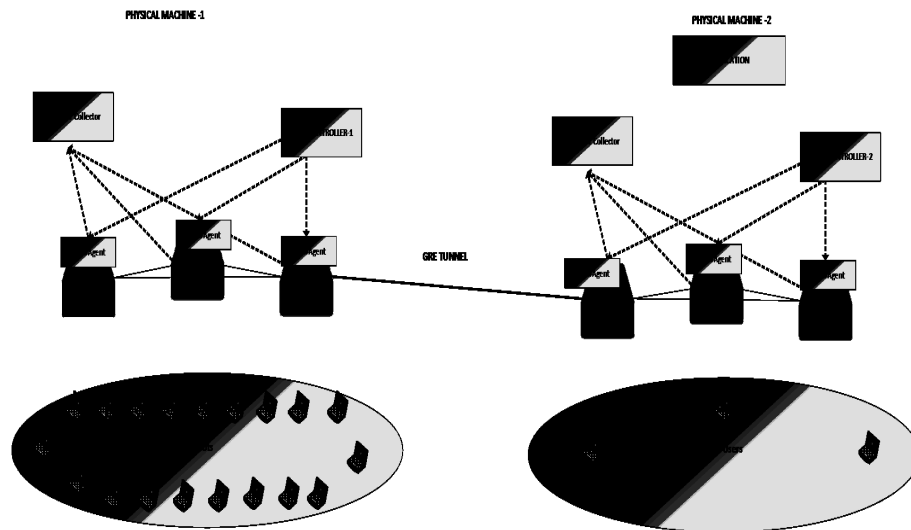


Figure 5.3. Deployment Scenario for Application

In the Figure 5.3, the deployment scenario is depicted. In the deployment scenario, we have used two ONOS controller on different physical machines. In the data plane, there are network switches and hosts, however, hosts of the left network is a little bit larger than the right network. Because, the network is designed to be able to used the hosts of the controller-1 as bots of the attack organization. There are also network switches on the data plane, the switches have sFlow-RT agents installed on them. Consequently, the network traffic is monitored via sFlow-RT agents pushing the network traffic to the central sFlow-RT collector for different machines. Furthermore, the rightmost switch on controller-1 and the leftmost switch on the controller-2 are connected to setup a GRE tunnel between these switches. GRE Tunnel will be explained a little bit more detail in the section 5.3. For the experimental tests, the application just runs on the second physical machine, however, this application can also be deployed on different machines to detect the attack on multiple servers. Moreover, we have written the application as a Python project and for the data visualization, we used matplotlib library of Python and additionally, PostgreSQL docker container is used to store the flow records, node information obtained from the controller and also attack detection metrics such as the frequency information and historical  $(\delta, \rho)$  tuples.

### **5.3. Cluster Communication Methods**

In this section, we examined two methods for the cluster communication. One of them is to set up GRE Tunnels among the different network elements of different clusters. The second method is ONOS SDN-IP application.

#### **5.3.1. GRE Tunnels**

Generally, Mininet emulates an entire network of hosts and switches in a single computer or virtual machine. However, the system resources limit the topology size. By extending Mininet with cross-machine virtual links, this kind of work allows researchers to create and emulate with more network elements (Blankstein et al., 2013). A single machine can only emulate a certain number of network elements with Mininet. Due that reason, according to the research having been implemented by (Blankstein et al., 2013), they extended the Mininet capability to be able to execute in a cluster by using the GRE Tunnel capability of OpenFlow switches.

Mininet network composed of controllers, switches, hosts and links. Mininet hosts are shell processes. Each host generated with Mininet is a child process inheriting from the local machine and has its own namespace and Ethernet interface. Each link between the nodes is a virtual Ethernet pair acting as if they are having a wired connection(Blankstein et al., 2013). However, Mininet link does not support connecting the different nodes under the clusters. For this reason, to support this kind of functionality, GRE protocol has been developed by Cisco System. The GRE protocol is broadly speaking protocol for performing encapsulation of an arbitrary network layer protocol over another arbitrary network layer protocol (Blankstein et al., 2013). When a system has a packet needing to be encapsulated and routed, two step processes will be performed. The packet to be sent is encapsulated in a GRE packet as a first step and then the GRE packet encapsulated is encapsulated again by the delivery protocol managing the actual forwarding of the packet (Blankstein et al., 2013). Figure 5.4 illustrates the encapsulated payload packet with GRE and delivery header.

The advantage of this protocol is that the delivery layer packets and the encapsulated GRE packets look same when the forwarding process is performed (Blankstein et al., 2013). For this reason, GRE protocol is running over IP protocol to transmit the

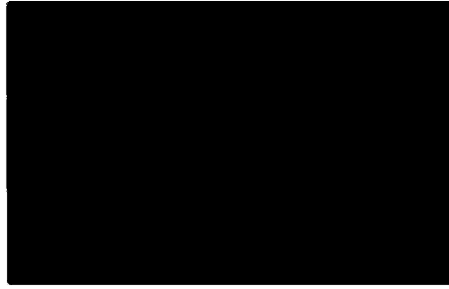


Figure 5.4. Encapsulated Payload Packet

Ethernet packets. Although using GRE tunnel is possible to connect multiple remote sites with switches, this protocol can also be constructed to connect two routers with a virtual link. For the requirement of using this protocol, two different Mininet clustering must be set up on different machines. In order to implement this, each instance will negotiate on how many GRE link is created and which devices are paired. After executing two different scripts on different machines by setting up negotiated points, then the topology is generated and Mininet clustering will assign a name for each GRE interface (Blankstein et al., 2013).

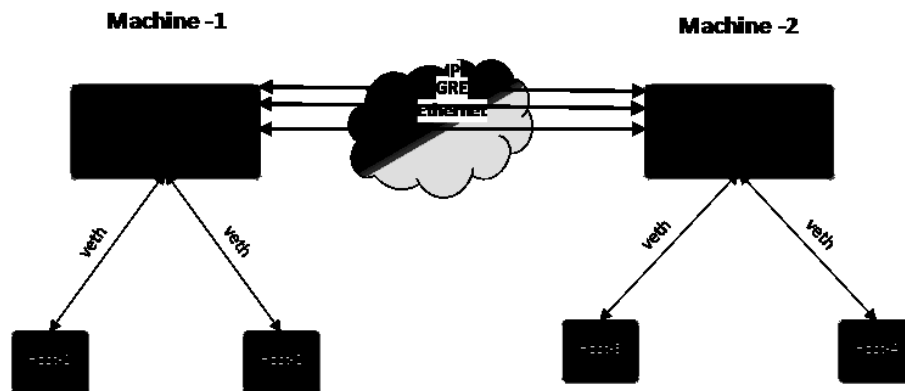


Figure 5.5. A simple Two Switch Topology

To exemplify the implementation, a simple two-switch topology can be constructed on two different machines as demonstrated in Figure 5.5. Two hosts are connected to each switch. It is noteworthy that Switch-1, Host-1 and Host-2 are generated by the Mininet clustering of the Machine-1 and the controller of Machine-1 is responsible for them. On the other hand, the Switch-2, Host-3 and Host-4 are generated by the Mininet clustering of Machine-2 and the controller of Machine-2 are responsible for them. A detailed example

to construct GRE Tunnel for the Figure 5.5 is given on Appendix F.

### 5.3.2. SDN-IP Application

SDN-IP is an ONOS application that allows a SDN to connect external networks using the BGP protocol (Hart and Koshibe, 2016). From the point of view of global networking, the SDN network is considered as a single AS. SDN-IP application achieves the integration mechanism between the ONOS and BGP routers which can be internal within the SDN network or external within another AS. At the protocol level, SDN-IP acts as a regular BGP speaker. SDN-IP allows an SDN network to peer and exchange traffic with the adjacent external networks. It has an operational flexibility feature because it can run on multiple ONOS clusters. It has a compatibility feature for adapting the network with other networks using BGP protocol. SDN-IP application has also high availability feature, inasmuch as the application continue to run until at least one SDN-IP application running on any cluster to keep the consistent forwarding state. It is also scalable because it can work on multiple ONOS clusters.

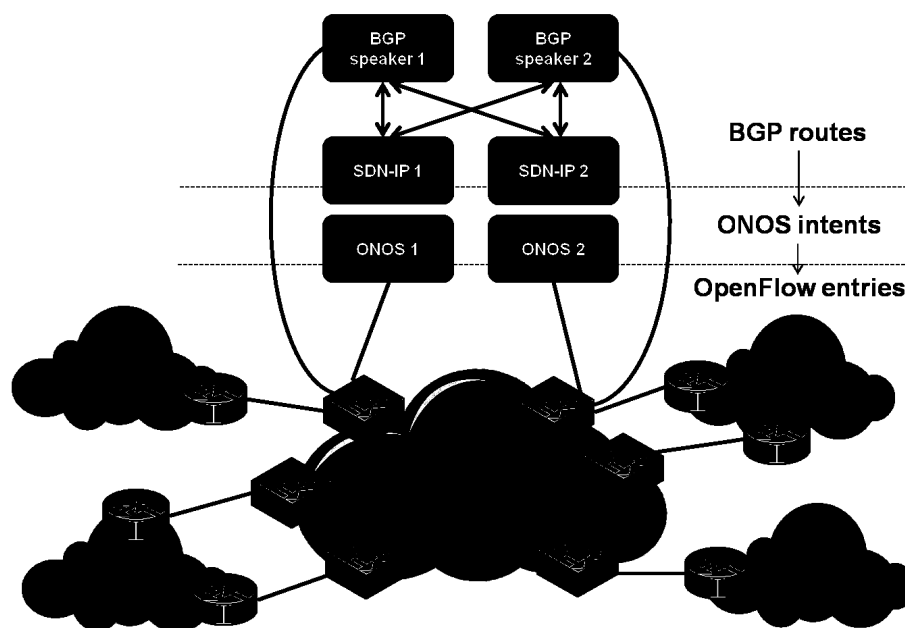


Figure 5.6. SDN-IP Architecture (Source: Hart and Koshibe, 2016)

Figure 5.6 illustrates SDN network describing the SDN-IP application. The network consists of two ONOS nodes running SDN-IP application on each of them. Each

external network is considered as a different autonomous system. There exist 4 external autonomous systems and two internal BGP routers on the typical deployment scenario. Within the SDN-IP network, there are OpenFlow switches and internal BGP speakers needing to have at least one connection with the OpenFlow switches. The BGP speakers might be an existing BGP router or any software router such as Quagga. The speakers use e-BGP peering sessions to exchange the routing information with the border routers whereas iBGP peering sessions are established among the BGP speakers and the SDN-IP instances. The routes advertised by the external routers are received by the BGP Speakers. Then, BGP speakers propagate these routes to other external routers. The routes are processed according to the normal BGP processing (Hart and Koshibe, 2016). Similarly, routes are also sent to the whole SDN-IP instances. The SDN-IP application converts these routes to Application Intent Requests which are defined as network-level abstractions or policies that allow applications to control the network behavior. Then, ONOS translates these intents requests into forwarding rules in the data plane. Forwarding rules are used to share traffic information among different interconnected IP networks (Hart and Koshibe, 2016). Additionally, in condition that there are several instances of ONOS clusters where SDN-IP runs, then all SDN-IP instances receive the same number of BGP routes while one of the instances act as a leader to install the necessary intents. In this section, we described the SDN-IP application of ONOS. As a final remark, ONOS allows developers to build different applications like BYON . It is possible to write a forwarding application in ONOS, and with the help of BYON feature of ONOS, it enables to transmit the traffic from one side to another side.



# CHAPTER 6

## EXPERIMENT AND ANALYSIS

This chapter firstly explains generating distributed networks on multiple clusters, and covers tests performed on a single controller and multiple controllers.

### 6.1. Distributed Network Generation

After installing tools specified in the chapter 4, we can build a distributed SDN network on two physical machines. As an initial process, it is required to use cell mechanism of ONOS to construct the environment variable of ONOS. Before executing cell file, IPv4 addresses of two machines need to be obtained. After configuring the cell file of each computer with the obtained IP addresses, cell file can be executed with the cell alias as specified below.

```
1 [REDACTED]
```

In case the cell file may change the ONOS\_USER variable as sdn, because ONOS expects that the machines are created with the name called as sdn. It is required to reset as the computer username. In our case, our machines have two different names. ONOS\_IP and ONOS\_ROOT variables also should be defined as specified below.

```
1 [REDACTED]
2 [REDACTED]
3 [REDACTED]
```

Then, we change the directory from home to onos directory. Then, we start the onos with ok clean command.

```
1 [REDACTED]
2 [REDACTED]
```

After a while, ONOS controller will be started. Figure 6.1 demonstrates the running ONOS applications and required installations of ONOS after the execution of ok clean command. As soon as the ONOS controller starts, ONOS GUI will be available on <http://192.168.1.101:8181/onos/ui/index.html>.

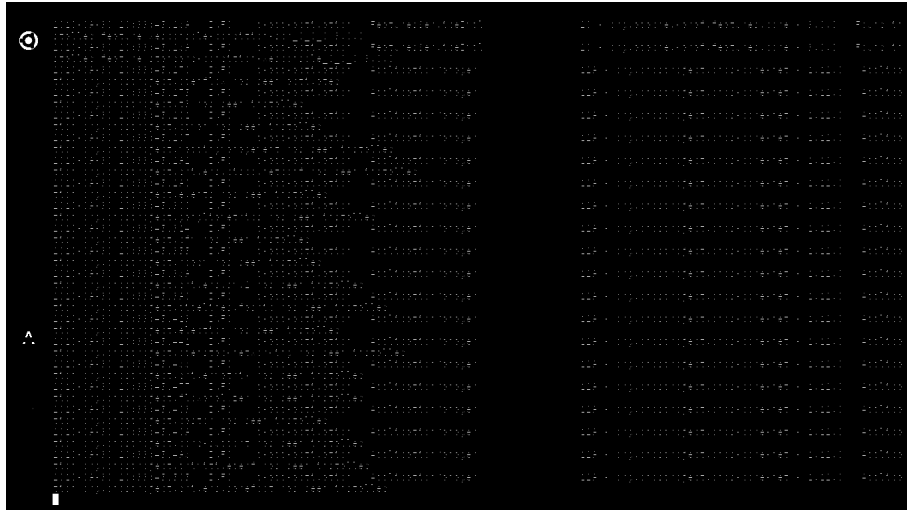
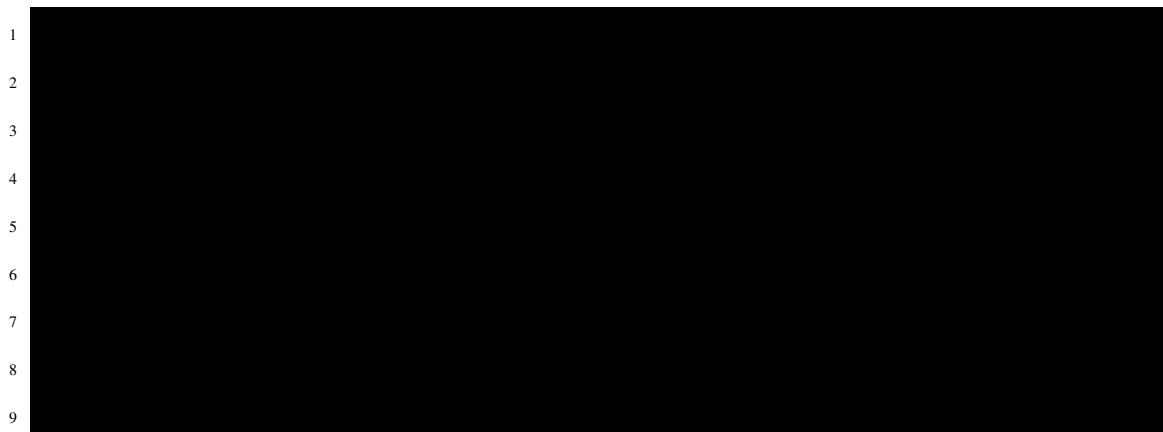


Figure 6.1. Running ONOS Controller

**Connecting ONOS CLI:** After the controller is executed, it is possible to access ONOS CLI by specifying the controller IP address with onos alias. Figure 6.2 illustrates the CLI access to ONOS controller to activate or deactivate ONOS applications. Here, we activated forwarding application, because in order to generate traffic among devices, it is required for forwarding application to be enabled on ONOS to experiment ICMP traffic on SDN network. There are a set of CLI command examples to get information of the corresponding devices on the network or activate and deactivate a specific application on CLI.



When the ONOS controller stands up, we can change the directory to mininet to execute our script existing in the mininet directory that will generate the network devices of the controller running on 192.168.1.101. The script is executed with the below command on the mininet directory.

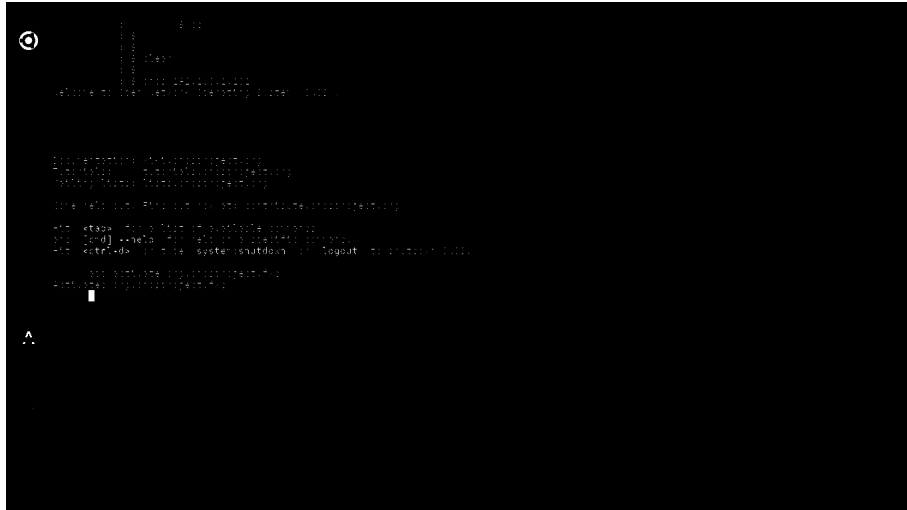
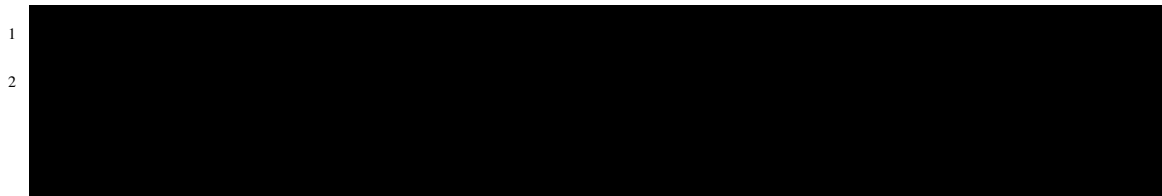


Figure 6.2. ONOS CLI



If we analyze the run script, custom topology called `onosTopology.py` is executed by referencing the ONOS controller. The default OpenvSwitch port and the protocol name is specified. After this script is executed, Figure 6.3 is observed on the terminal window. That is to say, a mininet terminal is opened to manage the entire network devices.

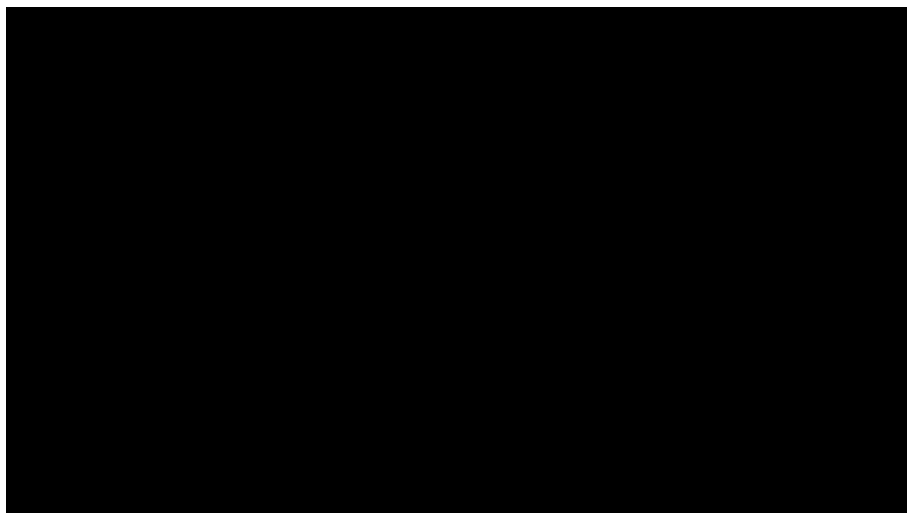


Figure 6.3. Mininet CLI

Figure 6.4 illustrates the code content of the onosTopology.py file generating network design on the SDN controller.

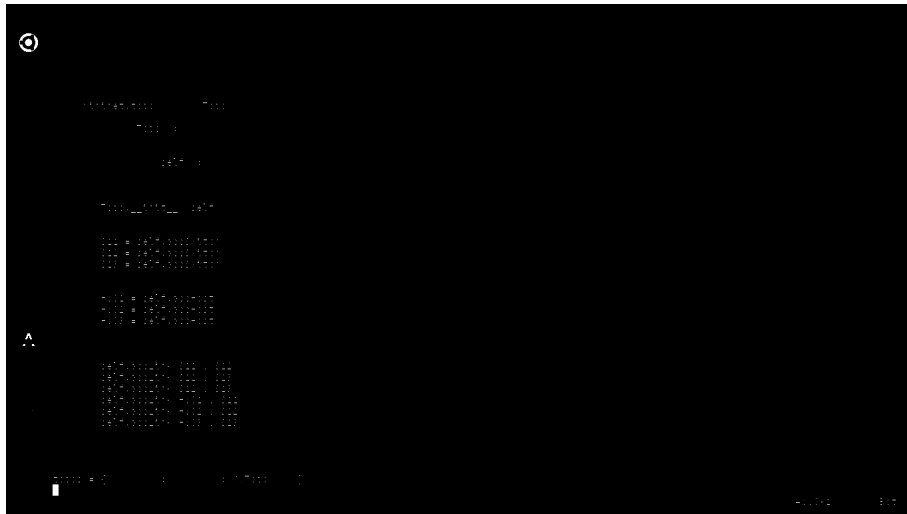
A terminal window with a black background and white text. The code is a Python script for generating network topology. It includes imports for 'mininet' and 'onos', and defines a class 'onosTopology' with methods for creating hosts, switches, and links. The script is being executed in a shell environment.

Figure 6.4. Mininet Python File

Then we can log in to ONOS GUI with username onos and password rocks. Then, the corresponding mininet script generates the topology on the ONOS controller as shown in Figure 6.5.



Figure 6.5. Initial Topology View on First Controller

After the network elements are generated on the ONOS controller, the computers connecting to the switches are not observed in the initial phase. After generating ICMP

traffic by using the hosts and then pressing the 'h' button on the keyboard, then the hosts will appear as demonstrated in Figure 6.6.



Figure 6.6. Generating Traffic on ONOS

It is possible to get network device information by using dump command of Mininet. Then, the controller, switch and host information are listed on the terminal. It is also possible to get the controller and device information using RESTAPI of ONOS. ONOS GUI also presents the device, cluster, host and also application information on the GUI. After observing the device information on the Mininet console, we wonder ping reachability of the devices, thus we execute pingall command of Mininet as exemplified in the Figure 6.7. As soon as we execute pingall command, we observe that the host 10.0.0.2 appears on the controller GUI. Because, any traffic has not yet been directed to this host before the pingall execution.

Up to now, we have one ONOS controller and three switches and three hosts connected to each switch separately. In order to run another ONOS controller on another computer, it is obligatory to set up ssh connection among different machines.

**SSH Configuration of Machines:** In order to set up SSH connection between controller machines, below scripts are followed to create public key for a machine and then this key is shared with the other machine. Below scripts indicate these steps in more detail.

```

1 [REDACTED]
2 [REDACTED]

```

The above script will ask to the password of the other machine, then we can successfully connect to the other computer as specified in the Figure 6.8.

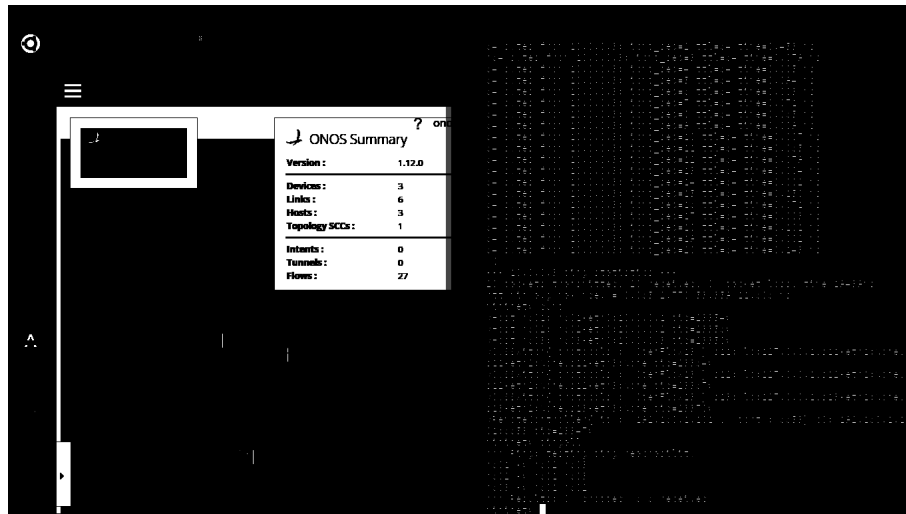


Figure 6.7. Pingall Execution in Mininet

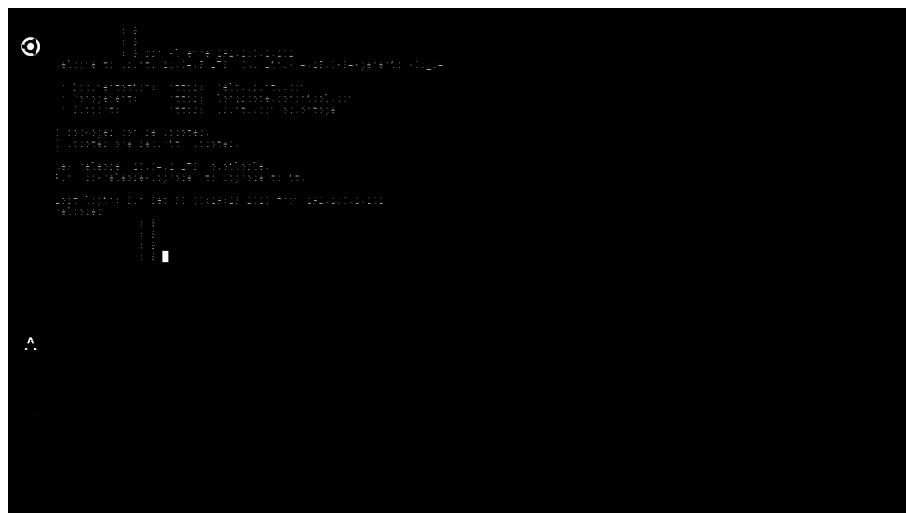
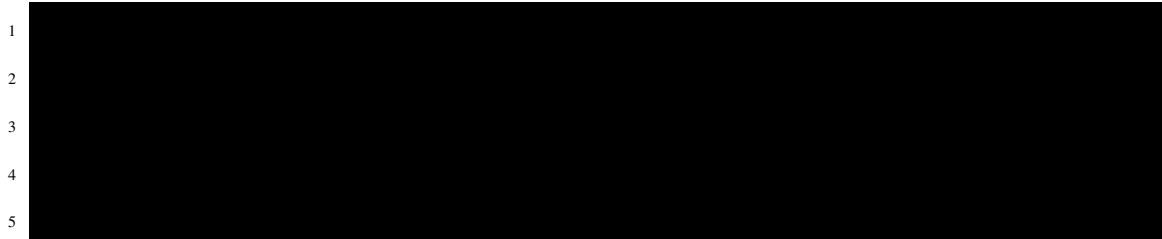


Figure 6.8. Connecting Other Machine with SSH

**Configuration of the Second Machine:** In order to construct the distributed clusters, we follow the same steps for the second machine. As an initial phase, we execute the cell file of the second machine. Then, we set required environment variables of ONOS. A sample cell file for the second machine is illustrated as below.



Then, we execute `ok clean` command to run ONOS controller on the second machine. As explained previously for the first machine, we access the ONOS CLI to activate forwarding application for this controller machine. ONOS applications can also be activated from the GUI. At that point, we executed different topology script residing in `mininet` directory. After the script is executed, the controller running on `192.168.1.102` can be accessed by the GUI. The username and password are entered as `onos` and `rocks` respectively. Then, we observe the second controller GUI on the second machine as specified in the Figure 6.9. However, for this time, we observe a different topology than the first topology im-

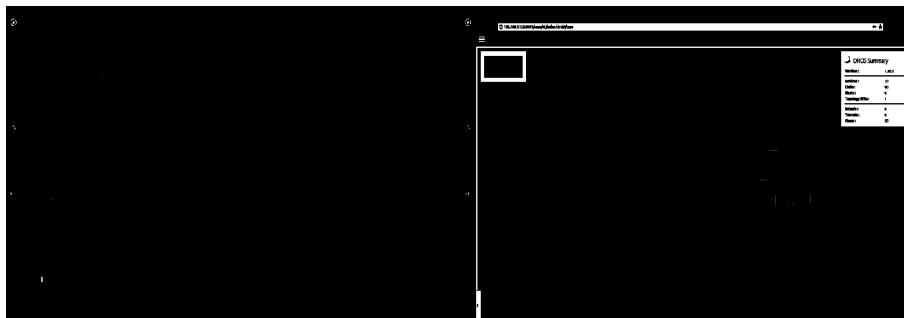


Figure 6.9. Initial Topology View on Second Controller

plemented. Here, we changed the name and the content of the script file to produce a mesh topology with 10 switch and 60 hosts. Each switch connect to 6 hosts separately. After the topology is created, we send a ping request from `10.0.0.1` to `10.0.0.2`. Then, we only observe these hosts on the network as indicated in Figure 6.10.

After that, we can get network device information from `mininet` to check the information of all devices and controller. Then, `pingall` command is executed to test the ping reachability of devices on `mininet` terminal. Figure 6.11 indicates the execution re-

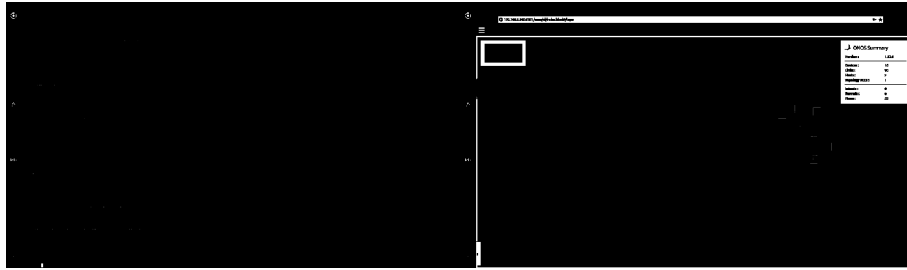


Figure 6.10. Ping Test on Second Controller

sult of pingall command on the left-hand side. On the right-hand side, we can observe that all hosts appeared and ONOS allows the topology background modification to set up the network on different countries. At that point, we are ready to combine the distributed SDN clusters.

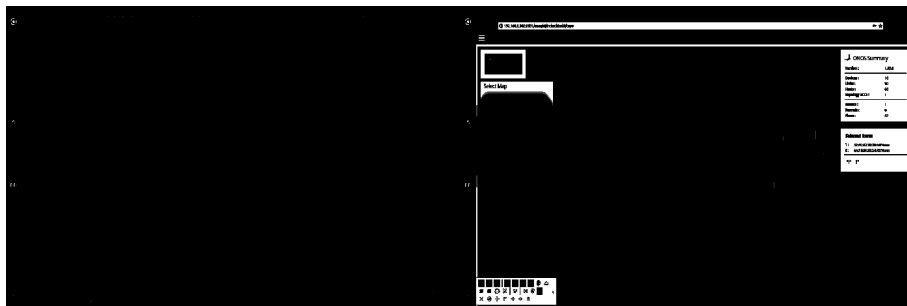


Figure 6.11. PingAll Test on Second Controller

**Combination of ONOS Controllers:** With regard to construct the distributed ONOS clusters, ONOS provides an alias named onos-form-cluster that can only be run under the onos/tools/test/bin directory. The requirement for clustering is to construct ssh-connected machines. The below script illustrates the usage of onos-form-cluster alias.

```
1 [REDACTED]
```

After executing this script, the existing topologies are destructed on both controllers. The distributed clusters are prepared by the controller machines approximately 2 or 3 minutes later. Figure 6.12 indicates that the sample execution of onos-form-cluster alias. However, while this alias is used, it is worth to say that if this command is executed on the first machine having 192.168.1.101 IP address, the first argument coming right after the alias should contain the other machine's IP address. Then we again login to ONOS GUI 2 or 3 minutes later, we can observe that the cluster has been set properly



with each topology belonging to different controllers as shown in Figure 6.13. Then, we resend ping to visualize the hosts again. Because, the topologies are recreated by the onos-form-cluster alias.

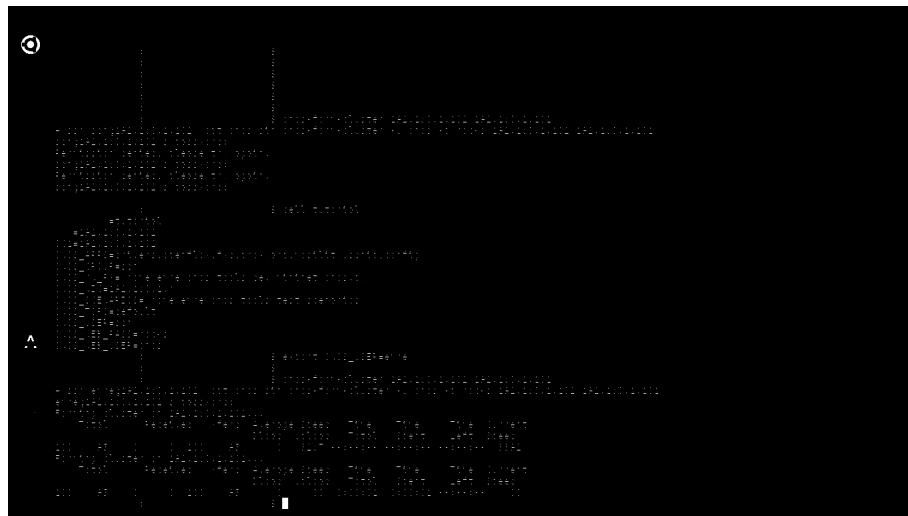


Figure 6.12. Cluster Combination

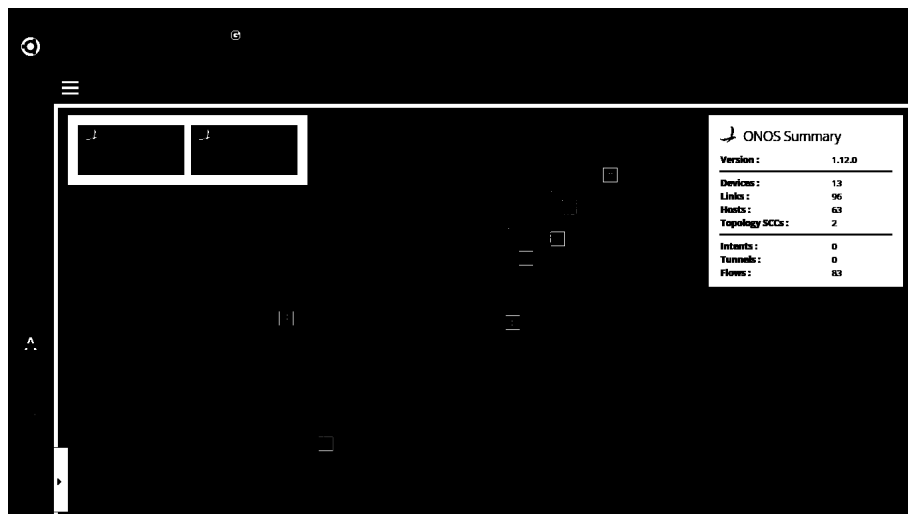


Figure 6.13. Combined Topology View

**Installation of s-Flow Agents:** After the cluster is established, we can install sflow agent on the whole switches on the network. Here, we install the agents just for the navy blue switches belonging to 192.168.1.101 cluster. Before installing sflow agents, we need to start sflow-RT collector on the localhost. We changed our directory to sflow-rt. The below script illustrates how to start sflow on s-flow-rt directory.

1 [REDACTED]

The default listen port of sFlow is 6343. It will serve to the user on 8008 http port. After starting sflow, agents can be installed on the s11, s12, s13 as shown in Figure 6.14.

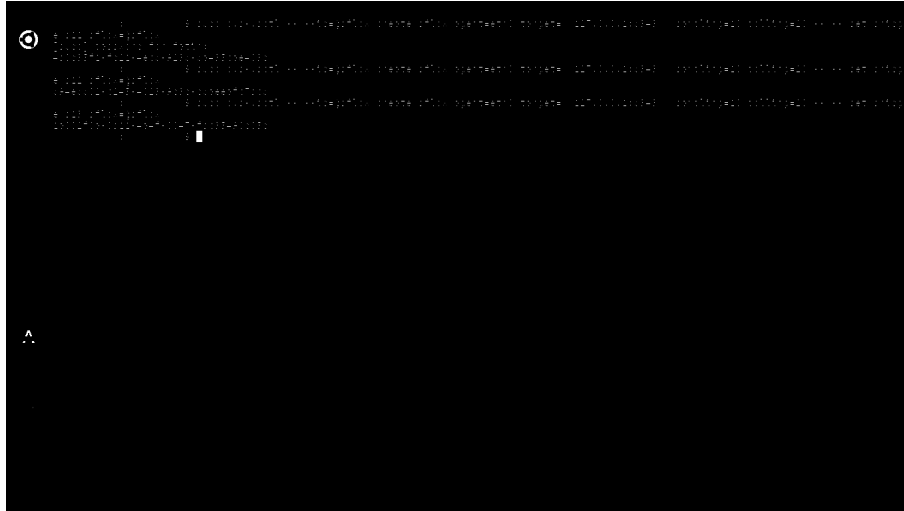


Figure 6.14. Installing sFlow Agents

At that point, sflow-RT GUI is available on the localhost 8008 port. It is observed on the page that there are a group of tabs on the user interface composed of applications, agents, metrics, keys, flows, thresholds and events. An existing application can be run with sFlow-RT. On the other hand, sFlow-RT allows developers to write their own application and deploy within the app folder inside the sFlow-RT directory. Additionally, real-time metric values are presented by means of metrics tab on the GUI and REST API according to the defined flow cache. After analyzing the functionality of sflow GUI, it is necessary to create a flow definition on the flows tab. This page asks the user to enter the flow definition on the flows page. The flow definition contains name, keys and value sections. Flow name is defined by giving a specific name for the flows. Because, after creating the traffic on the network, then we need this name to search the name on the agents page. Additionally, we specify the value of keys as comma separated ipsource, ipdestination and stack. Lastly, we specify the value variable as byte. After that, if there is any traffic on the network, flow definitions are observed with the specified name on the agents page. We can click on this flow definition, then we visualize the flows by time graph shown in Figure 6.15. Besides, on the mininet terminal, we can use xterm to reach the terminal window of the network hosts.

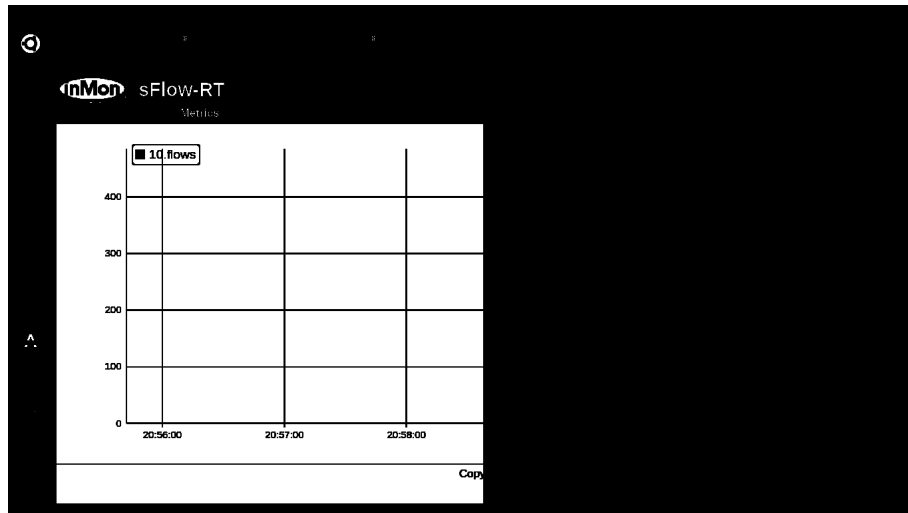


Figure 6.15. Flows By Time Graph of sFlow

As soon as we start the ICMP traffic, we start to observe normal ping values on the graph shown in the Figure 6.16.

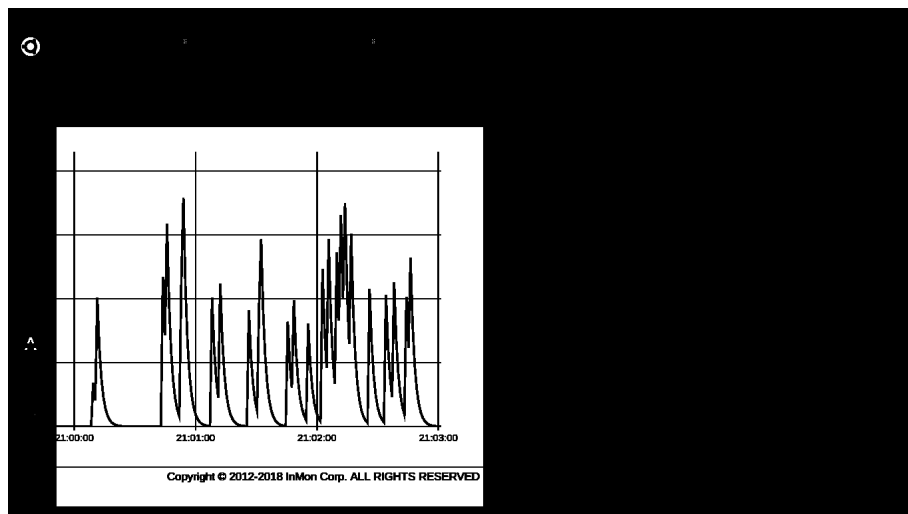


Figure 6.16. Normal ICMP Traffic Visualization on sFlow

In condition that ICMP traffic is created from 10.0.0.3 to 10.0.0.1, flows by time graph shows how many flows as byte value are passing through the agent at that second. After a while, we stop the ICMP traffic, then we organize an ICMP flood attack from 10.0.0.3 to 10.0.0.1 by using the below command. On the terminal window of 10.0.0.3, we will execute the below script.



Then, we can observe that while we get byte values for normal ping traffic, after creating ICMP flood attack, then the obtained flow values are not byte values anymore, we have flow rates up to megabytes as shown in Figure 6.17.

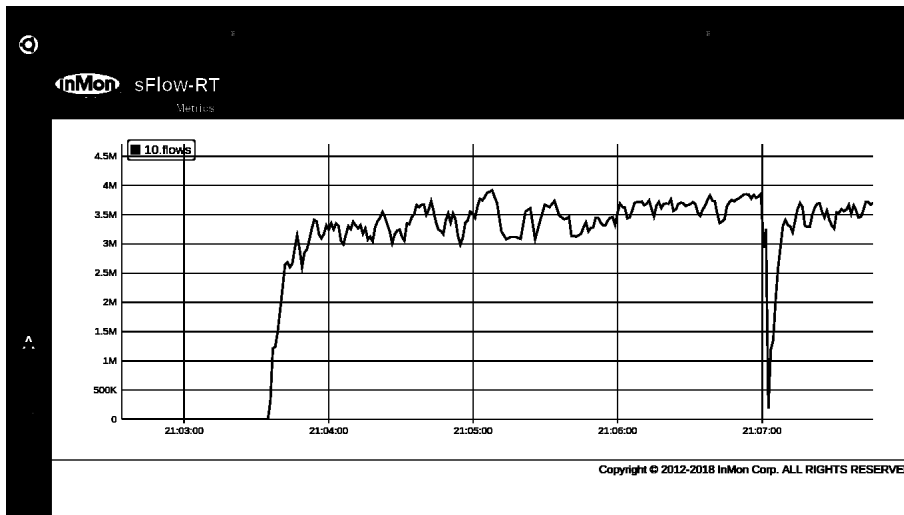


Figure 6.17. Attack Traffic Visualization on sFlow

Besides, it is possible to obtain the real-time megabyte value from the flow definition on the flow tab. The Figure 6.18 illustrates a sample values of the flow definition parameters for the virtual hosts.

ipsource	ipdestination	stack	bytes
10.0.0.3	10.0.0.1	eth.ip.icmp	3.498M

Copyright © 2012-2018 InMon Corp. ALL RIGHTS RESERVED

Figure 6.18. Flow Definition Parameters on sFlow

In addition to this, ONOS also points out the flow values with different colors on the GUI. If the channel has kilobyte per second value, the orange color appears on the link shown in the Figure 6.19.



Figure 6.19. Observing Flow as Kbps value on ONOS

If the channel has a megabyte per second value, then the color will be dark green as shown in Figure 6.20.



Figure 6.20. Observing Flow as Mbps value on ONOS

If the channel has gigabyte per second value, then the color is gradually darkened as shown in Figure 6.21.

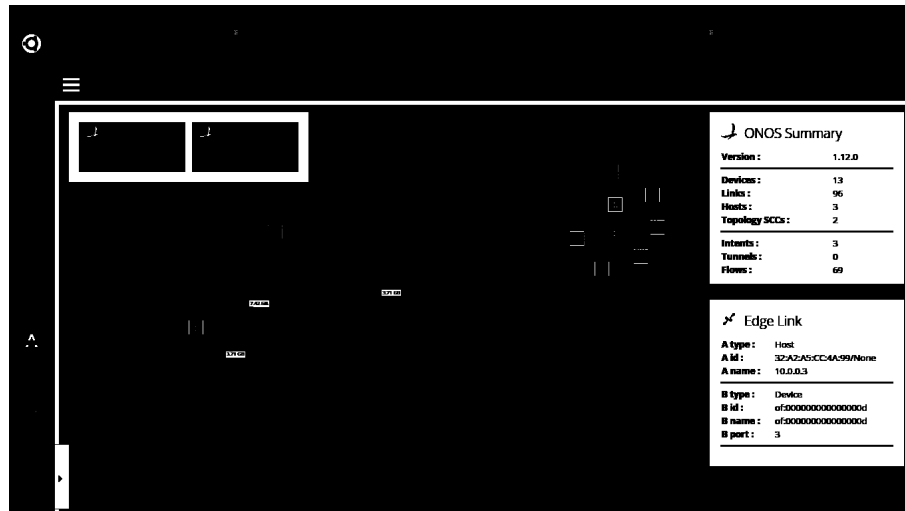


Figure 6.21. Observing Flow as Gbps value on ONOS

Consequently, in this section we have completed the analysis of the distributed network generation. At that point, this experiment enables us to judge the observation of the abnormality between the normal traffic and attack traffic by using sFlow-RT collector application and ONOS together.

## 6.2. Experiment with Multiple Controllers running on Two Physical Computers

This section contains normal and abnormal traffic experiments performed on two physical computers by extending the topology with GRE Tunnel between one of the switches on the first cluster and one of the switches on the second cluster.

### 6.2.1. Normal Traffic Test

In this experiment, we used two physical computers. The first one has an Intel Core i7 2670QM processor having 2.2 GHz processor speed, 8GB RAM and it has 480GB SSD whereas the other computer has Intel Core 2 CPU having 2.8 Ghz speed and 4GB RAM and its total HDD capacity is 230GB. In this experiment, we used 63 hosts and 13 OpenFlow switches created by a Python script. The emulation is performed

by the Mininet application. After that, our bash script application is triggered to provide automatically execution of all necessary applications. Then, the configuration of sFlow agents and the flow cache definitions are automatically done by the python script with the help of sFlow.py script shipping with the sFlow-RT application. In order to monitor the traffic, we used this application developed by InMon. The first computer has a triangle topology while the other has a mesh topology having 10 switches and each one of them has 6 host as indicated in Figure 6.22. Then, we integrated these topologies by creating a Linux GRE Tunnel on one of the switches of each cluster. After the topology is generated, hping3 application is used to generate the random normal traffic on the distributed network.

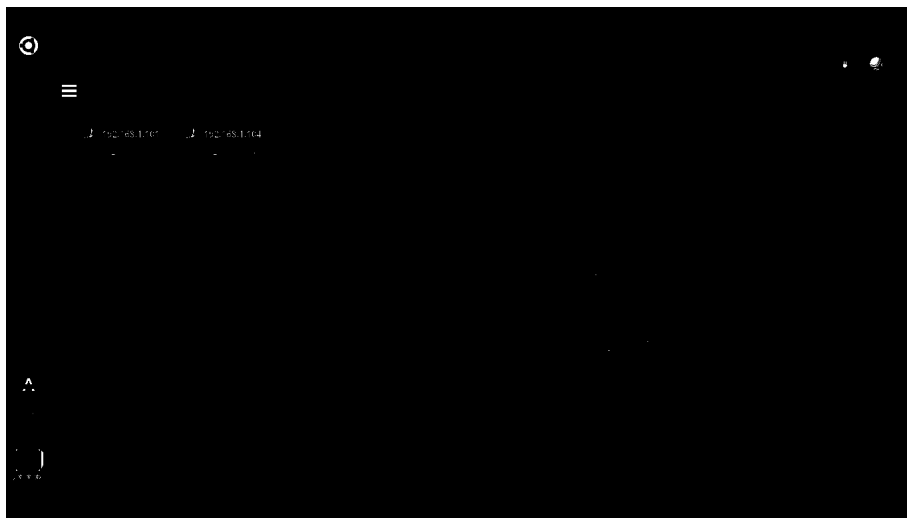


Figure 6.22. Distributed ONOS Clusters Combined Using GRE Tunnel

Each sFlow agent collected the whole traffic for us by pushing the corresponding flows passing through the agents to a central collector. After getting all flows, we calculated  $(\delta, \rho)$  tuple of the normal traffic generated by hping3 application.

In Figure 6.23, we demonstrated a three dimensional plot belonging to the normal situation by observing the traffic within 60 seconds. We evaluated  $\delta$  value and frequency parameter for every second. On this graph, x axis represents the time as a unit of second, y axis represents different  $\delta$  values whereas, the z axis indicates the frequency information. Then, the graph represents how many  $\delta$  values are observed in which frequency for each second. For a normal situation, while small  $\delta$  values are observed at larger frequency for every second, a limited number of big  $\delta$  values are less observed as expected.

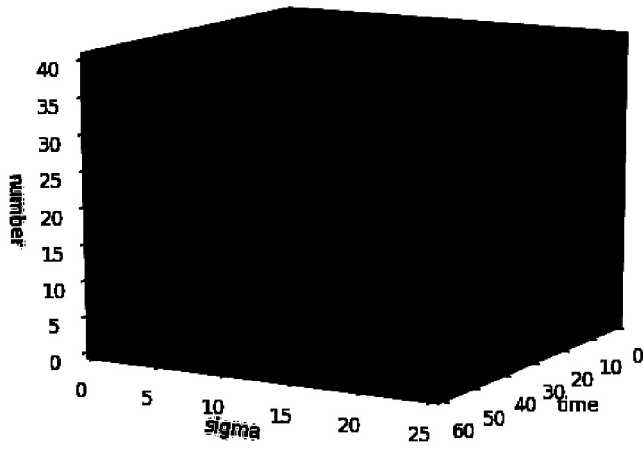


Figure 6.23. Result In Normal Situation on Multiple Clusters

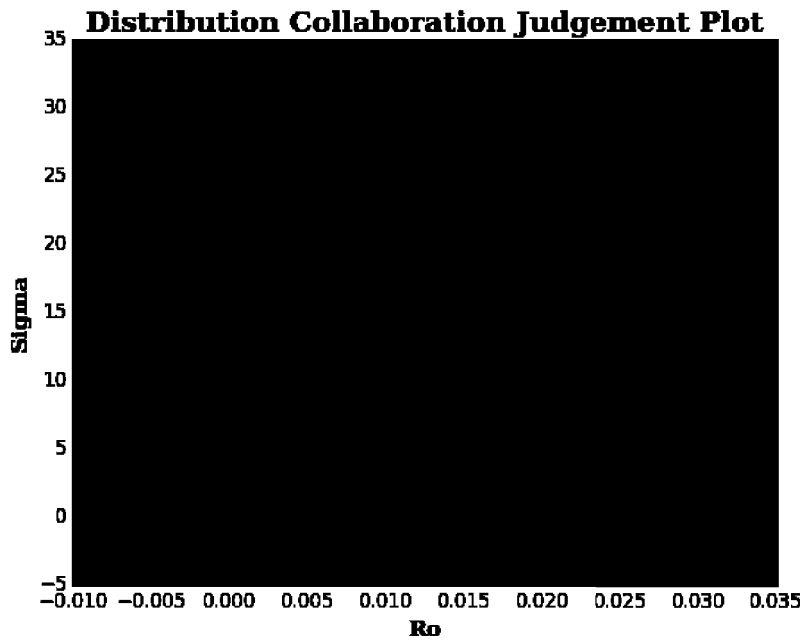


Figure 6.24. DCD Graph For Normal Situation on Multiple Clusters



In Figure 6.24, we illustrated the distribution collaboration judgement plot of normal situation by using matplotlib library of Python. On this graph it is observed that a set of points falls on the area A describing the normal situation having lower  $\delta$  and  $\rho$  value from both of Threshold values. Here, we observed the maximum  $\rho$  parameter as 0.025 Mb/sec. Similar to (Lu and Wang, 2016), most of the nodes does not have more than destination flows in normal situation even though some outliers might appear. Here, it is considered that Threshold  $\delta$  is half of the nodes which are approximately 33. Threshold  $\rho$  might be decided by looking at the total incoming flows of all nodes. According to the TCP flow cache, it is observed that the individual flows as byte values in Figure 6.25.

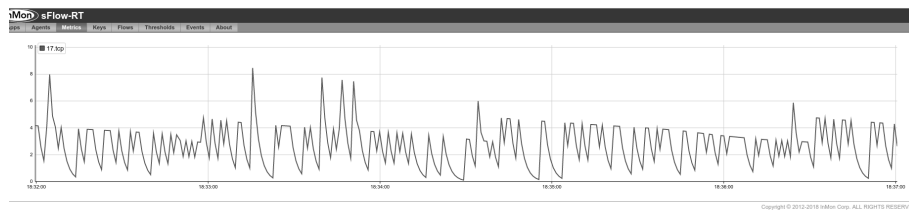


Figure 6.25. TCP Normal Flow Observation on TCP Flow Cache

According to the estimated  $\lambda$  parameter calculated as a mean  $\delta$  value for each node, a Poisson probability distribution graph is given in Figure 6.26.

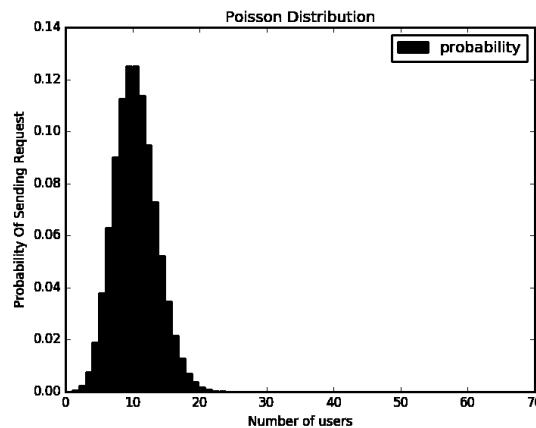


Figure 6.26. Poisson Probabilities For Normal Situation on Multiple Clusters

From the Figure 6.26, we can make a comment on approximately 23 nodes is active on the network. Because of some of the nodes not participating to the network

traffic, these node will have the  $\delta$  value as zero, the most of probabilities is evaluated as zero.

### 6.2.2. Attack Traffic Test

For the second experiment on the distributed network in SDN environment, we organized a Botnet Based DDoS attack by using the total 62 hosts excluding one accepted as a victim node. The attack is performed by sending TCP SYN requests from 62 hosts to a destination of 10.0.0.61. For this experiment, we have given a result of three dimen-

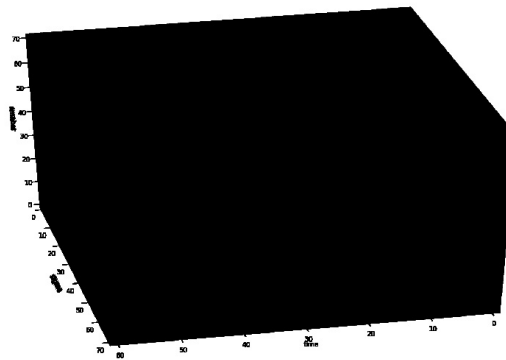


Figure 6.27. Result In Attack Situation on Multiple Clusters

sional graph for the attack situation. From Figure 6.27, it is possible to observe that for the time interval starting from 0 up to 60, the small  $\delta$  values observed at the beginning are directly proportional to the upper  $\delta$  values. As the time goes on, upper  $\delta$  frequency values and also the corresponding  $\delta$  values has been gradually increasing. From 40-th seconds, we observe the  $\delta$  value of 62 with the frequency of 62. As compared to the large  $\delta$  values, the count of  $\delta$  value 1 is also gradually increasing with the upper values inasmuch as when each bot sends SYN packet to the destination, the destination sends a SYN-ACK packet as a response on experimental situation. Due that reason, each one of 62 hosts gain a  $\delta$  value of one.

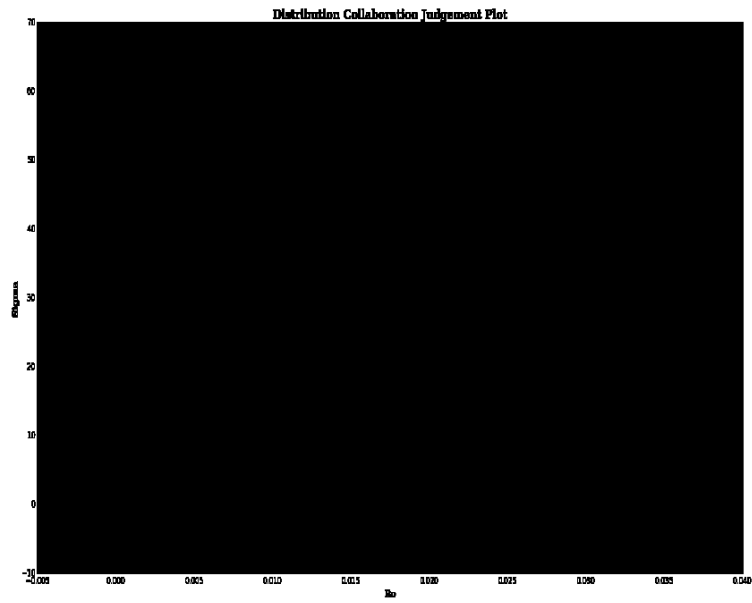


Figure 6.28. DCD Graph For Attack Situation on Multiple Clusters

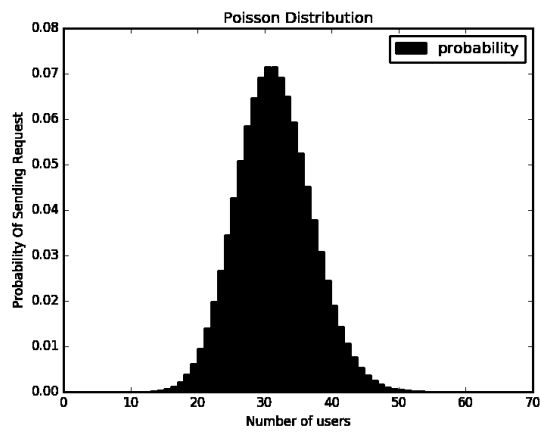


Figure 6.29. Poisson Probabilities For Attack Situation on Multiple Clusters

As indicated in Figure 6.28, the DCD graph demonstrates that the  $\delta$  value of destination flows of the victim falls into the area D which is considered as an attack flow, the  $\delta$  value of bots navigates around 1. The Poisson distribution graph of the attack situation is illustrated on Figure 6.29. From this graph, it is possible to observe that all of the nodes are used to generate attack traffic on the network. Additionally, it is judged that the probability of all users sending requests to a destination is theoretically lower than the probability of half of the nodes sending requests to a destination.

### **6.3. Experiment with Large Network on the Single Controller**

This section contains normal and abnormal traffic experiments performed on one physical computer with the maximum network devices being able to be generated.

#### **6.3.1. Initial Experiment**

In this experiment, normal and attack flow scenarios are implemented on the single SDN controller. In order to observe how many network elements can be generated on the single SDN controller, the initial experiment is done. This experiment is started by generating 15 switches and 135 hosts. In that condition, due to the fact that Mininet and ONOS SDN controller utilize the main physical computer resources, it is observed that the ping reachability of each host has been stopped after 84 hosts send ping request to every other hosts. This condition depends on the processing power of the physical computer. Then, the experiment is gradually repeated by reducing the network size. The second test is executed by Mininet script having 14 switches and 119 hosts. For second trial, it is again observed that the ping reachability of devices has no been continued after 83 hosts can successfully ping to each other. After that, the third test is executed by Mininet script having 13 switches and 117 hosts. For third trial, it is again observed that the ping reachability of devices has not been continued after 70 hosts is successfully ping to each other. By decreasing the network size, the fourth test is executed again with 12 switches and 108 hosts. Then, it is observed that after 84 hosts send ping to each other, the ping request of the remaining devices does not arrive to the other destinations. Then, the fifth test is executed by generating 11 switches and 99 hosts. The network is

generated successfully on the SDN controller. However, similarly the ping reachability of devices has been interrupted after 96 hosts send request to each other. The remaining devices stays unreachable. As the sixth test, the experiment is executed by generating 10 switches and 90 hosts. Then, it is observed that the 64 hosts could send request to the every other host. However, the remaining devices could not deliver the packets to every other destination. As the last trial, the experiment is repeated with 9 switches and 81 hosts. Then, the experiment results were observed successfully. That is to say, every host on the network was able to send every other destination. From this point, it is understood that the maximum counts of network elements of the single controller that will be created is approximately observed as 84 hosts and 9 or 10 switches on the physical computer having the processor speed of 2.2 Ghz.

### 6.3.2. Normal Traffic Experiment

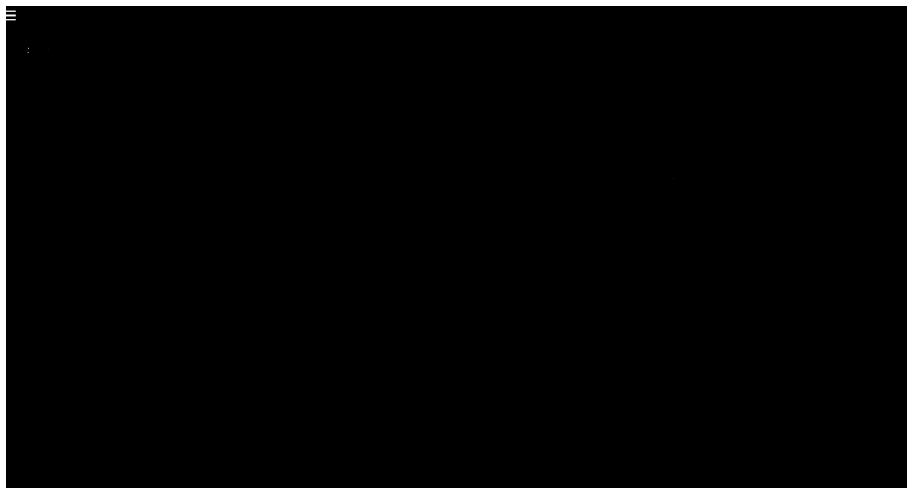


Figure 6.30. Single Controller Topology

As indicated on the Figure 6.30, for the single controller experiments, 81 hosts and 9 switches are created by the script program written in Python. On the topology, 81 hosts are evenly shared among 9 switches. Then, the random normal traffic is generated on the network by using hping3 program. For this traffic, it is provided by hping3 that 24 hosts send requests to random destinations. After that, the traffic is analyzed and the related flows are aggregated and the corresponding tuple information and frequency parameters

are calculated. At this step, it is possible to observe for frequency information, observed  $\delta$  parameters and related time information from the Figure 6.31.

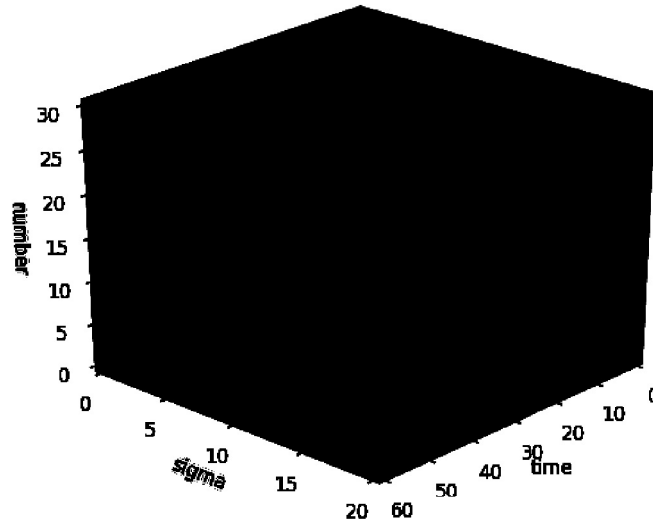


Figure 6.31. Result In Normal Situation On Single Controller

As indicated in the Figure 6.31, for the normal traffic scenario on the single SDN controller, we demonstrated a three dimension plot by observing the traffic 60 seconds. We evaluated the  $\delta$  and observation frequency information for every second. The first dimension represents the time and the second dimension indicates the different observed  $\delta$  values whereas the vertical axis refers to the observation frequency for each observed  $\delta$  value. The graph represents how many  $\delta$  values are observed at which second. It is observed that the small  $\delta$  frequencies are much higher than the big  $\delta$  frequencies. When analyzed the actual  $\delta$  parameters of destination flows, it can be seen from the Figure 6.32 that the  $\delta$  values does not exceed the threshold  $\delta$  value.

From the distribution collaboration plot as demonstrated in the Figure 6.32, the  $\delta$  set of destination flows are accumulated on the area A referring to the normal flow having lower  $\delta$  and  $\rho$  value from both of threshold values. Here, it is considered that Threshold  $\delta$  is half of the nodes which are approximately 40. According to the calculated  $\lambda_i$  value of each node, the Poisson distribution plot is indicated on the Figure 6.33 meaning to the approximately 24 hosts are used for generating random normal traffic on the network.

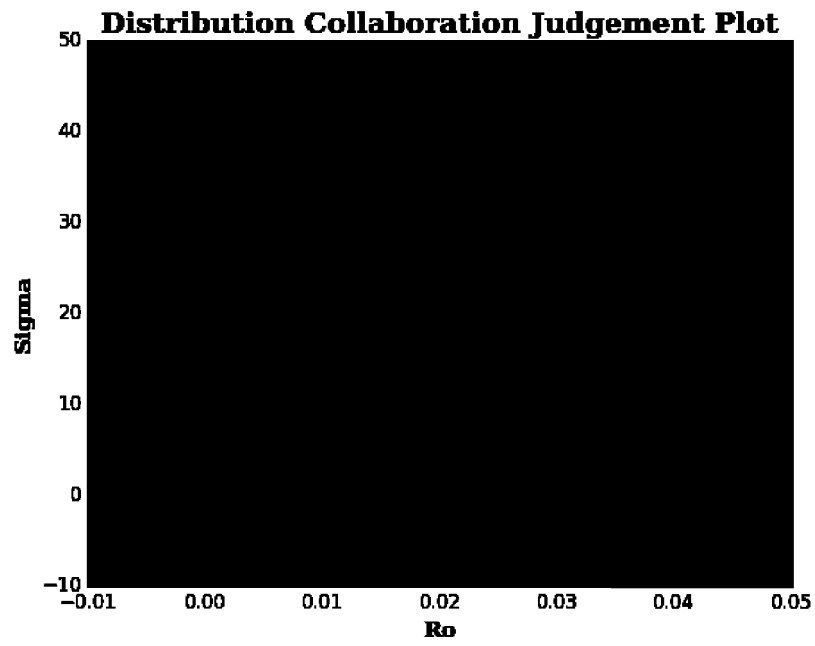


Figure 6.32. DCD Graph For Normal Situation On Single Controller

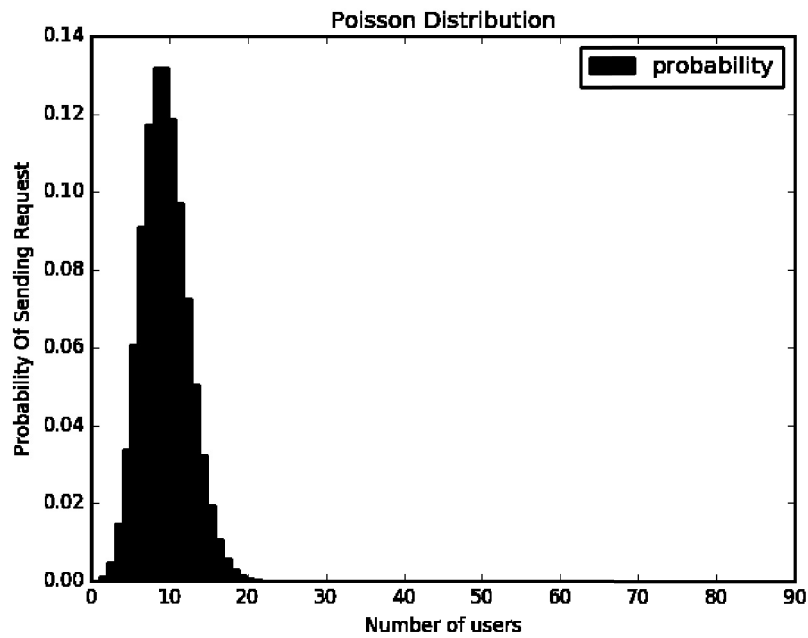


Figure 6.33. Poisson Probabilities For Normal Situation On Single Controller

### 6.3.3. Attack Traffic Experiment

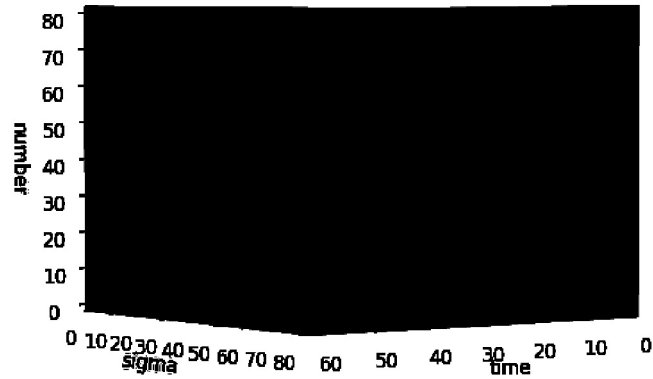


Figure 6.34. Result In Attack Situation On Single Controller

For this experiment, the same topology is used on the single ONOS controller. Then, we generated attack traffic on the network. Then, according to the analyzed traffic information for 60 seconds, a three dimensional plot is indicated in the Figure 6.34, the bigger  $\delta$  values are initially less observed, as the bots join the attack, the bigger  $\delta$  frequency values are mostly observed approximating to the number of 80 hosts participating the botnet based attack. Then, the small  $\delta$  values especially  $\delta$  values equal to 1 are observed incrementally as increased as blue line on the figure because of the SYN ACK responses.

According to the distribution collaboration plot as indicated in the Figure 6.35,  $\delta$  value equals to 80 seems to be an outlier whereas the cumulative points fall into the  $\delta$  value of 1 because of the SYN ACK responses. The outlier falls into the area C referring to the possible attack condition for the detection mechanism. According to the experiment of (Lu and Wang, 2016), the threshold  $\delta$  value is considered as the half of the nodes. Threshold  $\rho$  is calculated and decided by observing the total destination flows on each possible destination.



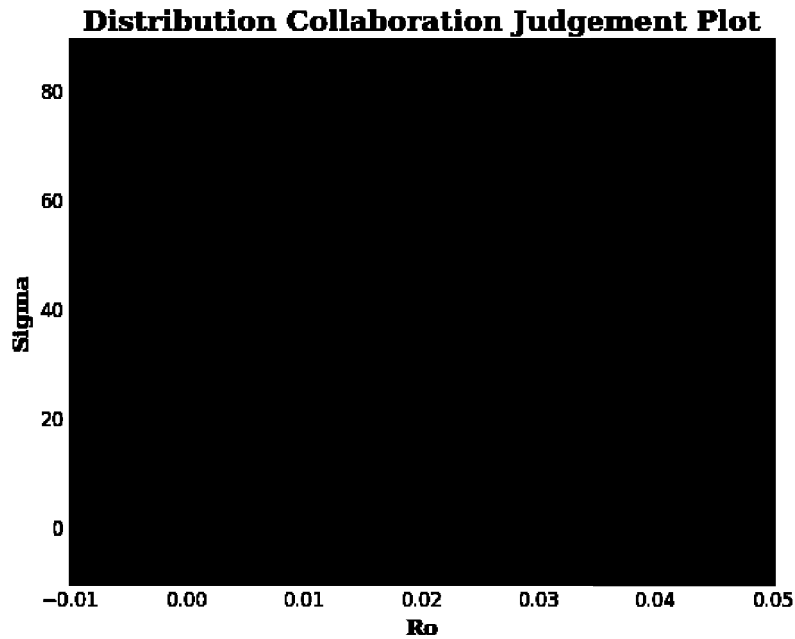


Figure 6.35. DCD Graph Attack Situation On Single Controller

All in all, in this section we tested the designed application depending on the available detection algorithm on single and multiple SDN controllers. Compared to the existing study performed by (Lu and Wang, 2016), we actually observed the similar results on both single and multiple controllers by using the extended network elements. According to the available detection mechanism, while  $\delta$  values of destination flows generally falls into the area A for the random normal traffic,  $\delta$  values of destination flows falls into the area C or D on the distribution collaboration graph for the attack situation. For attack situation and normal situation, it is observed that the detection mechanism also works on the extended networks with single and multiple controllers. The advantage of using multiple controllers was extremely critical. Inasmuch as, we observed that a limited amount of network elements can be created on a single controller while the other controller is useful for the extension mechanism.

## CHAPTER 7

### CONCLUSION

The motivation of this thesis is to implement the detection mechanism on an extended network. We extended the network with GRE Tunnel approach on distributed clusters. Though Botnet Based DDoS attacks may be stemmed from the same subnet, it can also come from the different AS having a different subnet. However, no matter how much we struggle to go outside of the same subnet, we have not achieved to go outside of this scope. This mechanism should be tested with different ASs. As a solution to this, ONOS has SDN-IP application routing traffic over the BGP speakers. It may be a further study to perform the detection method on different ASs. Secondly, the distributed application of ONOS allows writing a BYON application to forward the traffic from one side to the other. Moreover, we used Mininet to generate the network devices of SDN, however, we created splitted topologies by using two Mininet applications running separately on multiple clusters whereas the Mininet Cluster Edition and Philipp Wette's Maxinet offers a single console to manage the distributed devices. Using this approach can be assessed to extend the network to obtain a single console on multiple clusters as a further study.

As a consequence, in this thesis, we performed a similar study based on a detection mechanism proposed by (Lu and Wang, 2016) against Botnet Based DDoS flooding attacks on both of multiple clusters and single cluster. By getting inspired by this study, we implemented the same detection approach to compare the results of this study with the result of the small network study. Then, we developed an application having similar tasks with their application to perform this detection approach. Then, we calculated the related parameters of the detection mechanism to observe whether this study is working on a large network or not. According to the results of this study, the controller and devices generated by Mininet consume the physical computer resources and it has limited capacity to generate a large SDN network effectively. Thus, we extended our topology by using another SDN network on another cluster. By that way, we have successfully extended the network with more devices. According to observed results of the detection mechanism, the results were satisfactory on the larger network and we come across the results resembling the results of the authors on both multiple and single cluster experiments.

## REFERENCES

- Al-Shaer, E., Q. Duan, and J. H. Jafarian (2012). Random host mutation for moving target defense. In *International Conference on Security and Privacy in Communication Systems*, pp. 310–327. Springer.
- Alsmadi, I. and D. Xu (2015). Security of software defined networks: A survey. *computers & security* 53, 79–108.
- Andersen, D. G., H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker (2008). Accountable internet protocol (aip). In *ACM SIGCOMM Computer Communication Review*, Volume 38, pp. 339–350. ACM.
- Blankstein, A., S. A. Erickson, and M. Melara (2013). Mininet clustering.
- Buragohain, C. and N. Medhi (2016). Flowtrapp: An sdn based architecture for ddos attack detection and mitigation in data centers. In *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*, pp. 519–524. IEEE.
- Dao, N.-N., J. Park, M. Park, and S. Cho (2015). A feasible method to combat against ddos attack in sdn network. In *2015 International Conference on Information Networking (ICOIN)*, pp. 309–311. IEEE.
- Hart, J. and A. Koshibe (2016, 7). Sdn-ip architecture - onos - wiki. <https://wiki.onosproject.org/display/ONOS/SDN-IP+Architecture>. Online; Accessed: 2019-05-26.
- Kloeti, R. (2012). Openflow: A security analysis, april 2013. Available: *tip: l/yosemite.ee.ethz.ch/pub/students/2012 ? HS/MA ? 20*.
- Lantz, B., N. Handigol, B. Heller, and V. Jeyekumar (2018, 9). Introduction to mininet. <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#limits>. Online; Accessed: 2019-05-26.

- Lim, S., J. Ha, H. Kim, Y. Kim, and S. Yang (2014). A sdn-oriented ddos blocking scheme for botnet-based attacks. In *2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 63–68. IEEE.
- Lu, Y. and M. Wang (2016). An easy defense mechanism against botnet-based ddos flooding attack originated in sdn environment using sflow. In *Proceedings of the 11th International Conference on Future Internet Technologies*, pp. 14–20. ACM.
- Mendonca, M., S. Seetharaman, and K. Obraczka (2012). A flexible in-network ip anonymization service. In *2012 IEEE international conference on communications (ICC)*, pp. 6651–6656. IEEE.
- Mousavi, S. M. (2014). *Early detection of DDoS attacks in software defined networks controller*. Ph. D. thesis, Carleton University.
- Murtuza, S. and K. Asawa (2018). Mitigation and detection of ddos attacks in software defined networks. In *2018 Eleventh International Conference on Contemporary Computing (IC3)*, pp. 1–3. IEEE.
- Newman, L. H. (2018, 3). Github survived the biggest ddos attack ever recorded. <https://www.wired.com/story/github-ddos-memcached/>. Online; Accessed: 2019-05-26.
- Phaal, P. (2009, 15). Network wide visibility. <https://blog.sflow.com/2009/05/network-wide-visibility.html>. Online; Accessed: 2019-05-26.
- Phaal, P. (2013, 8). Restflow. <https://blog.sflow.com/2013/08/restflow.html>. Online; Accessed: 2019-05-26.
- Phaal, P. (2018, 11). Mininet, onos and segment routing. <https://blog.sflow.com/2018/11/mininet-onos-and-segment-routing.html>. Online; Accessed: 2019-05-26.
- Porras, P., S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu (2012a). A framework for enabling security controls in openflow networks. *ACM (Aug. 2012)*.

- Porras, P., S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu (2012b). A security enforcement kernel for openflow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 121–126. ACM.
- Radware (2017, 3). Ddos attack history | radware security. <https://security.radware.com/ddos-knowledge-center/ddos-chronicles/ddos-attacks-history/>. Online; Accessed: 2019-05-26.
- Ramachandran, A., Y. Mundada, M. B. Tariq, and N. Feamster (2009). Securing enterprise networks using traffic tainting. In *Proc. SIGCOMM*, pp. 1–2.
- Saikia, D. and N. Malik (2014, 9). An introduction to openmul sdn suite. <http://www.openmul.org/uploads/1/3/2/6/13260234/openmul-sdn-platform.pdf>. Online; Accessed: 2019-05-26.
- Sakellaropoulou, D. (2017). A qualitative study of sdn controllers.
- Sanfilippo, S. (2006, 7). hping3(8) - linux man page. <https://linux.die.net/man/8/hping3>. Online; Accessed: 2019-05-26.
- Seyed, F., Y. M. Sekar Vyas, and J. Mogul (2013). Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2013)*.
- Shin, S., V. Yegneswaran, P. Porras, and G. Gu (2013). Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 413–424. ACM.
- Zargar, S. T., J. Joshi, and D. Tipper (2013). A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE communications surveys & tutorials* 15(4), 2046–2069.

# APPENDIX A

## INSTALLING ONOS CONTROLLER

In order to install ONOS controller, it is required to have 2 Core CPU, 2 GB RAM, minimum 10 GB HDD on a single machine. Internet connectivity is required to pull packets from corresponding points like GitHub or other repositories. If it is deployed physically, it must be installed on a Linux operating system. If it is deployed virtually, it is again required to have a Linux operating system in a virtual machine like VirtualBox or VMware. ONOS can also be deployed on MacOS. And also it can run as multiple Docker containers.

**Java 1.8:** The other requirement to install ONOS, because of being Java based platform, Java 1.8 have to be installed and set its environment variables. We can execute the below command to install Java 1.8 on Ubuntu machine.

```
1 [REDACTED]
```

**Curl:** In order to perform web requests by the terminal, we need to install curl application.

```
1 [REDACTED]
```

**Git:** In order to pull some packages from ONOS repositories, it is required to have git installed on the machine. On the physical computers that we have used in this project we have used ONOS 1.12.0 Magpie version. So, it is possible to create a local repository on any machine with the below command.

```
1 [REDACTED]
```

**Maven and Karaf:** ONOS also requires Apache Maven and Karaf, thus a folder called Applications is created on home directory of any machine by using `mkdir Applications` command. Specific ONOS versions are dependent on different Maven and Karaf versions. The ONOS version in that project is 1.12.0, Apache Karaf and Maven versions can be obtained by analyzing `bash_profile` file in the directory of `onos/tools/dev`. Then, in order

to download Maven 3.3.9 and Karaf 3.0.8, tar.gz files are requested on the Downloads directory. So, we change the directory to Downloads folder.

```
1 [REDACTED]
2 [REDACTED]
3 [REDACTED]
```

Then we can listed our files in Downloads directory with ls command on the terminal. After that, we extracted tar.gz files to Applications directory.

```
1 [REDACTED]
2 [REDACTED]
3 [REDACTED]
```

It is necessary to indicate *bash\_profile* to the bashrc file of Ubuntu operating system. Thus, we execute the below commands on the terminal.

```
1 [REDACTED]
2 [REDACTED]
3 [REDACTED]
```

In order to ensure that the JAVA\_HOME and MAVEN environment variables have been set correctly, we execute the below command.

```
1 [REDACTED]
2 [REDACTED]
```

If it is not set, we can set it executing the below command.

```
1 [REDACTED]
2 [REDACTED]
```

There are some environment parameters to install ONOS operating system. These variables are configured according to the karaf and maven requirements of ONOS 1.12.0 version.

```
1 [REDACTED]
2 [REDACTED]
3 [REDACTED]
4 [REDACTED]
5 [REDACTED]
```

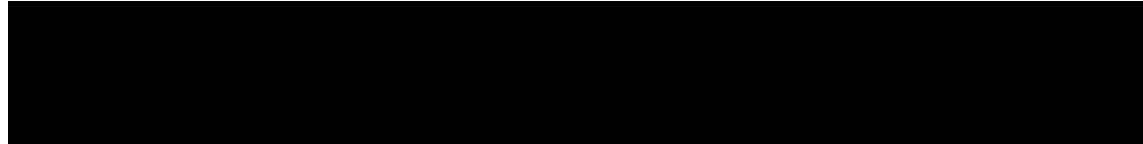
6



This will download onos-1.12.0 packages to opt directory and then untar the files and renames the package as onos.

After setting environment variables of ONOS, we can change the directory to onos with the below command and then install ONOS with skipping tests. Without skipping tests, ONOS installation may fail. It is necessary to build success result from the below command.

1  
2  
3



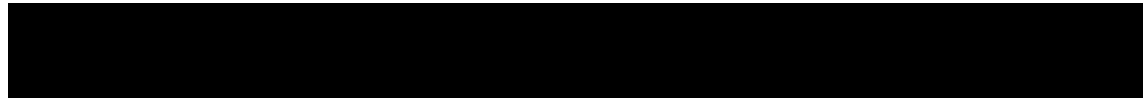
After a while, this command gives a build success result. Then, it is required to install vim editor to change some configurations inside the Apache Karaf.

1



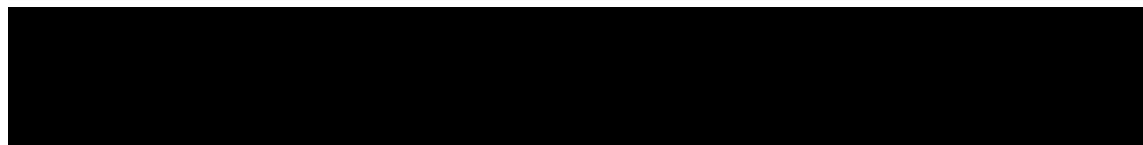
Above command opens an editor window, it is necessary to find featureRepositories and featuresBoot lines on that file. Then, at the end of featuresRepositories, after giving a comma, it is necessary to add the line specified below.

1



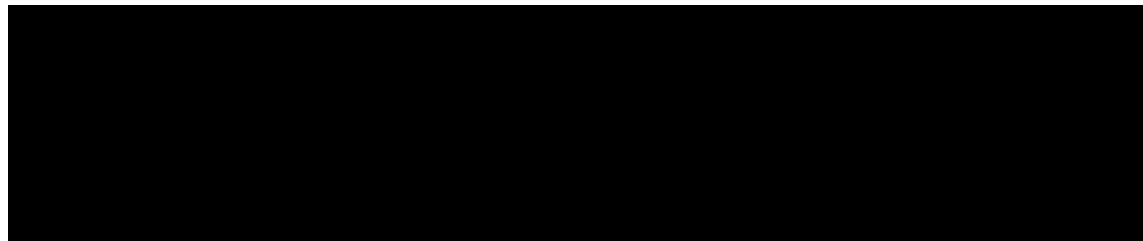
Also, it is necessary to add the below line on this file.

1



ONOS default packages assumes that ONOS gets installed in /opt directory, in order to be able to work onos-form-cluster command, it is necessary to follow below steps on the machine.

1  
2  
3  
4



After that, it is possible to check the status of onos service with the below command.

1





## APPENDIX B

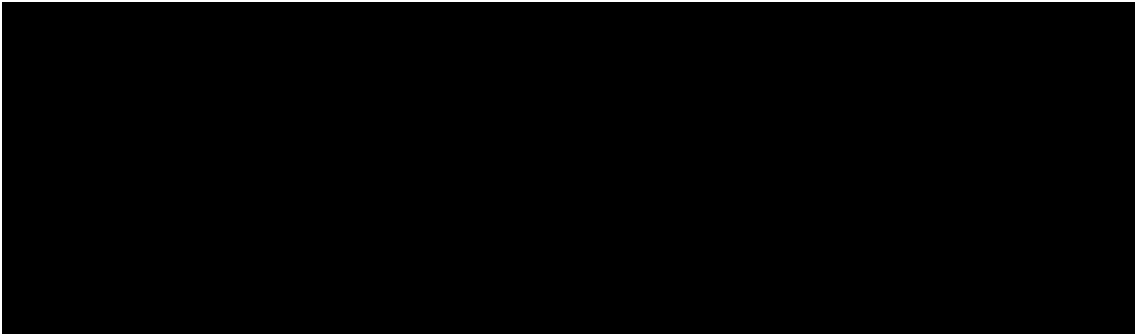
### INSTALLING AND USING MININET

Command syntax is accepted as the following descriptions.

- **\$:** comes before the Linux commands needing to be written on shell prompt.
- **mininet>:** refers to Mininet CLI to be able to run mininet commands.
- **#:** comes before the Linux commands needing to be written on shell prompt with a root privilege.

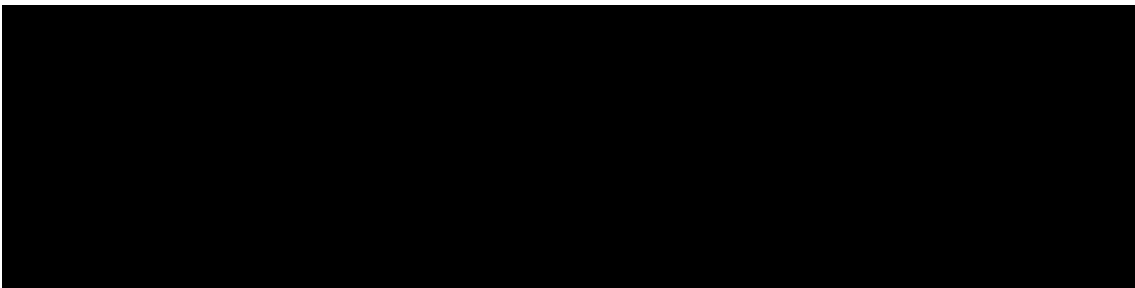
In order to install Mininet, there are two choice, one of them is pre-configured VM image and the second one is a native installation on Ubuntu, in our case, we prefer to install native installation on a physical machine. To install Mininet, a terminal window is opened and the following scripts are executed.

```
1  
2  
3  
4  
5  
6  
7
```



Above command will install all related packages related with OpenvSwitch, OpenFlow Wireshark dissector and POX controller as default. In order to test mininet is installed, sudo mn script can be executed to create minimal topology on CLI. Mininet has some command line utility to perform some actions on the CLI. On Mininet CLI, with the help of a single CLI, it is possible to manage the entire virtual network.

```
1  
2  
3  
4  
5
```



As exemplified with the above commands, link command creates a link among specified devices. It is possible to create a link between two devices or destruct an existing link connected to two devices of the network. Mininet also provides to access network elements. In order to access the elements, xterm application is necessary to install on the machine. Then, it is possible to execute the below command to open a new terminal window of the created network elements. After that, this host can be used to send traffic to other hosts in condition that there is a communication channel between these hosts.

```
1 [REDACTED]
```

-y flag means to install silently on the Linux operating system. If it needs to answer a question during the installation process, it silently answers the question as yes. Then, xterm command can be executed on the Mininet CLI to open a terminal window of the specified device.

```
1 [REDACTED]
```

It is possible to observe the IP address or MAC address information with the below command.

```
1 [REDACTED]
```

In order to test ping reachability of each device, it is possible to execute pingall command to send a ping request from each host to every other hosts.

```
1 [REDACTED]
```

In some cases, it is necessary to crash all networks created, thus we use this command to clean the whole network on Ubuntu terminal.

```
1 [REDACTED]
```

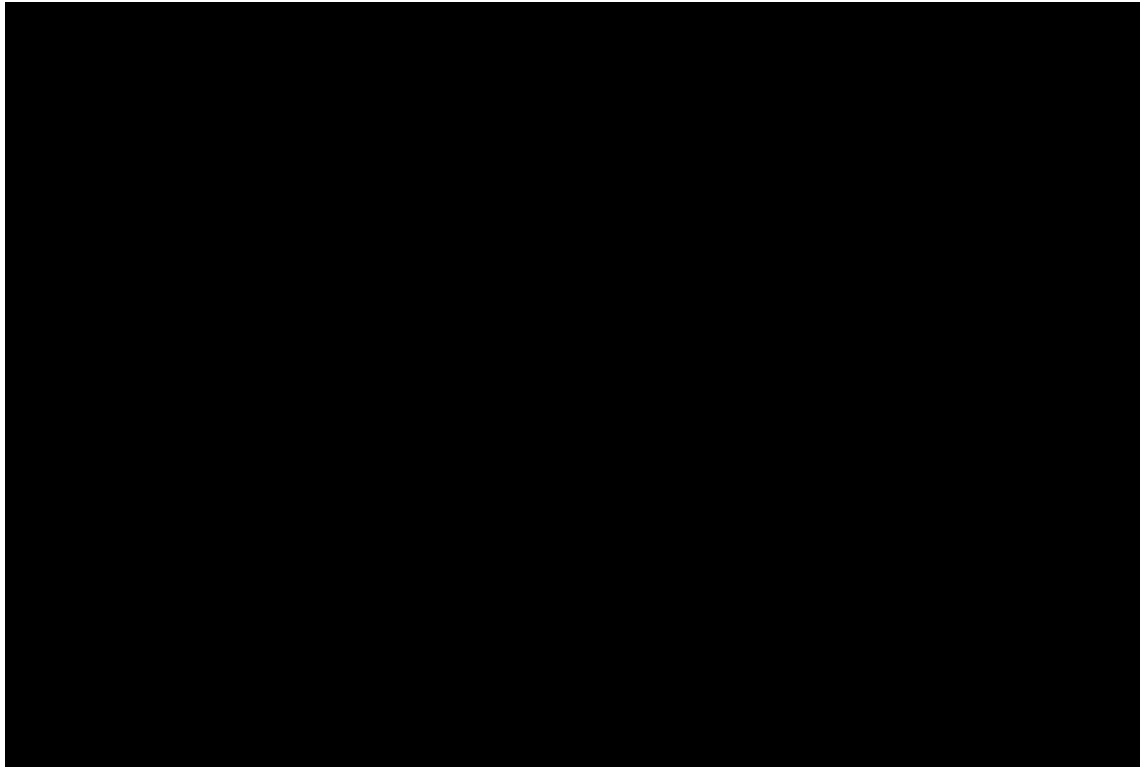
Mininet CLI allows to create the different types of network topologies such as linear topology, triangle topology and so on. It is also possible to create custom network topologies with the help of a python script.

**Linear Topology:** In order to create a linear topology with 4 hosts, and all switches connect with a line to each other and each switch connect to a single host. Below command is used to create a linear topology.

```
1 [REDACTED]
```

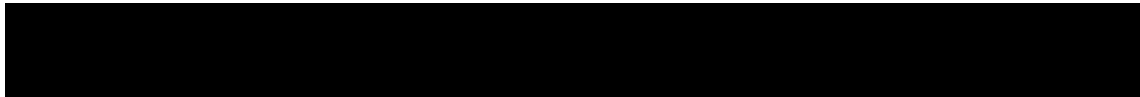
**Custom Topologies:** Custom topologies can be created by using Python API. An example is given below to create custom topologies. Supposing that we have a custom directory in mininet file. Under the custom directory, we can create a new file called as custom.py.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16



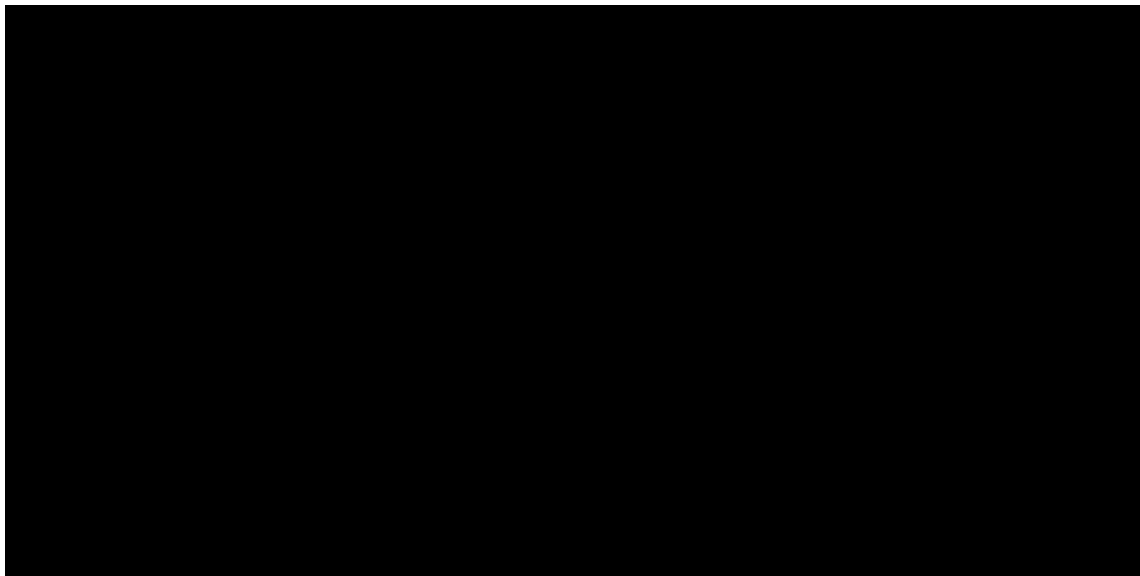
Above topology consists of two switch and two hosts. Left switch connect to left host whereas the right switch connects to right host and left and right switches connect to each other. Then, it is possible to execute this script with the below command.

1

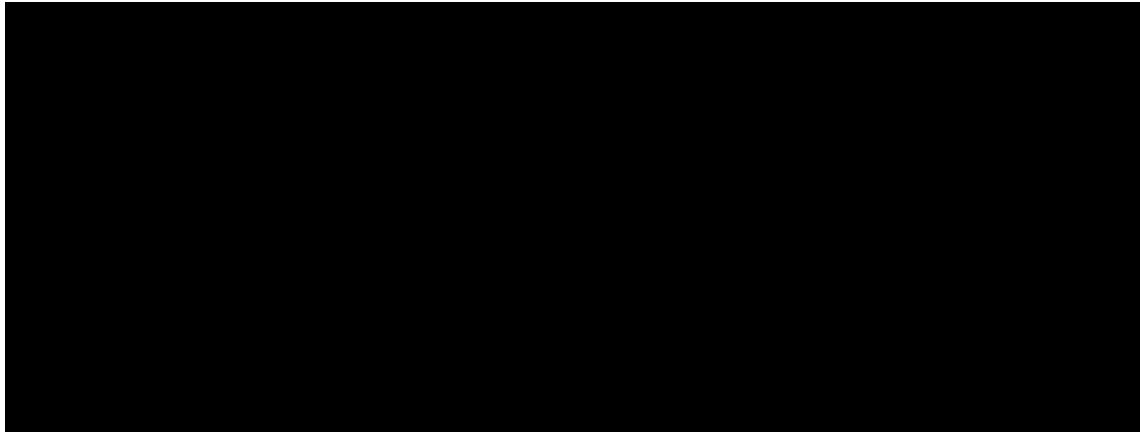


**Creating Network with Referencing A Remote Controller:** Mininet allows us to create the custom topologies and reference a remote SDN controller. In order to create the custom triangle topology, we can create the below python code naming it as onosTopology.py

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12

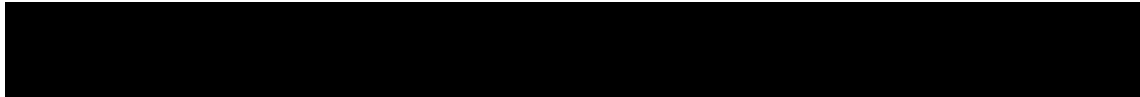


13  
14  
15  
16  
17  
18  
19  
20  
21



When the network is created without referencing a controller, Mininet uses OpenFlow/S-tanford reference controller as default. Otherwise, it is required to specify a controller IP address on the Mininet executing command after changing the directory where the file resides in. Then, we can execute the topology with referenced the controller. If the topology seems on the controller GUI, it means that the topology is successfully created by the Mininet script.

1



# APPENDIX C

## S-FLOW-RT RELATED CONFIGURATIONS

**Installing sFlow-RT:** s-Flow-RT requires Java 1.8 and we can install and start it by following the below commands. Another option is possible to run sflow as a docker container.

```
1  
2  
3  
4
```

**Installing sFlow Agents on Switches:** It can be installed manually from the Linux terminal with below command. sFlow-RT works on 6343 port and localhost as default. It can be changed by defining environment variables before executing Mininet as specified below.

```
1  
2
```

**Installing sFlow Agents Automatically by sflow.py file:** Another option to install agents into devices use sflow.py script ships with the sFlow-RT. It can be configured for custom topologies and executing with mininet command. Mininet can be started with the below command to generate network elements and install sflow agents on the switch simultaneously.

```
1
```

**Defining Flow Cache on s-Flow-RT:** It is necessary to install curl application on the Linux operating system. The below command will instruct the collector running on 192.168.1.101 to construct a flow cache.

```
1
```

2



In order to get existing flow definitions from the sflow-RT, the below command can be executed.

1



In order to retrieve all TCP flows from TCP flow cache defined previously, the below command returns all TCP flows as JSON objects according to the TCP flow cache definition.

1



The below python script defines TCP flow cache and retrieves all flow records continuously from the TCP cache.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23



24

25



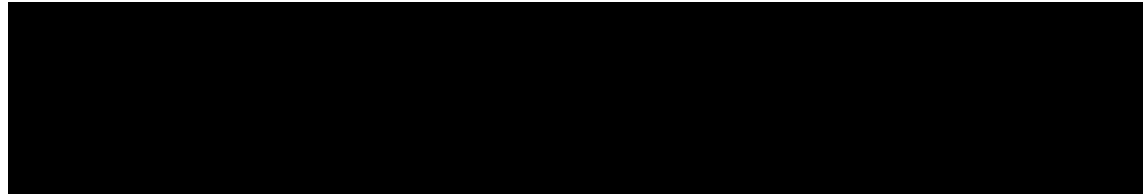
Sample output of the above python script would be indicated as below.

1

2

3

4



## APPENDIX D

### BASH SCRIPT APPLICATION AND ONOS CELL CONFIGURATION

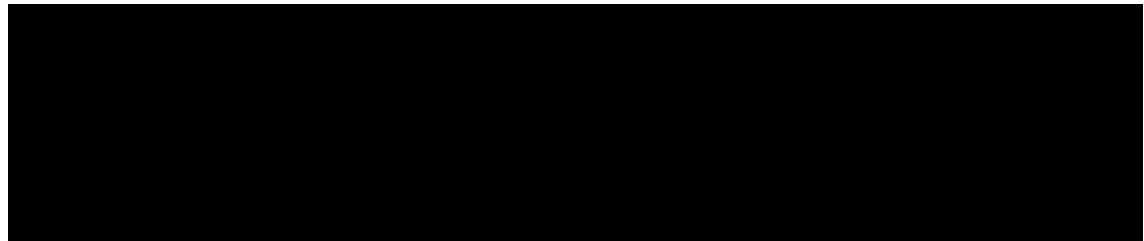
In order to run this application on Ubuntu machine, we have defined alias named as onosrun. Our run.sh script resides in /onos directory, thus we added the below line to the bashrc file.

1



**Create A Cell Definition File:** cell command allows us to execute a cell script defined in \$ONOS\_ROOT/tools/test/cells directory. This contains a set of environment variables like the default address used by ONOS controller and a set of applications need to be executed for every execution of ONOS. The content of a sample cell definition used by the bash script application is illustrated below.

1  
2  
3  
4  
5



After created a cell file named tutorial, it can be executed on the shell prompt as indicated below.

1





# APPENDIX E

## HPING3 INSTALLATION, USAGE AND DDOS ORGANIZATION

**Hping3 Installation:** For the Linux operating system environment, hping3 can be installed with the following instructions to perform different tests on different hosts on the network.

```
1 [REDACTED]
2 [REDACTED]
```

### Hping3 General Usage Examples:

```
1 [REDACTED]
```

This type of usage will send a NULL packet having no TCP flags with a TCP header on port number 0.

```
1 [REDACTED]
```

The above usage will send a certain number of packets to the target.

```
1 [REDACTED]
```

The above usage sends packets with certain size as bytes.

```
1 [REDACTED]
```

The above usage will wait for a specific time before sending the next packet.

```
1 [REDACTED]
```

It will send a packet to target by setting specified flag.

```
1 [REDACTED]
```

This will arrange hping3 running mode. If it is chosen 1, ICMP packets are sent to the target.

```
1 [REDACTED]
```

Above command will send as n number of ICMP packets to the targets.

```
1 [REDACTED]
```

It will send a TCP packet with SYN flag to target machine on the specified destination port number.

1

```
[REDACTED]
```

Hping3 will send a TCP packet with SYN flag by enhancing port numbers starting from the specified range.

1

```
[REDACTED]
```

In the above example, TCP packet will send to the target machine without specifying the actual source address.

1

```
[REDACTED]
```

Hping3 can send each packet with different spoofed source IP address with `-rand-source` alias. IP spoofing is performed with hping3, however, in this case, reply packets does not return to actual source address. It will return to randomly spoofed IP address.

1

```
[REDACTED]
```

Hping3 can send each packet to different destination addresses with `-rand-dest` alias. In this case, it is required to use an interface name like `eth0`.

1

```
[REDACTED]
```

By using this option, custom TTL value can be set for outgoing packets.

1

```
[REDACTED]
```

Hping3 can set each TCP packet to a custom window size.

1

```
[REDACTED]
```

Hping3 can send a packet with a bad TCP/UDP checksum.

1

```
[REDACTED]
```

A limited number of packets are sent with SYN flag from randomly chosen source port to the target machine on the destination port.

1

```
[REDACTED]
```

For the above example, hping3 send TCP packet with SYN flag from hided source IP address with a dedicated source port to the target machine. However, while sending each packet, t amount of time is waited before sending the next packet.

**Organizing ICMP Flood Attack with hping3:** In order to organize an ICMP flood attack, hping3 can be used as specified below. It will continuously send ICMP packets from spoofed y.y.y.y address to x.x.x.x address.

1



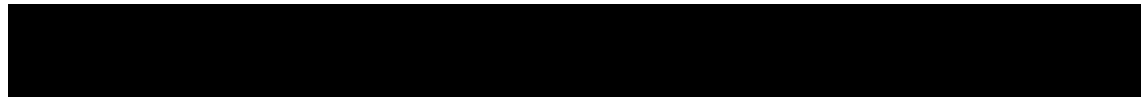
**Organizing SYN Flood Attack with hping3:** In order to organize an SYN flood attack, hping3 can be used as specified below. It will continuously send SYN packets from spoofed y.y.y.y address to x.x.x.x address on port number 80.

1



Another example can be given for SYN flood attack with `-fast` alias to send each packet with 10 second interval. This usage is more appropriate to perform the attack. Inasmuch as, the `-flood` option will destruct the network within a shorter time. For performing attack detection test, this usage is more convenient to get the results.

1



2

The main difference among the above commands is that the first one organizes a DoS attack from a spoofed source IP address to x.x.x.x destination whereas the other script organizes a DDoS attack, because every request is sent from different spoofed source IP addresses to the same destination.

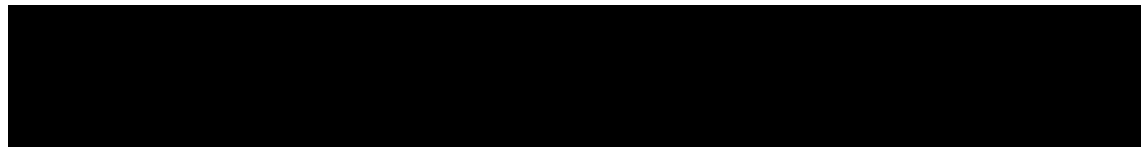
**Organizing UDP Flood Attack with hping3:** In order to organize an UDP flood attack, hping3 can be used as specified below. It will continuously send UDP packets from spoofed y.y.y.y address to x.x.x.x address on port number 6234.

1



**Observing the Replies Coming From Target Machine:** It is possible to observe the coming reply packets after sending request to the target machine. Below script exemplifies how to observe the reply packets of target machine with 192.168.1.107 IP address.

1



2

The above response will include a set of arguments. Table E.1 indicates the meaning of the response parameters.

Table E.1. The meaning of Response Parameter

<b>Response Parameter</b>	<b>Meaning</b>
len	The size of the incoming packet
ip	The IP address of the target system
ttl	The lifetime of the packet
DF	Active fragmentation bit.
Id	A unique identifier belonging to the IP packet.
Sport	Source port that the packet has been sent.
Flags	Active TCP flags.
seq	Sequence number of the packet.
win	Window size of the packet.
rtt	Round trip time as millisecond.

## APPENDIX F

### AN EXAMPLE OF CONSTRUCTING GRE TUNNEL

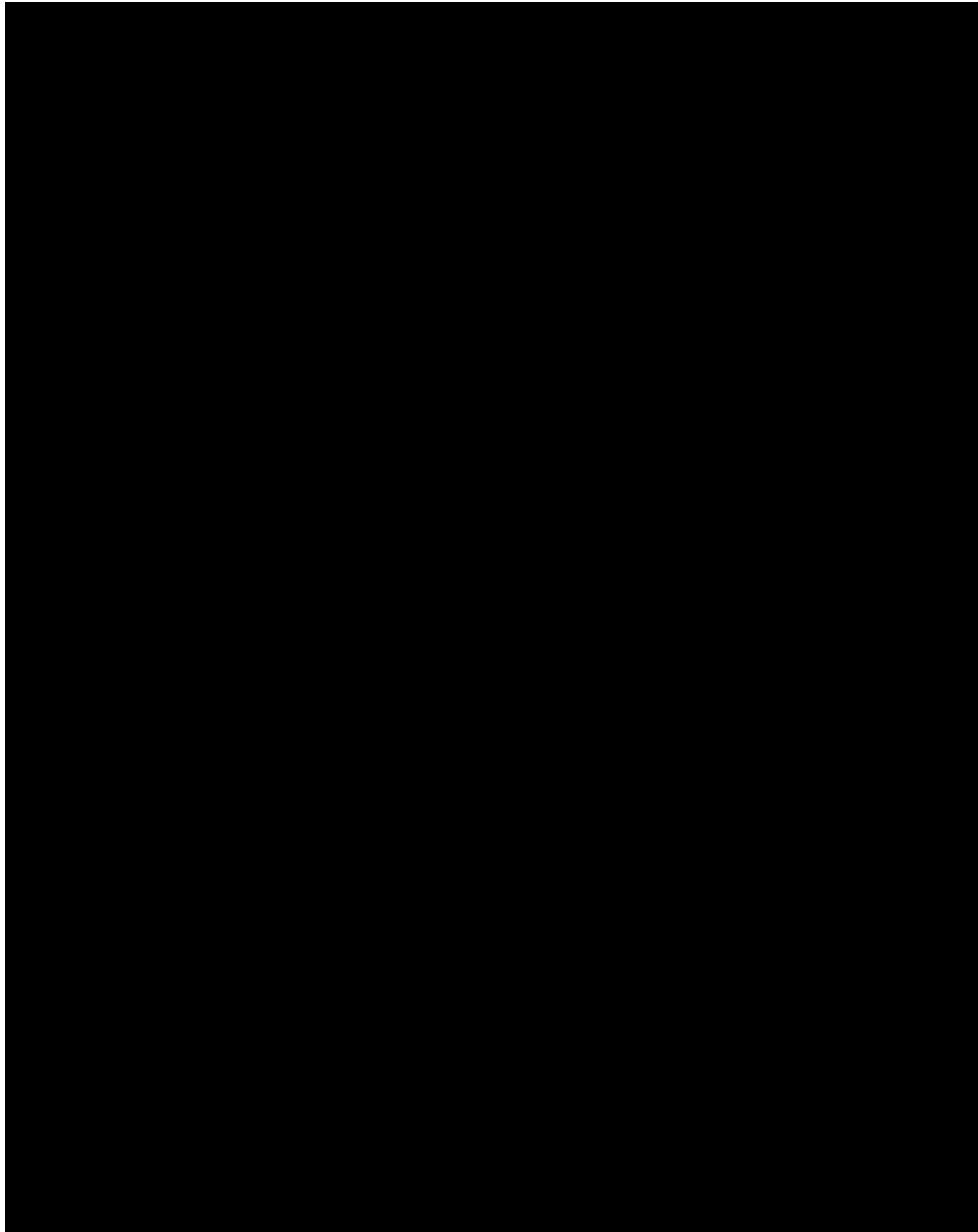
The first step to construct the combined topology is to split the combined topology into two small partitions. For Figure 5.5, the topology of Machine-1 can be considered as the first topology while the topology of Machine-2 can be considered as the second. Suppose that the IP address of the first machine is 192.168.1.101, the IP address of the second machine is 192.168.1.104. Then, it is possible to write two mininet scripts for different machines.

**The configuration of First Machine:** The below code demonstrates how to configure s1 switch of Machine-1. First of all, SingleSwitchTopo is created to create hosts, switches and links between them. Then, the main method creates a topology instance of this class. After that, a controller is created and referenced to a remote controller. Then, created topology instance and the controller is given to the Mininet object to return the related network instance. The network object has all information of the network elements and the controller. If there is an existing tunnel settings for s1 switch, a new terminal window is opened for s1 and then the following scripts are executed to firstly remove the existing tunnel and then create a new tunnel between s1 and s2.

1  
2  
3  
4

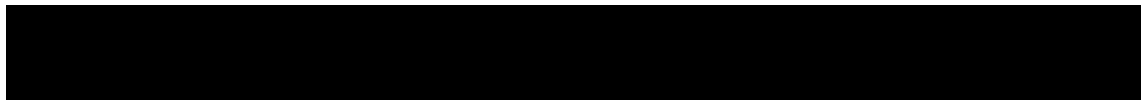
1  
2  
3  
4  
5  
6  
7  
8  
9

10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36



**The configuration of Second Machine:** The below code demonstrates how to configure s2 switch of Machine-2. Here, the configuration is similar with the previous configuration. However, for this time, Machine-2 IP address is firstly given in the script as the following command. Because, for this time, our local address will become 192.168.1.104.

1



2

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

