



ADAMA SCIENCE AND TECHNOLOGY UNIVERSITY

SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTING

**Performance Analysis of Software Defined Networking Distributed
Controller in Data center Networks**

**A Thesis Submitted in the Partial Fulfillment of the Requirements for
the Degree of Master of Science in Software Engineering**

By

Worku Muluye Wubet

February 2017



ADAMA SCIENCE AND TECHNOLOGY UNIVERSITY

SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTING

**Performance Analysis of Software Defined Networking Distributed
Controller in Data center Networks**

By

Worku Muluye Wubet

Advisor

Mesfin Abebe (Ph.D.)

February 2017

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other work and that all source of materials used for the thesis has been duly acknowledged.

Declared by:

Name: Worku Muluye

Signature: _____

Date: _____

This thesis has been submitted for examination with my approval as university advisor

Confirmed by advisor:

Name: Mesfin Abebe (Ph.D.)

Signature: _____

Date: _____

February 2017

Name and signature of Members of the Examining Board

<u>Name</u>	<u>Title</u>	<u>Signature</u>	<u>Date</u>
_____	Advisor	_____	_____
_____	Chairperson	_____	_____
_____	External Examiner	_____	_____
_____	Internal Examiner	_____	_____
_____	Department Head	_____	_____
_____	School Dean	_____	_____

Acknowledgment

First of all, I would like to thank Almighty GOD who makes things possible and helps me in every aspect throughout my life.

Next, I would like to express my sincere gratitude to my advisor and PG coordinator Dr. Mesfin Abebe for the continuous support for my thesis, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of the research and writing of the thesis. Besides my advisor, I would like to thank Dr. Vuda his guidance helped me in the time of writing this thesis proposal.

Next, I would like to thank you Dr. Mesfin my advisor and PG coordinator, Ato Mohamed computing department head, Ato Zellalem computer science and engineering chair and computing department lab supervisors Ato Henok and Ato Wondie for their permission to use network and security SIG lab and high-performance computers to do this thesis.

Last but not list, I would like to thank my families, teachers, friends and others who helped me for the accomplishment of this thesis. Thank you all.

Table of Contents

Acknowledgment	i
List of Figures	vi
List of Tables	viii
List of Abbreviations and Acronyms	ix
Abstract	xii
CHAPTER ONE	1
1. Introduction.....	1
1.1. Background of the Study.....	1
1.2. Statement of the Problem.....	3
1.3. Research Questions	4
1.4. Objectives.....	4
1.4.1. General Objective	4
1.4.2. Specific Objective	4
1.5. Scope and Limitation of the Study.....	5
1.6. Significant of the Study.....	5
1.7. Research Methodology	6
1.7.1. Literature Review.....	6
1.7.2. Research Emulation and Experimental Tools.....	6
1.8. Thesis Organization	9
CHAPTER TWO	10
2. Literature Review.....	10
2.1. Overview of Traditional Network.....	10
2.1.1. Traditional Network Architecture	10
2.1.2. Limitation of Traditional Network.....	12
2.2. History of SDN	14
2.2.1. Definitions of SDN	15

2.3.	SDN Architecture.....	15
2.3.1.	Application Layer	16
2.3.2.	Control Layer	17
2.3.2.1.	OpenFlow Protocol	17
2.3.3.	Infrastructure Layer.....	18
2.4.	Why SDN Network Needed?	19
2.4.1.	Open Networking Foundation.....	19
2.5.	SDN OpenFlow Controllers.....	19
2.5.1.	SDN Distributed OpenFlow Controllers.....	21
2.5.1.1.	Distributed SDN Controllers in a Data center Network	22
2.6.	Related Works.....	23
CHAPTER THREE.....		27
3.	Research Tools and Environment Setup	27
3.6.	Comparison of SDN Distributed Controllers.....	27
3.6.1.	OpenDaylight	28
3.6.2.	Onix.....	29
3.6.3.	HyperFlow	30
3.6.4.	ONOS.....	31
3.6.4.1.	ONOS Architecture.....	31
3.6.4.2.	ONOS Controller Clustering.....	33
3.7.	Emulator and Simulator Tool Comparison	33
3.7.1.	OMNeT++.....	34
3.7.2.	EstiNet.....	34
3.7.3.	Mininet	35
3.8.	Cbench	36
3.9.	Working Environment Setup Challenges.....	37
3.10.	Research Environment Setup Main Processes	37
CHAPTER FOUR.....		39
4.	Proposed Solution	39
4.1.	Performance Analysis Procedures	39

4.2.	Adapting ONOS Controller in a Data center Network	40
4.3.	Measuring ONOS Controller Performance	41
4.3.1.	Single ONOS Controller	41
4.3.2.	Two ONOS Controllers	41
4.3.3.	Three ONOS Controllers	42
4.4.	SDN Network Topologies.....	42
4.4.1.	Tree Topology.....	43
4.4.2.	Customized Topology	43
4.5.	SDN OpenFlow Controller Performance Parameters	44
4.5.1.	Throughput.....	44
4.5.2.	Latency.....	44
4.5.3.	TCP Bandwidth.....	45
CHAPTER FIVE.....		46
5.	Performance Analysis and Evaluation	46
5.1.	ONOS Controller GUI	46
5.1.1.	Single ONOS Controller GUI	46
5.1.2.	Two ONOS Controller GUI.....	47
5.1.3.	Three ONOS Controller GUI.....	48
5.2.	Designed Tree and Customized Topology GUI.....	48
5.3.	ONOS Controller Performance Analysis	50
5.3.1.	Throughput Analysis.....	50
5.3.1.1.	Single ONOS Controller Throughput Analysis	51
5.3.1.2.	Two ONOS Controller Throughput Analysis	52
5.3.1.3.	Three ONOS Controller Throughput Analysis	54
5.3.2.	Latency Analysis.....	58
5.3.2.1.	Single ONOS Controller Latency Analysis	58
5.3.2.2.	Two ONOS Controller Latency Analysis	60
5.3.2.3.	Three ONOS Controller Latency Analysis	61
5.4.	ONOS TCP Bandwidth between Hosts	65
5.5.	Evaluation	66
CHAPTER SIX		68

6. Conclusion and Future Work68

6.1. Conclusion68

6.2. Future Work69

References.....70

Appendices.....76

Appendix A76

Appendix B77

Appendix C83

List of Figures

Figure 1-1: Vertically integrated control plane and data plane	1
Figure 1-2: Separated control plane and data plane.....	2
Figure 1-3: Research procedures.....	9
Figure 2-1: The traditional network architecture.	12
Figure 2-2: History of Programmable Networking over the Past 20 Years.....	15
Figure 2-3: The advantage of SDN network over the traditional network.	16
Figure 2-4: SDN architecture.	16
Figure 2-5: OpenFlow Switch architecture.	18
Figure 2-6: Logically centralized but physically distributed OpenFlow controller architecture.	22
Figure 3-1: OpenDaylight Architecture.	29
Figure 3-2: Onix architecture.	30
Figure 3-3: HyperFlow Architecture.....	31
Figure 3-4: Distributed ONOS Architecture.	32
Figure 3-5: Three common topologies in Mininet.	35
Figure 3-6: Mininet CLI with switches, hosts, links and three remote ONOS controllers.	36
Figure 4-1: Performance analysis procedures.	39
Figure 4-2: Single ONOS controller.	41
Figure 4-3: Two ONOS Controllers in a cluster.	42
Figure 4-4: Three ONOS Controllers in a cluster.	42
Figure 4-5: Comparison of data center network architecture and tree topology.	43
Figure 4-6: Sample customized topology source code GUI view.	44
Figure 5-1: Single ONOS controller with tree topology.....	46
Figure 5-2: Two ONOS controller with tree topology.....	47
Figure 5-3: Three ONOS controller with tree topology.....	48
Figure 5-4: The designed customized topology graphical view with 30 switches and hosts.	49
Figure 5-5: The designed tree topology graphical view with 30 switches and hosts.....	49
Figure 5-6: Cbench running in throughput mode.....	50
Figure 5-7: Single ONOS throughput results per tree topology.	51
Figure 5-8: Single ONOS throughput results per customized topology.	52
Figure 5-9: Two ONOS controller throughput results per tree topology.....	53
Figure 5-10: Two ONOS controller throughput results per customized topology.	54
Figure 5-11: Three ONOS controller throughput results per tree topology.....	55
Figure 5-12: Three ONOS controller throughput results per customized topology.	56
Figure 5-13: Single, two and three ONOS controller average throughput results per tree and	

customized topology.	57
Figure 5-14: Cbench running in latency mode.....	58
Figure 5-15: Single ONOS controller latency results per tree topology.....	59
Figure 5-16: Single ONOS controller latency results per customized topology.....	59
Figure 5-17: Two ONOS controller latency results per tree topology.....	60
Figure 5-18: Two ONOS controller latency results per customized topology.	61
Figure 5-19: Three ONOS controller latency result per tree topology.	62
Figure 5-20: Three ONOS controller latency results per customized topology.	63
Figure 5-21: Single, two and three ONOS controller average latency results per tree and customized topologies.	64
Figure 5-22: Two ONOS controller TCP bandwidth results between 20 hosts.....	65
Figure 5-23: ONOS Controller TCP bandwidth between hosts.....	66

List of Tables

Table 2-1: Members of Open Networking Foundation.....	19
Table 2-2: The basic requirements to build SDN controllers.	20
Table 2-3: List of centralized OpenFlow SDN controllers.	21
Table 2-4: Identified gaps in the literature review.	26
Table 3-1: Comparison of distributed OpenFlow controllers.	28
Table 3-2: Comparison of OMNET++, EstiNet, and Mininet.	34
Table 3-3: List of Mininet commands to run the topology and to connect with the controller.	36
Table 3-4: Cbench OpenFlow controller benchmarking tool running options.	37
Table 5-1: Single ONOS controller throughput results in flow per second.....	51
Table 5-2: Two ONOS controller throughput results in flow per second.....	53
Table 5-3: Three ONOS controller throughput results in flow per second.....	55
Table 5-4: Single ONOS controller latency results in response per ms.....	58
Table 5-5: Two ONOS controller latency results in responses per ms.....	60
Table 5-6: Three ONOS controller latency results in responses per ms.....	62
Table 5-7: Distributed ONOS controller TCP bandwidth between hosts in GB/s.....	65

List of Abbreviations and Acronyms

API	<u>A</u> pplication <u>P</u> rogramming <u>I</u> nterface
ARP	<u>A</u> ddress <u>R</u> esolution <u>P</u> rotocol
ARPANET	<u>A</u> dvanced <u>R</u> esearch <u>P</u> roject <u>A</u> gency <u>N</u> etwork
ASIC	<u>A</u> pplication <u>S</u> pecific <u>I</u> ntegrated <u>C</u> ircuit
AT&T	<u>A</u> merican <u>T</u> elephone & <u>T</u> elegraph
BGP	<u>B</u> order <u>G</u> ateway <u>P</u> rotocol
CLI	<u>C</u> ommand <u>L</u> ine <u>I</u> nterface
CPC	<u>C</u> onsistent <u>P</u> olicy <u>C</u> omposition
CPU	<u>C</u> entral <u>P</u> rocessing <u>U</u> nit
D-CNTL	<u>D</u> istributed <u>C</u> ontroller
DCPA	<u>D</u> istributed <u>C</u> ontrol <u>P</u> lane <u>A</u> rchitecture
DCP-GK	<u>D</u> ynamic <u>C</u> ontroller <u>P</u> rovisioning <u>P</u> roblem
DCP-SA	<u>G</u> reedy <u>K</u> napsack and <u>S</u> imulated <u>A</u> nneling
DHA	<u>D</u> istributed <u>H</u> opping <u>A</u> lgorithm
DHA-CON	<u>D</u> istributed <u>H</u> opping <u>A</u> lgorithm <u>C</u> ontroller
DPID	<u>D</u> elivery <u>P</u> oint <u>I</u> dentifier
DISCO	<u>D</u> istributed <u>S</u> DN <u>C</u> ontrol
EBay	<u>E</u> lectronic <u>B</u> ay
ForCES	<u>F</u> orwarding and <u>C</u> ontrol <u>E</u> lement <u>S</u> eparation
IETF	<u>I</u> nternet <u>E</u> ngineering <u>T</u> ask <u>F</u> orce
INT	<u>I</u> nteger
GB	<u>G</u> iga <u>B</u> yte
GUI	<u>G</u> raphical <u>U</u> ser <u>I</u> nterface
IP	<u>I</u> nternet <u>P</u> rotocol
ISIS	<u>I</u> ntermediate <u>S</u> ystem to <u>I</u> ntermediate <u>S</u> ystem
LACP	<u>L</u> ink <u>A</u> ggregation <u>C</u> ontrol <u>P</u> rotocol
LFB	<u>L</u> ogical <u>F</u> unctional <u>B</u> lock
LXC	<u>L</u> inux <u>C</u> ontainer
MB	<u>M</u> ega <u>B</u> yte
MAC	<u>M</u> edia <u>A</u> ccess <u>C</u> ontrol <u>A</u> ddress
MCDM	<u>M</u> ulti- <u>C</u> riteria <u>D</u> ecision <u>M</u> aking
NBAPI	<u>N</u> orthbound <u>A</u> pplication <u>I</u> nterface
NETCONF	<u>N</u> etwork <u>C</u> onfiguration <u>P</u> rotocol

NetConf	<u>N</u> etwork <u>C</u> onfiguration
NetFlow	<u>N</u> etwork Traffic <u>F</u> low
NFV	<u>N</u> etwork <u>F</u> unction <u>V</u> irtualization
NOS	<u>N</u> etwork <u>O</u> perating <u>S</u> ystem
NTT	<u>N</u> ippon <u>T</u> elegraph and <u>T</u> elephone
NVP	<u>N</u> etwork <u>V</u> irtualization <u>P</u> latform
ODL	<u>O</u> pen <u>D</u> ay <u>l</u> ight
OMNeT++	<u>O</u> pen <u>M</u> ultipurpose <u>N</u> etwork <u>T</u> estbed
ONOS	<u>O</u> pen <u>N</u> etworking <u>O</u> perating <u>S</u> ystem
OVSDB	<u>O</u> pen <u>v</u> Switch <u>D</u> atabase
OVSC	<u>O</u> pen <u>v</u> Switch <u>C</u> ontroller
OFP	<u>O</u> pen <u>F</u> low <u>P</u> rotocol
ONF	<u>O</u> pen <u>N</u> etworking <u>F</u> oundation
OS	<u>O</u> perating <u>S</u> ystem
OSGI	<u>O</u> pen <u>S</u> ervice <u>G</u> ateway <u>I</u> nitiative
OSPF	<u>O</u> pen <u>S</u> hortest <u>P</u> ath <u>F</u> irst
PC	<u>P</u> ersonal <u>C</u> omputer
PCEP	<u>P</u> ath <u>C</u> omputation <u>E</u> lement <u>P</u> rotocol
PDV	<u>P</u> acket <u>D</u> elay <u>V</u> ariation
PG	<u>P</u> ost <u>G</u> raduate
POX	<u>P</u> lain <u>O</u> ld <u>X</u> ML
PorGFE	<u>P</u> rogrammable <u>G</u> eneric <u>F</u> orwarding <u>E</u> lement
RAM	<u>R</u> andom <u>A</u> ccess <u>M</u> emory
R&D	<u>R</u> esearch & <u>D</u> evelopment
REST	<u>R</u> epresentational <u>S</u> tate <u>T</u> ransfer
RESTCONF	<u>R</u> epresentational <u>S</u> tate <u>T</u> ransfer <u>C</u> onfiguration
REUSETAG	<u>R</u> epresentational <u>S</u> tate <u>T</u> ransfer <u>T</u> ag
RESTful	<u>R</u> epresentational <u>S</u> tate <u>T</u> ransfer <u>F</u> ull
RSPAN	<u>R</u> emote <u>S</u> witched <u>P</u> ort <u>A</u> nalyzer
S-CNTL	<u>S</u> ingle <u>C</u> ontroller
SDMN	<u>S</u> tate <u>D</u> ebt <u>M</u> anagement <u>N</u> etwork
SIG	<u>S</u> pecial <u>I</u> nteresting <u>G</u> roup
SNMP	<u>S</u> imple <u>N</u> etwork <u>M</u> anagement <u>P</u> rotocol
SDN	<u>S</u> oftware <u>D</u> efined <u>N</u> etworking
SDNLAB	<u>S</u> oftware <u>D</u> efined <u>N</u> etworking <u>L</u> aboratory

SMP	<u>S</u> witch <u>M</u> igration <u>P</u> roblem
SPAN	<u>S</u> witched <u>P</u> ort <u>A</u> nalyzer
SSL	<u>S</u> ecure <u>S</u> ockets <u>L</u> ayer
STR	<u>S</u> tring
TB	<u>T</u> era <u>B</u> yte
TCP	<u>T</u> ransmission <u>C</u> ontrol <u>P</u> rotocol
TCAM	<u>T</u> ernary <u>C</u> ontent <u>A</u> ddressable <u>M</u> emory
TSL	<u>T</u> ransport <u>L</u> ayer <u>S</u> ecurity
UDP	<u>U</u> ser <u>D</u> atagram <u>P</u> rotocol
VM	<u>V</u> irtual <u>M</u> achine
VND	<u>V</u> isual <u>N</u> etwork <u>D</u> escription
VLAN	<u>V</u> irtual <u>L</u> ocal <u>A</u> rea <u>N</u> etwork
WAN	<u>W</u> ide <u>A</u> rea <u>N</u> etwork
ZTE	<u>Z</u> hejiang <u>T</u> ongxiang <u>E</u> lectronic <u>I</u> ndustry <u>C</u> orporation

Abstract

A computer network is a critical issue in our day to day activity; however, this day it works under several problems. To solve these problems OpenFlow-based programmable Software-Defined Network (SDN) is released. OpenFlow is a protocol for SDN which vertically separates the control plane from the data plane of the network devices. SDN differs from the traditional networking architecture, where control planes and the data planes exist in the same networking devices. In SDN the controllers are the brains of the network that manages and controls the network devices. Data center network is one of the application areas that required successful integration of distributed OpenFlow controllers by making the network more consistent. However, the need for high-performance SDN controllers' increases with the increasing of network devices in the data center network. SDN distributed controller is a controller which can increase or decrease the number of controllers according to the change of traffics. Therefore, in this thesis, we have presented a performance analysis in both throughput and latency parameters for the recent SDN distributed ONOS controllers.

The aim of this thesis has been to perform a study and performance analysis of SDN distributed controllers in a data center networks by considering two designed topologies. Initially, has been performed a detailed study on SDN distributed controllers and selected the best ONOS controller. Then, we have created three ONOS controllers in a cluster using LXC and designed tree and customized topologies using python in Mininet. Then we have tested, analyzed and evaluated these three ONOS controllers' throughput, latency, and TCP bandwidth. We have presented the throughput, latency, and TCP bandwidth results of single, two and three ONOS controllers. The single ONOS controller has 60665.22 to 82566.85 flow/s throughput, 62.89 to 94.45 responses/ms latency and 13.5 GB/s TCP bandwidth. Two ONOS controller has 105564.30 to 118911.20 flow/s throughput, 117.53 to 147.91 responses/ms latency and 18.25 GB/s TCP bandwidth. Three ONOS controller has 216816.80 to 229388.90 flows/s throughput, 297.61 to 245.32 responses/ms latency and 19.4 GB/s TCP bandwidth. Single, two and three ONOS controllers per tree topology has high throughput and high response per ms than customized topology. Finally, based on these results have been adapted distributed ONOS controller in a data center networks.

Keywords: Software-defined networking, distributed SDN controller, OpenFlow, Open networking operating system, Data center networks

CHAPTER ONE

1. Introduction

1.1. Background of the Study

Computer network changes the way we do business and the way we live. During this time our day to day activity performs or processes through the network and it makes computers and their users capable of being connected together. This facilitates sharing of resources and information between the users. Networks are a critical element for companies and institutions because it radically increases business activities. Therefore, network administrators are struggling to ensure high network availability. According to [1] communications between remote parties can be achieved through a process called networking, with the connection of computers, media, and networking devices. When a number of devices connected to the Internet grow every day and more and more data are transferred from device to device, scaling up the traditional network gets increasingly difficult. In order to meet the performance, network expansion takes place and much of the efforts go towards configuring each switch and routers individually even for changes in a smaller segment of a LAN that may contain hundreds of nodes [2].

In the traditional networking, most network functionality is implemented in the device; i.e., switch, router, security device, application delivery controller. In addition, within the device, most of the practicality implemented in hardware, for example, an application specific integrated circuit (ASIC). The growth of ASIC functionality is under the control of the provider of the devices, each device is patented and configured individually. Also to delivery and to change task management consumes a lot of time and error prone [3]. It is hard to configure the network according to predefined rules and to reconfigure it to respond to faults, loads, and changes. Since there is an excessive number of switches and routers and a set of protocols that determine the optimal path from source to destination by looking into Ethernet and IP headers. Because the traditional networks are vertically combined i.e. the control plane and data planes are bundled or integrated together. Typically in networking devices, the control plane is developed and maintained by vendors only [4].

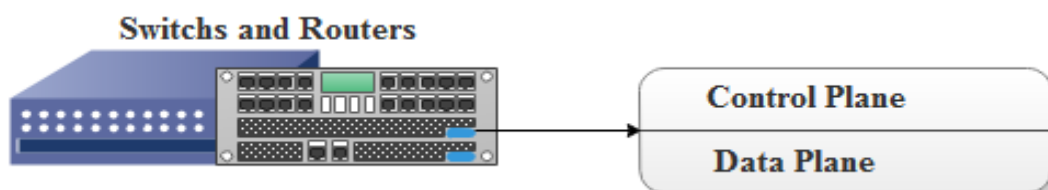


Figure 1-1: Vertically integrated control plane and data plane [5].

This makes the network devices to be vendor dependence or depend on a specific vendor. In order to increase the performance, scalability and user satisfaction; the network is upgraded and at this time each individual device is affected in traditional networks. To solve these network limitations a new networking architecture was invented which are called Software Defined Networking (SDN).

SDN is a new network architecture that controls the entire network devices centrally. The concept was introduced by [6], a professor at Stanford University and is based on defining a model where all switches move the capacity of the decision to a central element, to a controller. SDN decouples control and forwarding plane and makes the network directly programmable and the infrastructure to be abstracted for applications and network services [7]. In SDN controller, for a single change in the network, the network configurations are changed only at central or some specific controller rather than configuring individual network devices. Because the control plane separated from the data plane enables the control plane to control several devices and data plane to forward the data according to the predefined forwarding table.

ONF [8] is responsible for standardizing and promoting SDN and defines it as in networking devices, there exist three planes. Data plane is the hardware where packet forwarding takes place. Control plane is a software where all network logics and intelligence performed. Management plane typically it is a portion of control plane that used for network monitoring and controlling purposes. We have taken a real world analogy to describe and to understand the concept of SDN such as cars and trucks, roads, traffic light and the drivers. Due to this cars and trucks are packets on the network. And the road acts as data plane of SDN and the traffic light controllers are the control plane of the network. And also the driver in the car is the application of SDN network architecture.

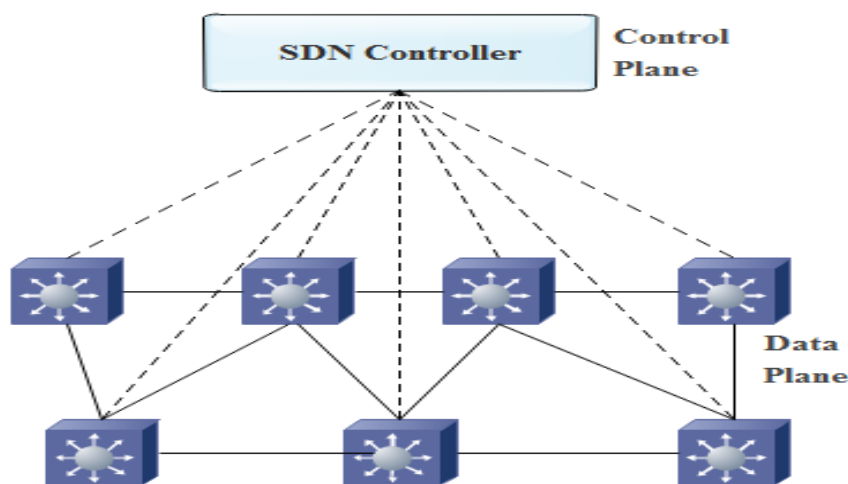


Figure 1-2: Separated control plane and data plane [4].

As [7] SDN is an emerging architecture that is dynamic, manageable, cost-effective adaptable and scalable network. SDN makes the network devices simpler while giving network administrators better control over the network and increasing network performance. SDN uses OpenFlow protocol (OFP)

method to enable the control plane to communicate with the data plane [9]. In SDN the controller estimates a routing path for that flow on the basis of the global view and then sends the forwarding rule of that routing path to all switches through a secure channel. When those SDN switches receive the forwarding rules, they update their flow tables. Then forward the received flow according to the corresponding flow rules resulting from the controller [10].

Distributed means something shared among multiple systems in the same or in different locations. Distributed network is a computer network that is spread over diverse networks and offers a single data communication network, which can be managed together or separately by each network. SDN distributed controller is a controller that enables concurrent and robust network policy implementation. Distributed controller consisting of a collection of controllers that controllers the entire network and provide high network availability. It controls the connected forwarding devices in a master or slave way. In distributed controller, the master controller can read and write the OpenFlow switches. And the slave controller can only read the switches and wait as standby [11].

Data center network is a network that contains network devices, servers, security devices, power supplier and environment controllers. In a traditional network, it is very difficult to configure millions of devices individually, manage the network and to scale the network. Also, it is difficult to secure control, troubleshoot and maintain the network. So, SDN plays a great role to solve these problems in a data center networks [12,13]. However, centralized SDN controller in a data center network faces performance, scalability and availability problems. In SDN, when the number of devices is increasing difficult to use a single controller to control the entire forwarding devices. Due to the capacity limitation of a single controller and a lot of flows, one controller is insufficient to control the entire network. Depend on these open issues we have started to find the research problems related to SDN controllers.

1.2. Statement of the Problem

SDN can be the future of networking, there is no question about it. SDN has emerged as a popular architecture for managing large scale networks including service providers, enterprises, data centers and cloud networks. The key principle of SDN is the centralized control plane architecture, which allows the network to be programmed by applications running centrally, enabling easier management and faster innovation. However, many of these large-scale data center networks consist of several hundreds of servers interconnected with thousands of switches and routers in network topology, which cannot easily be controlled by a single centralized controller [14].

SDN is moving network configurability from individual switches and routers to logically centralized network controller. However, this SDN-based centralized network control has performance, scalability, availability, and single point failure problems of the controllers. In large data center

networks, this can build to an unacceptable level of delay, degrading network performance and introducing single points of failure. If we use a single controller, the controller gives poor performance, with the combined effect causing scalability and load balance problems.

The solution for several performance problems in large and growing data center networks is to push more intelligence to the data plane or change to distributed control plane architecture. The distributed controller can be the main solution to the performance, scalability, and availability of the control plane. In order to solve the problems of centralized SDN controller, much different architecture has been proposed recently, such as the multi-core controller, the logically centralized controller and the completely distributed controller [4, 10, 15].

An SDN distributed controller architecture needs performance analysis using multiple controllers in a cluster. In addition, this SDN distributed controller requires evaluation by running a different number of the controller using key performance parameters. Performance analysis and evaluation of distributed ONOS controller in a data center network are not investigated. Also, from small to large data center network needs high performance, scalable, and high available SDN distributed controllers. The performance of SDN distributed controller needs study and evaluation in a data center networks. We have motivated to do this thesis, because SDN network is new, interesting, and it is a current hot research area. We have formulated the following research questions for these problems.

1.3. Research Questions

This proposed research answered the following research questions.

- ❖ How to create and run SDN distributed controllers?
- ❖ How to measure and analysis the performance of SDN distributed controllers?
- ❖ How the SDN distributed controllers control and manage the data plane?

In order to answer these research questions needs a stated general objective and specific objectives that are achieved by this thesis. So, we have defined the objective of this thesis as follow.

1.4. Objectives

1.4.1. General Objective

The main objective of this research is to analysis the performance of SDN distributed controller in data center networks by considering a different number of controllers and two designed topologies.

1.4.2. Specific Objective

The specific objectives that are explicitly achieved by the proposed system are listed below:

- ❖ Study SDN-related literature.
- ❖ Compare and select the SDN distributed controllers and emulator tools.
- ❖ Create SDN distributed controller in a cluster.

- ❖ Create tree and customized network topologies.
- ❖ Measure the throughput, latency and TCP bandwidth of SDN distributed controllers.
- ❖ Analysis and evaluate a different number of distributed controllers in a cluster.
- ❖ Adapt distributed ONOS controller in a data center networks.
- ❖ Disseminate of finding and results.

The goals of this thesis defined; to achieve these specific and general objectives needs to be specified the research area and outlined the limitations of the research. According to this, we have mentioned the delimitation and limitation of this thesis to meet the defined goals.

1.5. Scope and Limitation of the Study

The area or the scope that is considered in the proposed research briefly introduces the performance analysis of SDN distributed controllers in data center networks. The study does not include software defined cloud, WI-Max, WI-Fi and Mobile networks. In addition due to in unavailability of physical laboratory and the high cost of real network devices; this research does not include physical configuration of the network. This needs within this scope and limitation what benefit will provide and for whom. Depend on this we have listed the importance of this work next to here.

1.6. Significance of the Study

An SDN distributed controller in data center network is a decentralized repository, which allows adding thousands hosts, servers and networking devices. The SDN distributed controller performance analysis and using of SDN distributed controller in a data center networks can offer the following benefits:

- ❖ Save data center network operational time
- ❖ Provide better data center network management
- ❖ Make SDN controllers highly scalable
- ❖ Improve SDN distributed controllers performance
 - Increase throughput
 - Decrease latency
- ❖ Improve the bandwidth size of distributed controllers
- ❖ High availability of data center network
- ❖ Identify best network topology for data center networks.
- ❖ The study will create awareness for researcher's persons who want to do in SDN networks.

Beneficiaries of the research:

A person or organizations that obtain direct or indirect benefit from this thesis are:

- ❖ Network administrators
- ❖ SDN Network operators

- ❖ ICT experts
- ❖ Distributed controller developers
- ❖ Distributed controller testers

This thesis provides these direct or indirect significances to the target beneficiary. In order to solve the research problems, to answer research questions, to achieve the objective within the specified area and to get the target significance of this thesis needs a research way. So we have designed the following methods to conduct this study.

1.7. Research Methodology

In this section, we have described the research methodology that was followed to conduct this study. Identifying a suitable research design and methodology related to our study is the key concept that considered before performing any research work. We have specified suitable research methodologies and associated design for this research. In this research, used three parameters, 45 scenarios, experiment each scenario, analysis and evaluate the results. Therefore, this research is experimental research.

1.7.1. Literature Review

In order to have an understanding of the problem, it is very important to review several kinds of literature that have been conducted in the field of SDN controller so far. Due to this, related literature such as proceeding papers, recent books, articles, journals articles and additional useful materials from the internet should be reviewed. This helps to understand and investigate the underlying concepts, principles, methodologies, techniques and tools of the various methods. This can be adapted, manipulated and used for synthesizing and gaining a new perspective on what has been done in previous researches in the area and what can be done in this research.

1.7.2. Research Emulation and Experimental Tools

In order to analysis SDN distributed controller performance efficiently building a large experimental test bed using emulator software's is the best option. Therefore, we used emulation, experimentation, controller, benchmarking, programming language and command tools. Because using these tools is more reliable, efficient, accurate and effective than a manual process. The tools that we have used in this thesis progress are:

Operating System: In this research, we used windows operating system host machine and Ubuntu operating system for guest machines. Because, ONOS SDN controller, cbench, and Mininet are compatible with desktop or server-based Linux operating system. In addition, the research environment setup by using commands in the Ubuntu terminal.

Commands: In this research, most of the task done using Ubuntu, ONOS controller commands,

Mininet emulator commands, cbench commands and network commands. We used a lot of commands throughout the thesis process such as to install tools, configure the network, set the research environment, set the path, pull the features, integrate tools, measure the controller performance and to distributed the controller.

ONOS Controller: We used ONOS SDN controller because it is newly released, scalable and available than the other SDN distributed OpenFlow controllers. In the environmental setup, ONOS requires 4 or more CPU, 4 GB or more memory, and 20 GB or more hard disk size. Especially, ONOS requires a study on its performance and it is a hot research area. So, we have selected and used ONOS OpenFlow controller as the best controller to do our research [16].

Mininet Emulator: We have selected Mininet SDN network emulator to emulate the designed network topology. Because Mininet is python based SDN network emulator and allows to design single, tree, linear and customized network topologies using python language. In addition, Mininet provides real Linux operating systems, real applications, real network devices, and real configurations commands and support OpenFlow protocols [17].

Cbench: It is the most commonly used tool for benchmarking SDN OpenFlow controllers, which operated in either latency or in throughput mode. Cbench used to test the controller for throughput and latency performance by running it in throughput and latency mode respectively. So, we have selected and used cbench OpenFlow controller throughput and latency benchmarking tool [18].

Oracle VM VirtualBox: We used oracle VM VirtualBox to install Linux operating system. Virtualbox is suitable for low-performance computers in terms of memory, CPU, and computer processing speed; so we used VirtualBox [19].

Linux Container: Linux container (LXC) is operating system-level virtualization method for running multiple isolated Linux containers on a control host using a single Linux kernel. In order to distribute SDN ONOS controller using multiple VM is very complex and requires a high processor, large size memory and hard disk of a computer. We have used LXC to solve these problems since it helps us to create three LXC containers in a single VM [20].

Python: Python is an open source, interpreted and multi-platform programming language [21]. We used python to design the SDN network topology because Mininet network emulator is python based. This helps us to create hosts, links, OpenFlow switches, networks and to connect the designed topology with distributed ONOS controllers at the remote.

Microsoft Office Word, Excel, Edraw Max, Google chrome etc.: We have used Microsoft office word to write and prepare the document of the thesis. We have used Excel to analysis and to show graphically the throughput, latency, and TCP bandwidth results of ONOS controllers. We used Edraw max, to design architectures, models and to draw high-quality image; because Edraw max contains network and other templates. We used chrome to show graphically the designed network topology

and distributed ONOS controller since ONOS controller has GUI.

OpenFlow Protocols: The OpenFlow protocol is a rule that separates control plane from the data plane. In our case, we have used OpenFlow protocol, because of most of SDN controllers, Mininet and cbench are support it and different network device companies produce OpenFlow devices. So, we used OpenFlow protocols to separate control and forwarding layer [6].

OpenvSwitch: We used OpenFlow switches for packet forwarding purpose at the data plane of the data center. In addition, different versions of OpenvSwitch are supported by Mininet and ONOS controller, so OpenvSwitch is suitable to switch to do this research [7].

SDN network topology: In this research, we have designed tree and customized topologies using python language since the Mininet emulator tool is python based [22].

Iperf: It is a Mininet command tool that measures TCP bandwidth between hosts that is connected to the controller. We have used this tool to measure the amount of TCP bandwidth between hosts that is controlled and managed by distributed ONOS controller in a cluster. This enables to use ONOS controller in a data center networks based the result of distributed ONOS controller bandwidth and performance. So, we have selected it as the best tool to use in this thesis.

Parameters: Parameters are the main factors to do this thesis. To do performance analysis and to use ONOS controller in a data center networks, we have used and measured two key performance parameters such as throughput and latency and TCP bandwidth between hosts.

Finally, the main method of this research literature review and related work study. This enabled us to understand SDN concepts, SDN and traditional network architecture differences, SDN controller concepts, the current state of SDN distributed controllers in data center networks and to find the gaps. So in the next chapter, we have performed literature review.

1.7.3. Performance analysis and evaluation

In this thesis, after measured the throughput latency and TCP bandwidth of ONOS controller we have performed performance analysis and evaluation of the results.

1.7.4. Adapt distributed ONOS controller in a data center networks

An SDN distributed ONOS controller was designed for internet service provider networks. Datacenter networks need highly scalable, available controller so we have used ONOS controller in a data center networks.

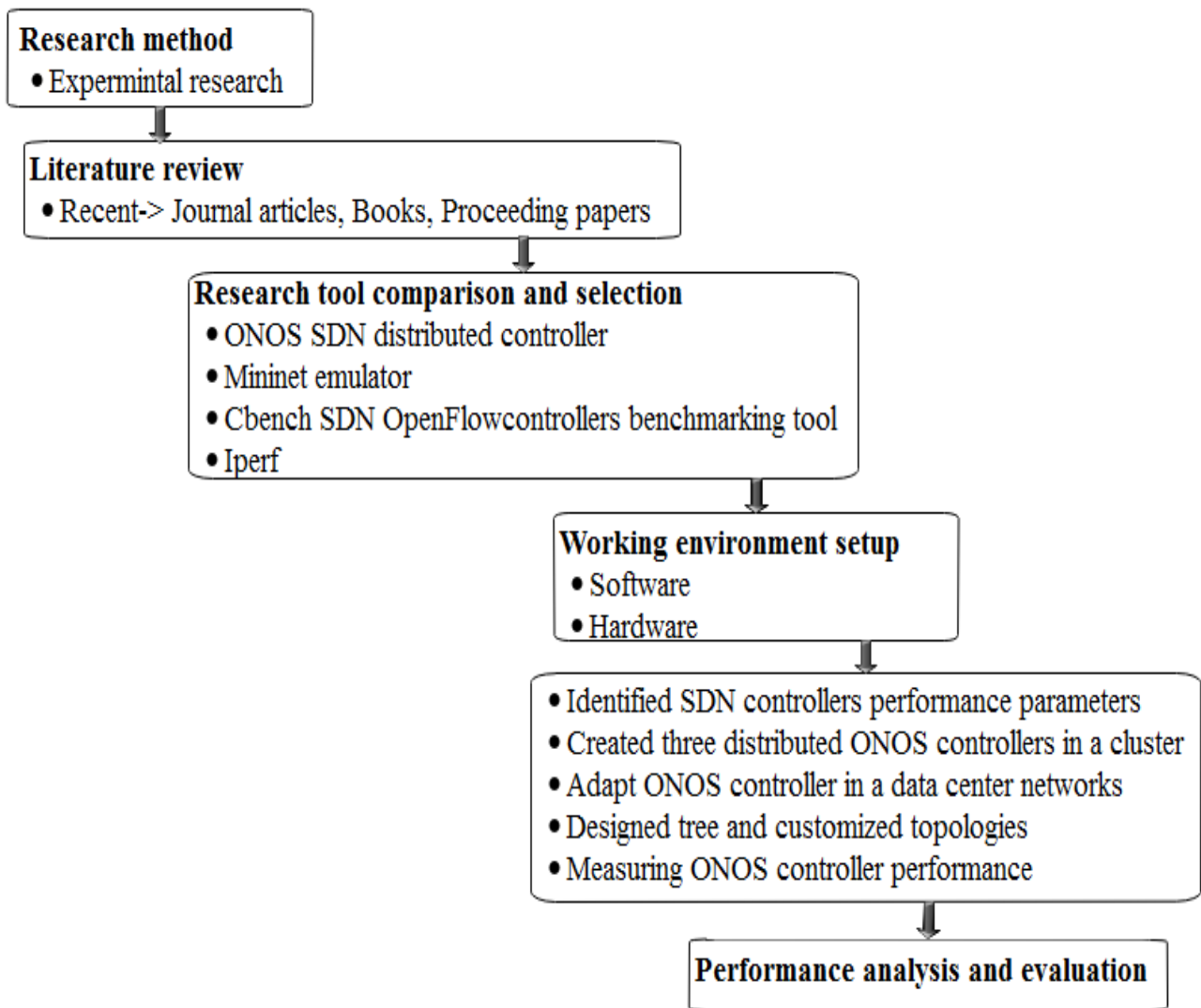


Figure 1-3: Research procedures.

1.8. Thesis Organization

The remaining of the document is organized as follows chapter two discusses the literature review, chapter three discusses research tool comparison, selection and research environment setup, chapter four discusses proposed solution of the work, chapter five briefly discusses controller performance analysis and evaluation and finally gives the conclusion of the work and future work.

CHAPTER TWO

2. Literature Review

In this chapter, we present the literature review on the performance analysis of distributed software-defined network controller in data center network. In order to address some of the most relevant techniques, algorithms and methods. In addition, it is very important to review several related kinds of literature that have been conducted in the field of SDN so far to have an understanding of the problem. Therefore, related literature such as proceeding papers, recent books, articles, journals and other sources are retrieved and consulted. This helps to understand the concepts, principles, and methods that are important for designing, implementing and deploying SDN. Finally, the novelty of the research work proposed in this thesis is presented.

2.1. Overview of Traditional Network

The computer network is the basis for today's telecommunications technologies, as networks become part of the critical infrastructure of our businesses, homes, and schools. The first project in computer networking was introduced in 1967 and it was called ARPANET (Advanced Research Projects Agency Network). ARPANET was the foundation of the global communication. The network technology reaches today's high technologies and the network computing technology has become a part of our day to day life due to the massive amount of information and data that are stored in the cloud [23].

The authors in [24] presented, even though the traditional network is widely spread and adopted, it is very complex and hard to manage such as to express the desired high-level network policies, the network operators need to configure each individual network device separately using low level and vendor specific commands. The control plane and the data plane are bundled inside the networking devices. This reduces the flexibility and delays the innovation and evolution of the networking infrastructure. In addition, network environments have to endure the dynamics of faults and adapt to load changes. According to [25], traditional networks are complex constructs built from special purpose devices like switches, routers, firewalls, load balancers, and middleboxes. Such devices often support standard protocols for transporting data but they come with proprietary interfaces for configuration and control.

2.1.1. Traditional Network Architecture

It is necessary to review the traditional network architecture to better understand the new SDN approach. It has transmission equipment, connectivity between components can be wired or wireless, software and communication protocols. To connect hardware and to create a network different

mediums are used and as a result of those various connections, different transmission modes exist. These transmission modes are Ethernet, Wireless or Optical. The core of the network is made of hardware such as switches, routers, and servers that are connected together with one or more transmission methods. When we consider an end to end communication, the information travels through the network equipment hop by hop until it arrives at its final destination. Sometimes, the information can be segmented into multiple pieces and sent through different paths to get to its destination [26].

To decide where the information should go is not straight forward because the devices require having knowledge about who else is on the network. This information is managed by protocols that are installed in network devices. These protocols communicate with their neighbors and advertise information to keep a concise view of the whole network such protocols install the rules on the network devices on TCAM tables. Since they already know where the information goes, the next step is to know how and where to send such information. In each of the network devices, the TCAM table is looked up for the next hop information for that particular instance. Once the next hop port is found, it is just a matter of sending such information on that port for transmission. This is called the forwarding plane because of the physical operation it takes to look for information and move it to the next network device [27, 28].

In the conventional networks, the logic that decides how packets should be forwarded through the network and the mechanism that is responsible for actually forwarding the packets are strongly coupled. These two constructs are typically referred to as the control plane and the data plane. Furthermore, the entire network structure is highly decentralized; the configuration of one device can be entirely independent of the others. In practice, however, these qualities make large networks exceedingly difficult to manage, debug and reconfigure [26].

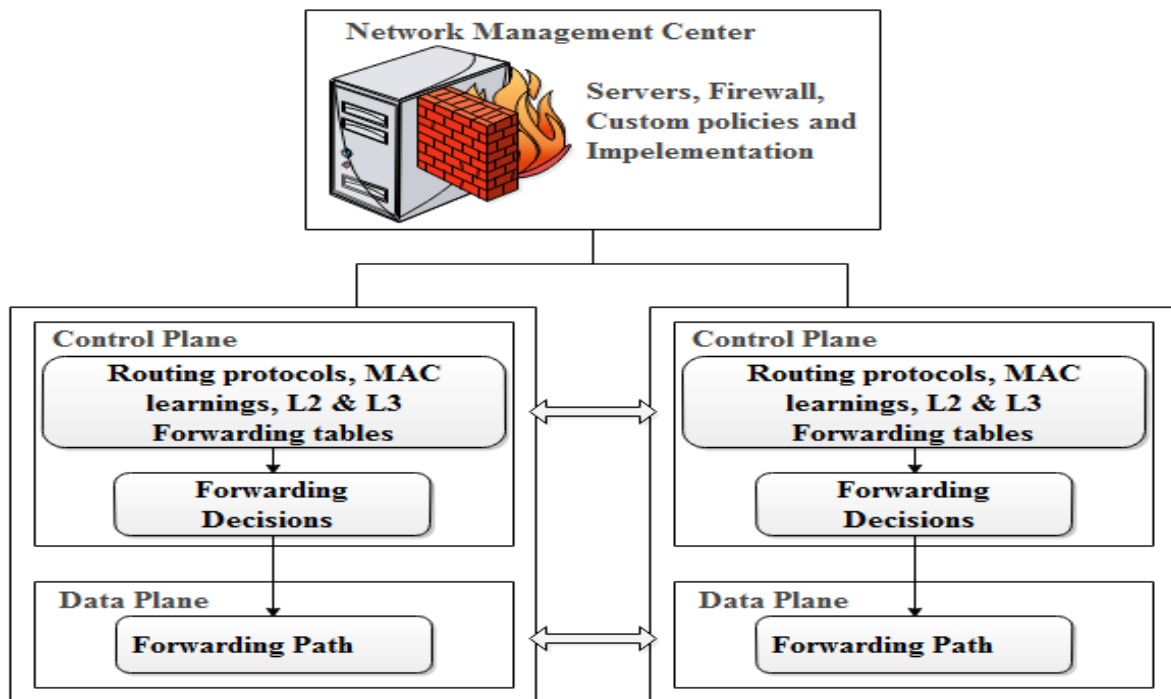


Figure 2-1: The traditional network architecture [26].

Figure 2-1, shows the vertically integrated control plane and data plane of traditional network architecture. The network management center is responsible for managing the network. The network devices such as the routers and switches are made path decision to forward the data to its final destination. In another word each of the network elements such as router and switch perform both the forwarding plane and the control plane operations. This network has several disadvantages some of which are limited innovation, not scalable, prone to error, difficult to manage, expensive equipment, consumes a lot of time to configure each individual device and very complex [26].

2.1.2. Limitation of Traditional Network

The authors in [27] present the evolution of network configuration with a story of two campuses. The researcher presents large campus network has around 1,000 to 1,700 or more network devices and experiences 1,000 to 18,000 changes per month. Regardless of the growth of size and complexity; the method and tools used for configuring the network have not changed much. Network operators still perform complex network tasks by typing low-level commands to configure individual network devices. And they make frequent mistakes when try to change the network configuration.

There are several reasons why network configuration is so hard and error prone. First, network configuration process remains low level and mostly manual. Operators still use CLI to interact and to configure network devices. Second, the configuration files reside in the network devices are distributed all over the network that makes the management so hard. Distributed configuration across routers is the cause of the majority of the configuration faults in access lists. Third, network conditions are dynamic, thus frequent and continuous network configuration updates are inevitable. Traffic

patterns and volume change, host devices arrive and depart new departments appear and move and so forth. Operators generally rely on manual updates to configuration files on each network devices, but also often use ad hoc scripts or tools to automate frequent changes to the configuration. However, such scripts or tools do not prevent misconfiguration, but in fact, can even introduce more misconfiguration [29, 30].

Open networking foundation studies the major drawbacks of the existing traditional network architectures. They found out that the traditional network architecture was not built in a way to meet today's requirements of end users, service provider, data center and enterprise networks. The following are some of the limitations of the traditional network architecture [31]:

Scalability: Today networks have become extremely complex due to the addition of hundreds or thousands of network devices. Large data centers, like Amazon, eBay, Google and Facebook face more serious difficult scalability problems. Since, it is very difficult to scale, set and configure manually these large networks.

Difficulty in applying Policies: In order to maintain a data center network or other network policies, network administrators would need to configure hundreds of routers, and switches individually and manually. It is difficult for network administrators to maintain a consistent setting for access privileges, quality of services and security.

Management Complexity: In order to meet the organization requirements such as high availability, security, and extended connectivity. When add or remove devices from a network become a problem for network administrators, where several parts of the network have to be reconfigured such as access lists, VLANs, QOS, policies, routing protocols and topologies. Due to this, the traditional network is extremely complex to manage.

Equipment Vendor Dependability: The traditional network devices are vendor locked, this influences the continuous research and innovation to make better products at a lower cost. The absence of open standardized interfaces has restricted the ability of the network operators to customize the network to their required environments this affected the network industry badly.

In general, [32] studies that, in the traditional network, most of the network functionality such as routing, switching or network access decisions are handled by network devices. This does not provide much space for novel research ideas such as new routing algorithms to be tested in wide scale, real networks. Moreover, it also allows vendors to give researchers access to their equipment in a unified way without opening up their products. Therefore researchers are able to conduct experiments with new protocols in a real world network without affecting the production traffic. To solve this problem the concept of SDN is released.

2.2. History of SDN

According to [33] the history of SDN began 20 years ago, just as the Internet was taking off, at a time when the Internet's amazing success altered the challenges of managing and developing the network infrastructure. The focus here is on innovations in the networking community whether by researchers, standard bodies, or companies. SDN makes computer networks more programmable; innovation in network management possible and lowers the barrier to deploying new services. In [33] the authors report the history of SDN by dividing it into three main stages according to the time period:

Active Networks: Starts from the mid-1990s to the early 2000s, which introduced programmable functions in the network that lead to greater innovation. The idea of a programmable network initially took shape as active networking, which takes many of the same visions as SDN but lacked both a clear use case and an incremental deployment path.

Control Plane and Data Plane Separation: (From around 2001 to 2007), after the era of active networking research, separating the data and control planes was started to make the network easier to manage. This work focused primarily on better ways to route network traffic a much narrower vision than previous work on active networking.

The OpenFlow and Network Operating Systems: (From 2007 to around 2010), which represents the first widespread adoption of an open interface and developed ways to make control plane and data plane separation scalable and practical. In [28] explain that OpenFlow provides an open protocol to program the flow table in different switches and routers. A network administrator can partition traffic into production and research flows.

The Relationship between Network Virtualization and SDN: Network virtualization is an abstraction of the physical network in terms of a logical network and to virtualize the network it does not require SDN. Similarly, SDN is the separation of a logically centralized control plane from the underlying data plane and it does not imply network virtualization. SDN and network virtualization relate in three main ways: SDN as an enabling technology for network virtualization, network virtualization for evaluating and testing SDNs and Virtualizing SDN. In general, researchers describe the history of SDN and each technology developed over the past 20 years is shown graphically below.

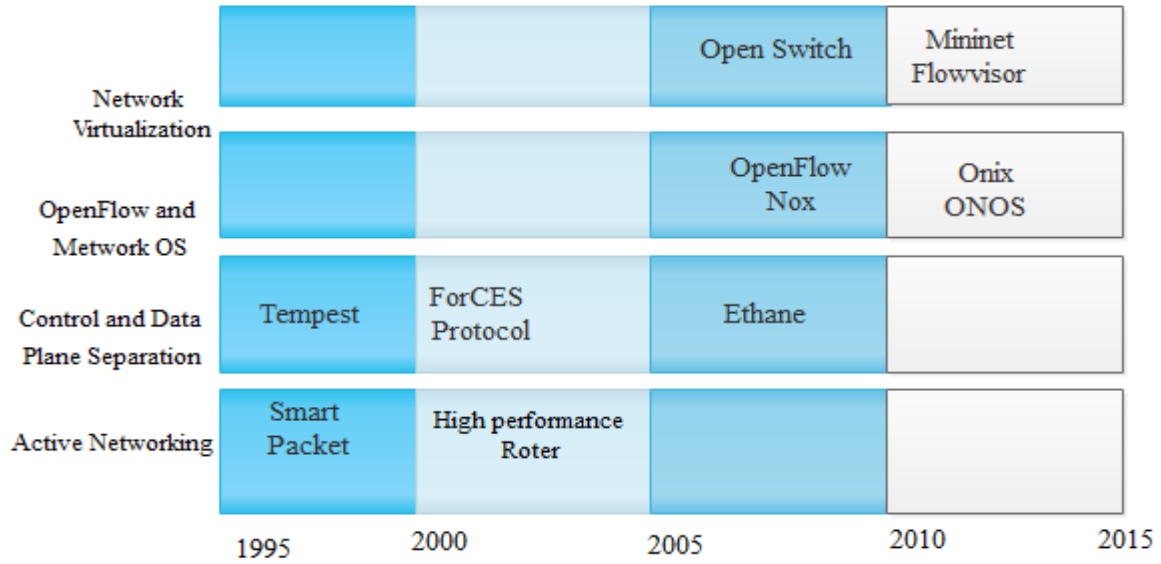


Figure 2-2: History of Programmable Networking over the Past 20 Years [33].

2.2.1. Definitions of SDN

The authors in [34] defined SDN as a new network concept that separates the control plane from data plane to enables the centralization of a programmable control plane. SDN is simply a process that separates the control plane from the forwarding hardware. The Open Networking Foundation (ONF) also defines the process as a product unique to the SDN architecture, which enables the control and data planes to be decoupled with the network intelligence and state are logically centralized and the fundamental network infrastructure is abstracted from the application [35]. ONF also define SDN in terms of its architecture, SDN is a new architecture that is dynamic, manageable, cost-effective, adaptable, making high-bandwidth, and dynamic nature of today's applications [35].

2.3. SDN Architecture

According to [24] SDN seeks to resolve these challenges by completely rethinking network management. SDN has three main characteristics such as decoupling of data planes and control planes, centralized control of the entire network and programmability. By exposing centralized view and control over an entire network infrastructure, new functionality can be added as needed. SDN discovers and updates a global abstract network view to facilitating the central control logic. SDN terminology can be described from up to bottom as management plane, northbound interface, control plane, southbound interface, data plane and forwarding devices. According to ONF [29] the SDN architecture has advantages over the traditional network are listed in Table 2-1.

Figure 2-3: The advantage of SDN network over the traditional network [35].

<i>Advantages of SDN network over traditional network</i>	
Open standards	When implemented through open standards, SDN simplifies network design and operation.
Vendor independent	Instructions are provided by SDN controllers instead of multiple, vendor specific devices and protocols.
Configured Programmatically	SDN lets network managers configure, manage, secure and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves.
Centrally managed	Network intelligence are logically centralized in software based SDN controllers, which maintain a global view of the network, appears to applications and policy engines as a single logical switch.
Programmable	Network control is directly programmable due to this the network manager can update, control and modify the controller program easily.
Agile	Abstracting control plane from forwarding plane lets administrators dynamically adjust network wide traffic flow to meet changing needs.

The Figure 2-4 shows the three main layers of SDN architecture. An SDN architecture has an application layer that runs on a controller, the control layer that contains a controller to control the device and to make a decision and the infrastructure layer that contains a lot of OpenFlow devices. Also, SDN architecture has three application interfaces such as northbound, southbound and east/westbound application interfaces

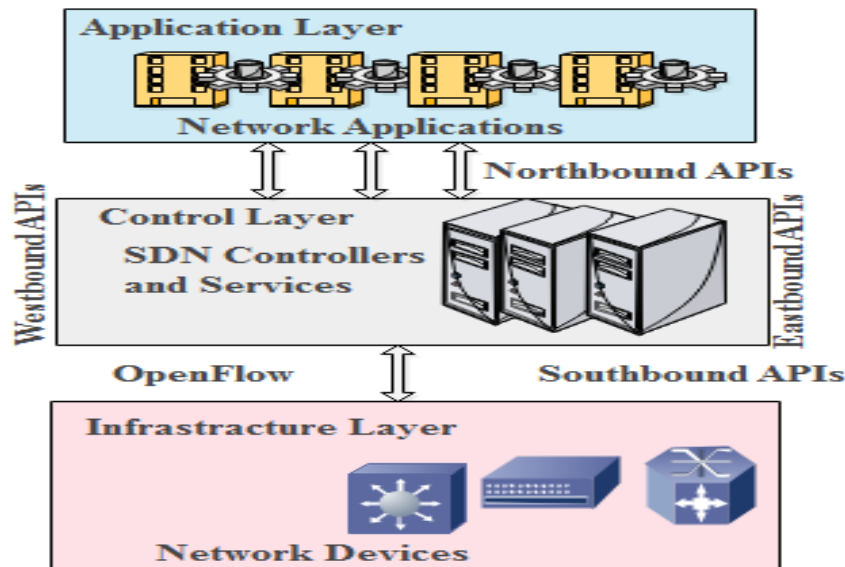


Figure 2-4: SDN architecture [4, 26].

2.3.1. Application Layer

The application layer contains network applications that can introduce new network features, such as security, manageability, forwarding schemes to assist the control layer in the network configuration.

It can receive an abstracted and global view of the network from the controllers and use that information to provide appropriate guidance to the control layer. The interface between the application layer and the control layer is called the northbound application interface [34].

As illustrated in Figure 2-3 **Northbound APIs**, offers the software interfaces between the application layer and the control layer, through providing universal data and functionality for network applications. Current controllers offer a quite broad variety of northbound APIs, such as ad-hoc APIs, RESTful APIs multilevel programming interfaces, file systems, between other specialized APIs such as NVP NBAPI and SDMN API [4].

2.3.2. Control Layer

Control plane or layer is the interface defined between an SDN controller and an SDN data plane. It provides programmatic control of all forwarding operations, capabilities advertisement, statistics reporting and event notification. One value of SDN is that the control plane is implemented in an open, vendor-neutral and interoperable way. Between the control layer and infrastructure layer contains southbound APIs, east/westbound APIs, and verity of protocols.

As shown in Figure 2-3, **East/Westbound APIs** are a special case interfaces required by distributed controllers. The functions of these interfaces include import or export data between controllers, algorithms for data consistency models and monitoring notification capabilities i.e. check if a controller is up or notify a takeover on a set of forwarding devices. Eastbound APIs are able to interconnect traditional IP networks with SDN networks, which often demand a translation module in between. Westbound APIs are able to deliver information between multiple SDN controllers in diverse areas and enable management of distributed SDN architecture [4].

Southbound APIs offer the software interfaces between the control layer and the infrastructure layer. OpenFlow is the most popular southbound interface to separate control and forwarding plane. On the lower level of the control plane, the southbound APIs can be seen as a layer of forwarding device drivers. It provides a common interface for the upper layers while allowing a control platform to use different southbound APIs namely OpenFlow, OVSDb, ForCES etc. Also protocol plugins to manage existing network devices such as SNMP, BGP NetConf etc. However, most controllers support only OpenFlow as a southbound API [4, 5].

2.3.2.1. OpenFlow Protocol

OpenFlow was first proposed by McKeown to enable easy network experiments on a campus network and is currently used in most SDN practices. It is the first protocol that popularizes the concept of SDN. OpenFlow takes advantages of most modern Ethernet switches and routers contain flow tables for essential networking functions such as routing, subnetting, firewall protection and statistical analysis of data streams [28, 36]. OpenFlow protocol offers suitable flow table operation services for

a controller to insert, delete, modify, and look up the flow table entries through a secure TCP channel remotely. The OpenFlow network architecture has three basic concepts such as OpenFlow switches in the data plane, control plane consists of one or more OpenFlow controllers in the control plane and secure control channel connects the switches with the control plane.

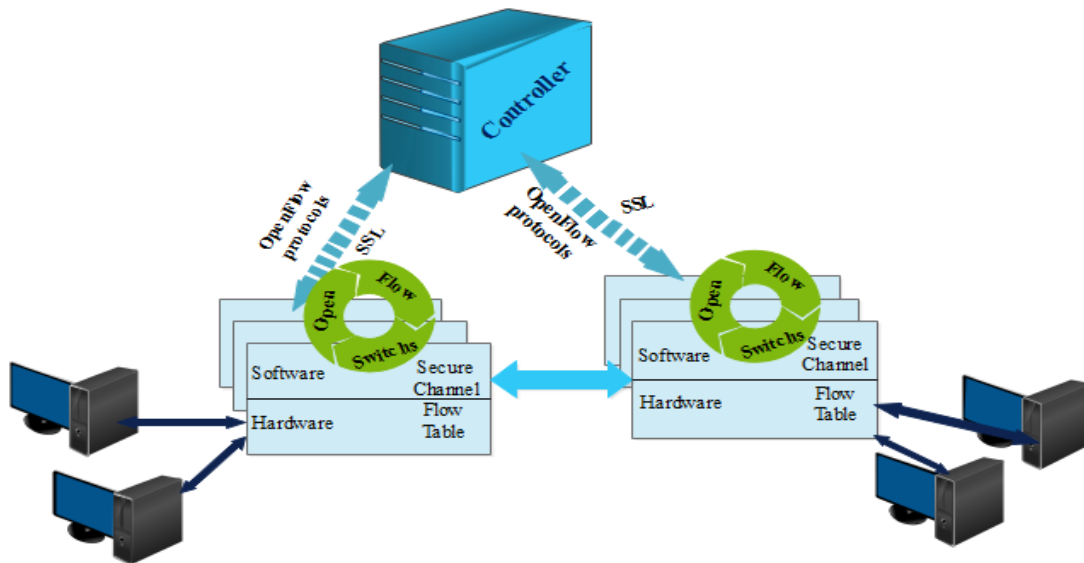


Figure 2-5: OpenFlow Switch architecture [28].

OpenFlow switch is a software program or hardware device that forwards packets in an SDN environment based on the OpenFlow protocol. It has three main parts such as flow table is used to lookup packet and also do forwarding, a secure channel that connect the control and data plane and OpenFlow protocol that carries the message between SDN controllers and the underlying network infrastructure. The connection between the OpenFlow controller and the switch is secured using SSL or TLS cryptographic protocols, where the switch and the controller are mutually authenticated by exchanging certificates signed by both sides' private key [36, 37].

OpenFlow is a new standard open source protocol and it is standardized by the ONF with the aim to provide programmable networks. OpenFlow secure channel acts an interface to connect switches and controllers. Three main types of messages can be sent through the secure SSL or TLS channel such as a controller to switch, asynchronous and symmetric messages. Controller to switch messages are sent by the controllers to manage and check the state of the switch. Asynchronous messages are sent by the switch to update the controller about network events and changes to the switch state. Symmetric messages can be initiated by either the switch or the controller and sent without solicitation [4, 31, 35].

2.3.3. Infrastructure Layer

The infrastructure layer or data plane contains forwarding devices such as switches, links, and hosts. In the data plane, the switching device performs packet forwarding, based on the forwarding rules

imposed by the controller. The main function of an SDN switching device's data plane is packet forwarding. Specifically, upon receiving a packet, the switching device first identifies the forwarding rule that matches with the packet and then forwards the packet to the next hop. Compared to packet forwarding in legacy networks based on IP or MAC addresses, SDN packet forwarding can also be based on other parameters like OpenFlow, TCP or UDP port, VLAN tag and ingress switch port [36].

2.4. Why SDN Network Needed?

Since SDN provides easier and more flexible system for network management. Any change in the network configuration such as adding or removing of devices can be very easily handled in SDN. The network administrator does not need to go to each individual device in the network to modify the configuration. Instead, configuring the changes in the controller would deploy the modifications on the entire network. Because SDN controller has a centralized view of the overall network devices. SDN facilitates communication between the applications and the network. This results in a dynamic network for a dynamic application. SDN provides various features than the vertically integrated traditional network. In general, SDN network is scalable, flexible, reliable, secure, programmable, and has high performance and availability than the existing traditional network [31, 38, 39, 40].

2.4.1. Open Networking Foundation

The ONF [8] is a user driven nonprofit organization focused on promoting, standardizing, specifying and adopting of SDN network through open standard development. It was founded by Deutsche Telekom, Facebook, Google, Microsoft, Verizon and Yahoo in 2011. As [15] ONF brings together about 97 companies and is dedicated to releasing, promoting and adopting of OpenFlow specification. In [35] now a day ONF has more than 501 members to standardize, promote, implement and adopt SDN network. This enables to make SDN network the future best network throughout the world.

Table 2-1: Members of Open Networking Foundation [7].

<i>Open Networking Foundation Members</i>			
Nokia	Ericsson	Globe Telecom	Corsa Technology
Ciena	Facebook	SDN Essentials	Criterion Networks
AT&T	Google	SK Telecom	Daho Cloud Networks
ZTE	Huawei	Telecom Italia	ATTO Research
Oracle	Intel	Vodafone	Juniper Networks
Samsung	Lenovo	Ecode Networks	Extreme Networks
SDNLAB	Yahoo	Cisco Systems	Microsoft Corporation

2.5. SDN OpenFlow Controllers

SDN OpenFlow controllers are the core of SDN network. The concept of SDN controller was born along with the introduction of SDN. The controller is like a human brain; human brain control and order all organs of the human similarly control plane control and manage all of the forwarding devices

in the network. SDN OpenFlow controller is an application that manages flow controls to enable intelligent networking. Any communications between applications and devices have to go through the controller. The controller uses OpenFlow protocols to configure network devices and choose the optimal network path for application traffic and to tell switches where to send packets. The controller APIs enables the controller to communicate to the switches below southbound APIs and applications above northbound APIs [41, 42].

Nowadays there are more than 30 different OpenFlow controllers designed by different groups, written in different programming languages, having different architecture and performances. Some of OpenFlow controllers are: NOX [43], POX [44], Terma [41], Ryu [45], Floodlight [46], OpenDaylight [5], ONOS [47] etc. These SDN controllers need a research on capability, availability, scalability, performance and communication between distributed controllers. The basic requirements for building SDN controller provided by the SDN controller community are [24, 28, 43, 48]:

Table 2-2: The basic requirements to build SDN controllers.

<i>Basic requirements for building an SDN controller</i>	
Performance	High performance controller for various network loads and applications is a key issue.
Scalability	The controller should be scalable without losing the modularity in design. This avoid single point of failure and increase the network availability.
Flexibility	Various applications should be able to run on the controller and use the common functionality that the controller has provided.
Scale the Development Process	Controller applications and modules can be dynamically plugged into the controller; hence the architecture should allow them to be developed independently.
Availability	The percentage of time, in a specific time interval, during which a controller, or other machine can be used for the purpose that it was originally designed and built for.

The Table 2-3 below shows the comparison between six centralized SDN OpenFlow-based controllers using two criteria's with the short description of the controller. Each controller has different criteria and different researchers do their research on the controller performance.

Table 2-3: List of centralized OpenFlow SDN controllers [30, 37].

Controllers	Language	Developer	Description
NOX [43]	C++/Python	Nicira	It is the first centralized OpenFlow SDN controller.
POX [44]	Python	Nicira	It is a general, open source SDN controller that supports OpenFlow protocol.
Maestro [49]	Java	Rice University	It tries to improve the system throughput by exploiting multicore processors and parallelism. It provides interfaces
Ryu [45]	Python	NTT	It is SDN operating system to provide logically centralized control and APIs. It supports OpenFlow.
Beacon [50]	Java	Stanford University	It is a cross platform and modular OpenFlow controller. It supports event based and threaded operation.
Floodlight [46]	Java	Big Switch	Floodlight supports mixed OpenFlow and non OpenFlow networks. It based on the Beacon implementation.

2.5.1. SDN Distributed OpenFlow Controllers

SDN distributed controller is fault tolerance, avoid a single point of failure, improve performance, resilience, scalability, availability, avoid a single point of failure and reduce the impact of problems caused by a network partition. Distributed controller handle complex tasks such as quality of service, managing virtual private networks in large networks and traffic engineering [5] [49].

In order to solve the centralized SDN controller problem various distributed controllers are designed and implemented. This [50] design a scalable control mechanism to solve switch migration problem (SMP). So that, the authors proposed DHA to solve this problem. They indicate two distributed controller architectures such as hierarchical architecture is vertically layered from root controller to leaf and flat architecture is horizontally equalize all the controllers. This [10] reports, a single controller is insufficient to control the entire network; so distributed controller is needed. A distributed controller architecture helps to implement well-known distributed routing protocols, like BGP, OSPF, and ISIS. They also determined how many controllers to use and where to place them.

In [24] multi-controller architecture is a set of controllers working together to achieve some level of performance and scalability. Since multiple controllers are used to cooperatively manage the entire network devices. In an SDN, there are three types of architectures, centralized, physically distributed but logically centralized and completely distributed. Logically centralized but physically distributed controller is one of the implementation methods of distributed control plane.

The Logically Centralized but Physically Distributed Control Planes [24]: In a logically centralized controller, all the controllers have the same responsibilities and split the charge equally, but one is a master. They always aware every change in the network and share the same information instantly to the network synchronization. It allows each controller in a cluster to exchange information each other using east/west APIs and protocols.

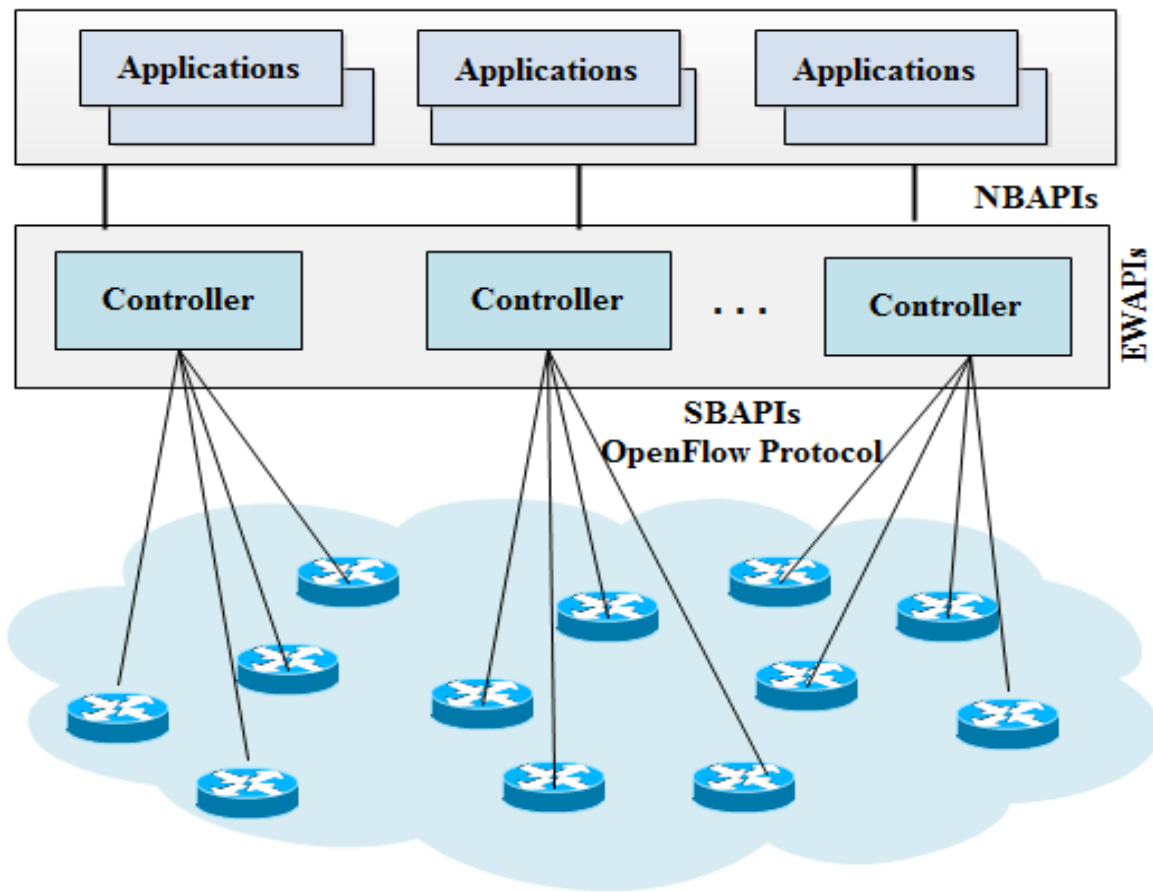


Figure 2-6: Logically centralized but physically distributed OpenFlow controller architecture [24].

2.5.1.1. Distributed SDN Controllers in a Data center Network

Data center network (DCN) is a network that contains telecommunications, switches, routers, links, hosts huge servers, power supplies, environmental controls and security devices [70]. DCN needs, high availability, high bandwidth, and low latency than enterprise and internet service provider networks. Because, today data center networks are needs scalable, flexible, available, secure, fast and optimal devices. Companies try to find solutions that allow them to expand their network easily and cheaply while maintaining the option to choose their equipment without linked to their previous vendors [51].

To address these problems the researchers, network companies, data centers, ISP and enterprise networks are turning to SDN network. SDN distributed controllers played a key role in the evolution of the modern data center networks. SDN distributed controllers have emerged as a viable solution to manage from small to large data center networks [52]. So, to make DCN scalable, high performance, available, and flexible distributed controller is the first option [12].

After this to achieve the aim of this thesis, we have reviewed the journal articles, and articles that are more related to our objective.

2.6. Related Works

In [53] the authors evaluate the performance of the floodlight controller on a single server and shown that it can handle over 1.6 million requests per second with an average response time of 2ms. Also, the authors in [54] present the evaluation result of SDN performance, where the authors run a single controller and determine that a single controller can control up to 5000 switches with 46 cores. They were able to achieve a throughput of 14 million flows per second with a latency of 200 μ s. These authors in their future work indicates, one controller can't well in many scenarios, so large networks need more than one controller and performance evaluation

In [18] compare four OpenFlow controllers such as NOX, Beacon, Maestro, and Floodlight. They considered four key performance bottlenecks including multi-core support, switch partitioning, packet and task batching. The performance evaluation results show that static switch partitioning and packet batching designs are best for controllers in high throughput networks. Packet and task batching designs are good for controllers with delay sensitive control plane applications. The results also show that sending control messages out individually can improve the latency performance. Based on these results, a Floodlight controller is the best controller, whose performance was better than the other three controllers.

In [55] also compare five controllers each other such as POX, Ryu, Terma, Floodlight, and OpenDaylight. The authors used multi-criteria decision making (MCDM) method and priority vector parameter to select the best controller. They compare these five controllers based on available interfaces, supporting of virtual switching, GUI, supporting of REST API, productivity, being open source, having documentation, age, modularity, supporting language, platform, TLS, OpenFlow and OpenStack networking. The results show that Ryu has the best priority vector value (0.287). Floodlight, OpenDaylight, Terma and Pox controller has the next best results respectively. The priority vector values of Floodlight and OpenDaylight controllers (0.275, 0.265) were very close to the Ryu controller.

The authors in [52] perform performance evaluation on centralized OpenDaylight and Floodlight SDN controllers. The authors used cbench tool to measure the throughput and the latency this controller in a data center network. And they used Mininet emulator to emulate the topology that contains 8, 16 and 32 switches. The latency test was run for a minimum of 10 minutes and repeated 10 times with 8 switches, Floodlight produces 1214 responses/sec. When the number of switches is increased slight increase in the number of responses, from 1214 with 8 switches to 1239 with 16 switches and to 1335 with 32 switches. In all evaluation, the minimum number of responses per second per switch is recorded to be 1208 whereas the maximum is recorded to be 1404. The latency result was 55/sec, 27/sec, and 15/sec on average with 8, 16 and 32 switches respectively.

The latency of Floodlight increases when the number of switches is increased but ODL shows

unexpected behavior as the numbers of responses decrease when the number of switches is increased. The final result is OpenDaylight SDN controller has low average responses and high latency as compared to Floodlight.

In throughput mode, the controllers are evaluated for how many packets it can process in a second. Testing scenarios were same as latency tests except tests were run for a minimum of 60 seconds and each test was repeated 6 times. As increased the number of switches to 16 and 32, the number of responses per second decreased to 9408 and 4536 respectively. The OpenDaylight had very low responses as compared to Floodlight. Average per switch responses recorded on the OpenDaylight controller were 270, 33 and 34 for 8, 16 and 32 switches respectively.

This article [56] introduces when the number of connected devices increase requires a novel solution to control the incoming SDN network resources. In order to evaluate the benefits of having multiple SDN controllers, the researchers used OpenDaylight controller, Mininet emulator, and cbench benchmarking tools. The testing environment contains two SDN controllers and 26 switches created using OpenvSwitch. They used three experimental scenarios such as single connection, all connection, and selective connections.

The result is when the number of packet-in is greater than 40,000 the all and the selective connection are becomes relevant. At 50,000 packet-in this difference becomes more than 8 seconds. Additionally, when the network load is high the throughput of the selective test scenario is 66% greater than the all connection test scenario and 100% greater than the single connection test scenario. Finally, the authors conclude that the selective connection better than any other configuration. In general, the authors doesn't evaluate the performance two OpenDaylight controller and its key parameters properly. Due to this, the test results influence the control plane design of three projects. Furthermore, they are not investigating different topologies and tests to identify the advantage and drawbacks of a distributed control plane [56].

The aim of this paper [57] is to understand the performance and limitation of SDN OpenFlow controllers. The authors conduct numerical analysis to evaluate the proposed model performance with different parameter values. Their experimental results indicate that the controller model provides a more accurate approximation of SDN controller performance. Finally, they indicate to apply the queuing theory to build analytical performance models for other widespread SDN controllers of data center networks as future work. Furthermore, extend the analytical performance models from single SDN controller to clusters of SDN controller still requires a research.

Authors in [50] used Chinanet contains 8 switches and 59 links and Cernet contains 36 switches and 53 links real network topologies. They used Beacon controller in an individual machine to simulate single centralized controller. Then compared DHA-CON contains 4 controllers with the scenarios of the single controller (S-CNTL) contains a single controller, static distributed controllers (D-CNTL) contains 4

controllers and Dynamic Controller Provisioning Problem with Greedy Knapsack and Simulated Annealing (DCP-GK and DCP-SA).

These authors obtained the following results single controller has the lowest scalability due to its resource-limited architecture. The average flow setup time for S-CNTL and D-CNTL are more than 0.65s and 0.35s. Average utilization ratio S-CNTL has 100% utilization due to its limited resource. Each controller in DHA-CON has more than 90% utilization. The controller in D-CNTL has less than 80% utilization in average because of the load imbalance among different controllers. When $\beta=50$ the average actual utility loss is 0.23 and 0.22 for Chinanet and Cernet. Finally, they indicate resilience and performance evaluation requires study and experiment.

This survey [58] present many essential research issues about SDN controllers; especially focus on the controller architecture, performance, scalability, placement, interface and security problems. Because its aim is to provide an up to date view to the SDN controller. The network uses a single controller, which leads to performance, single point of failure, availability and scalability problems. If the controller fails or becomes the performance bottleneck, the network will lose the advantages of SDN. Much different architecture has been proposed recently, such as the multi-core controller, the logically centralized controller, and the completely distributed controller. However, these designed distributed control plane methods and architectures need performance analysis and still lack in-depth study.

In this literature review section, we have compared traditional and SDN network architecture. Also, we have discussed traditional network limitation, the evolution of SDN, SDN layers, use of SDN, OpenFlow protocols, SDN OpenFlow controllers, distributed controllers, the application area of distributed controllers and related papers to this thesis. In addition, we have described the organization, which standardize and promote SDN network. This helps us to understand SDN concept and to find a gap that is fulfilled by this thesis.

In order to summarize the related work, we have used [18, 50, 52, 53, 55, 56, 58] journal articles as a base paper from the related work. Because, these articles are related to our work, this helps us to find the gaps. The Table 2-4 stated that the gaps that were identified in these base papers.

Table 2-4: Identified gaps in the literature review.

<i>Journal Articles</i>	<i>Identified Gaps</i>
[18]	S.Rowshanrad et al, were only focused on performance evaluation and comparison of a centralized floodlight and OpenDaylight controllers. Also they only consider latency performance metrics with low load network, mid load network and with high load network by using tree, linear and single topology. In addition, they are not consider performance evaluation on a distributed SDN controllers in a cluster under complicated topology.
[56]	G.Cheng et al, were performed performance evaluation on two distributed OpenDaylight controllers by using only 26 switches. This article does not consider any network topology and not used different scenarios. Also these authors were not consider comparison of single and two OpenDaylight controllers by using OpenFlow controllers key performance metrics.
[52]	Z.Khan Khattak et al, were only focused on performance evaluation and comparison of a centralized floodlight and OpenDaylight controllers with the same title with [18]. These authors also not consider different topologies and distributed SDN controllers.
[53]	A. Tootoonchian et al, were evaluate single floodlight controller performance on a single server and not consider different topologies and different scenarios.
[55]	R.Khondoker et al, were compared five centralized SDN OpenFlow controllers and select the best controller by using fifteen criteria. These authors does not consider distributed controllers comparison and performance evaluation.
[50]	L. Zuccaro et al, in there research perform evaluation between single, four static and four dynamic Beacon SDN controller on a two real networks. However, this paper perform evaluation only in average flow setup time and average resource utilization ratio. In addition, this paper does not, which controller and topology is suitable or not for datacenter network. Also, the paper not measured the throughput and latency of Beacon controller.
[58]	We used this survey as a base paper. Since, it recommends performance analysis of SDN distributed controller as future work.

Finally, all previous studies haven't considered SDN distributed controller performance analysis by running different number of controllers with different topologies. Although, previous works haven't considered ONOS distributed controller performance analysis and using in a data center networks. In addition, previous works haven't considered different scenarios and comparison of the controller by running a different number of controllers in a cluster. In general, the work presented in this thesis is different from all previous works. Therefore, we have initiated to do performance analysis on distributed ONOS controllers in a data center networks.

In this literature review, we have understood the concept of SDN networks, SDN controllers, state of the art of SDN networks and the gaps. Also, we have learned and understood what journals say about distributed controllers, research tools, research environmental setups, research methods, SDN controller performance parameters, standards and the ways to analysis evaluate the research results. According to this, we have been compared and selected research tools and setup the research environment in the next chapter, which is chapter three.

CHAPTER THREE

3. Research Tools and Environment Setup

This section focused on the comparison and description of research tools and working environment throughout in this thesis progress. We have both the software and hardware environmental setup. In the hardware setup, we have used OptiPlex 7020 model desktop with an Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz x 4 processor, 4000 MB of RAM and 1 TB of hard disk. In the software environment setup, the host machine runs windows 7 ultimate 64 bits operating the system with model OptiPlex 7020 and the rating 5.1. To create the guest machine install VirtualBox 5.1.2 in VM install Ubuntu 64-bit operating system. Inside the guest machine, we have installed and used ONOS 1.2.1, Mininet 2.2.1, OpenvSwitch 2.3.90, cbench 1.8, oracle java 8 and OpenFlow version 1.0 and 1.3.

3.6. Comparison of SDN Distributed Controllers

Today there exists many SDN OpenFlow controllers including Ryu [45], POX [44], NOX [43], Floodlight [46], ONOS [41], Onix [59], HyperFlow [60], OpenDaylight [61] and DISCO [62]. The question is, which of the controllers is to be selected and used? To solve this problem, we studied literature, websites, blogs and any available resource providing information about the existing SDN distributed controllers. The method is analyzed several existing open source distributed controllers to collect their properties and parameters. We have ranked the above-listed controllers based on their distributed architecture, released year, scalability consistency, application area, and need performance study and supporting of different applications and tools. Among them, the top four controllers have been selected for the study to gather their properties. The selected controllers are ONOS [16], OpenDaylight [61], Onix [59] and HyperFlow [60]. The table below shows the comparison of SDN distributed controllers.

Table 3-1: Comparison of distributed OpenFlow controllers [5, 16, 36, 55, 58, 59, 60, 63].

<i>Criteria</i>	<i>ONOS</i> [65]	<i>OpenDaylight</i> [63]	<i>Onix</i> [61]	<i>HyperFlow</i> [62]
Architecture	Distributed	Distributed	Distributed	Distributed
Platform Support	Linux, Mac	Linux, Wins,	Linux	Linux
OpenFlow version	1.0-1.5	1.2.0, 1.3, 1.4	1.0.0, 1.3	1.0.0/1.1.0
Language	Java	Java	Python, C++,C	C, C++
Physically Distributed	Yes	Yes	No	Yes
Logically Centralized	Yes	Yes	Yes	Yes
Logically Distributed	No	No	No	No
Northbound APIs	RESTful API	REST, Java APIs RESTCONF	NVP, NBAPI	NO
Southbound APIs	OpenFlow, OVSDDB, NetConf	OpenFlow, OVSDDB, NetConf	OpenFlow, OVSDDB	OpenFlow, OVSDDB
East/Westbound APIs	Yes	No	Yes	Yes
Availability	High	High	Medium	Yes
Scalability	High	Medium	High	High
Performance	High	Medium	Medium	High
Security	High	High	Medium	Medium
Reliability	High	Medium	Medium	Medium
GUI support	Yes	Yes	No	Yes
Management Interfaces	GUI/CLI, REST API	GUI/CLI, REST API	No	No
Open Source	Yes	Yes	No	Yes
State consistency	Strong	Weak	Strong	Weak
Modularity	High	High	Medium	Medium
Flexibility	Yes	Yes	No	Yes
Developer Team	ON.LAB	Linux Foundation	Nicira	Amin Tootoonchian
Faults	No	No	Yes	Yes
Documentation	Weak	Medium	Weak	Weak
1 st Released Year	12/5/2014	2/8/2013	8/14/2010	6/20/2010

In Table 3-1 based on 26 criteria, four distributed controllers have been compared. Our mandatory requirements are controller must have distributed architecture, high performance, high scalability, high availability, new released, GUI, northbound and southbound APIs, Linux, application area, consistency and open source. In addition, we have considered which controller needs study and which are not. All of the controllers in Table 4-1 above almost support our mandatory requirements and they are suitable to use. Even though, depend on these mandatory criteria we have selected the most recent ONOS controller.

3.6.1. OpenDaylight

OpenDaylight was designed in 2013, the OpenDaylight Project was made as a Linux Foundation project. It can be deployed on any hardware that supports Java. Some of the features of this controller are Java based (OSGI), modular, and supporting of multiple southbound protocols. OpenDaylight supports the program bidirectional REST that supports applications running in the same address space [18]. Internal and external requests for services will be planned by a service abstraction layer to the suitable southbound plugin and a basic service abstraction that higher-level services are built. Topology abstraction and discovery, PCE-P and CSPF, OpenFlow, I2RS, and NETCONF are other built-in services within in OpenDaylight [18].

There are three major releases of the OpenDaylight controller such as Hydrogen, Helium, and Lithium. Linux foundation team released new version OpenDaylight controller two times a year [61]. We have not selected this controller since, it is designed for data center networks, not support east/westbound interface, not consistence and has medium performance.

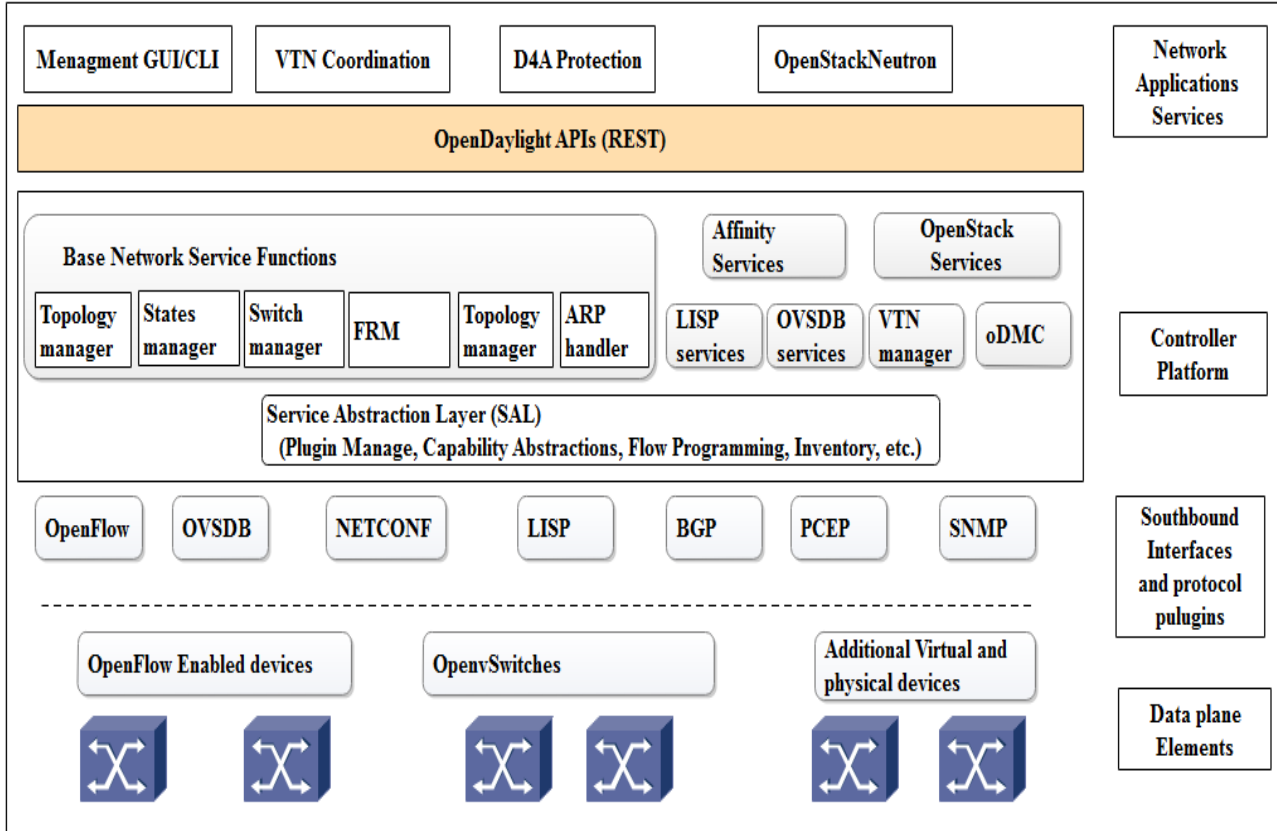


Figure 3-1: OpenDaylight Architecture [61].

3.6.2. Onix

Onix [59] is distributed controller, in a cluster. It uses one or more physical servers, each of which may run multiple Onix controllers. Onix is responsible for giving the control logic programmatic access to the network both reading and writing network state. It reduces workload by adding controllers without merely replicating it. It allows aggregation in which the network managed by a cluster of Onix controllers appears as a single controller in a separate cluster's network information base. It provides applications with control over the consistency and durability of the network state. It can handle four kinds of network failures such as forwarding element failures, link failures, Onix controller failures and failures in connectivity between network elements and Onix controllers.

Onix controller is not supported GUI, not support management interface, not flexible and not support all OpenFlow versions. Due to these limitations, we haven't selected Onix controller to do this research [5, 55].

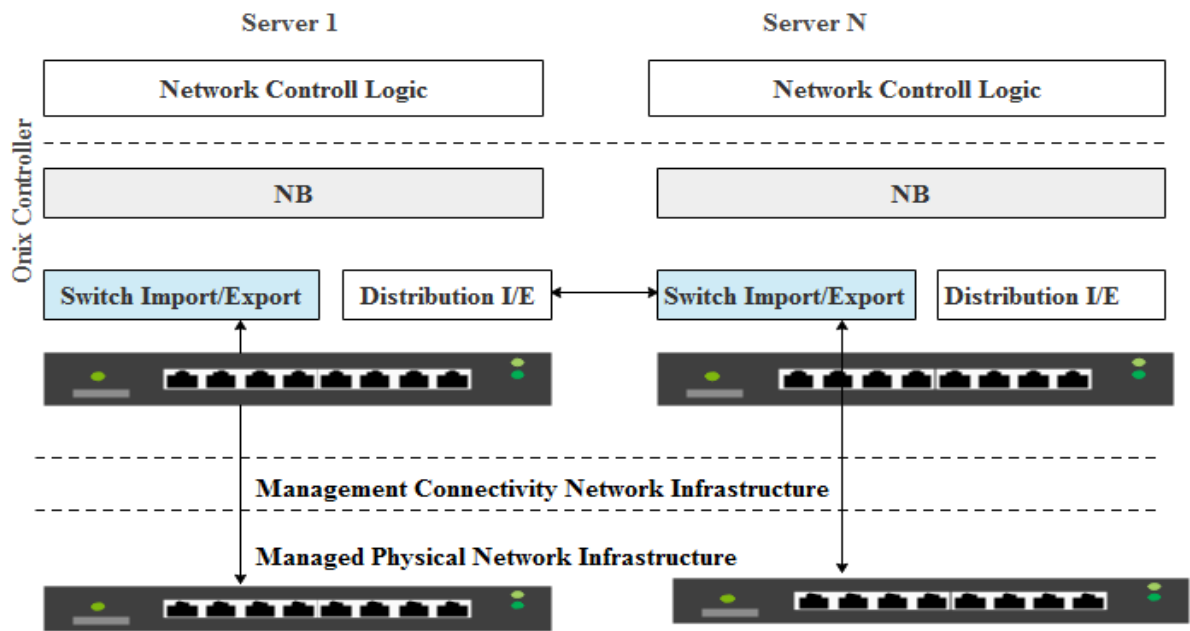


Figure 3-2: Onix architecture [59].

3.6.3. HyperFlow

The aim of HyperFlow designers was to keep the network control authority centralized and provide the distributed controllers with a global view of the network. It uses event propagation system, to keep information consistent among the controllers. Also, it introduces new functionalities in switches to suppress frequent events and to reduce the load on the controllers. Adding new primitives to switches is undesirable because it breaks the general principles of SDNs [55].

The HyperFlow has control, forwarding, and application layers. The control layer contains multiple NOX controllers that are working cooperatively. Forwarding layer, the switches are connected to the nearest controller. When the controller fail a switch can be reassigned to another controller. Application layer which contributes application to the global status of the network. These events are stored in the publish-subscribe system, which is set to synchronize the global view of the controllers. As each controller has a global view of the network and its status, it has capabilities to install flows into all switches. All required flows for a particular data stream are published to the distributed controllers [60].

We haven't selected HyperFlow to do this thesis because it has not management interface, northbound APIs, weak consistence and not support switch migrations.

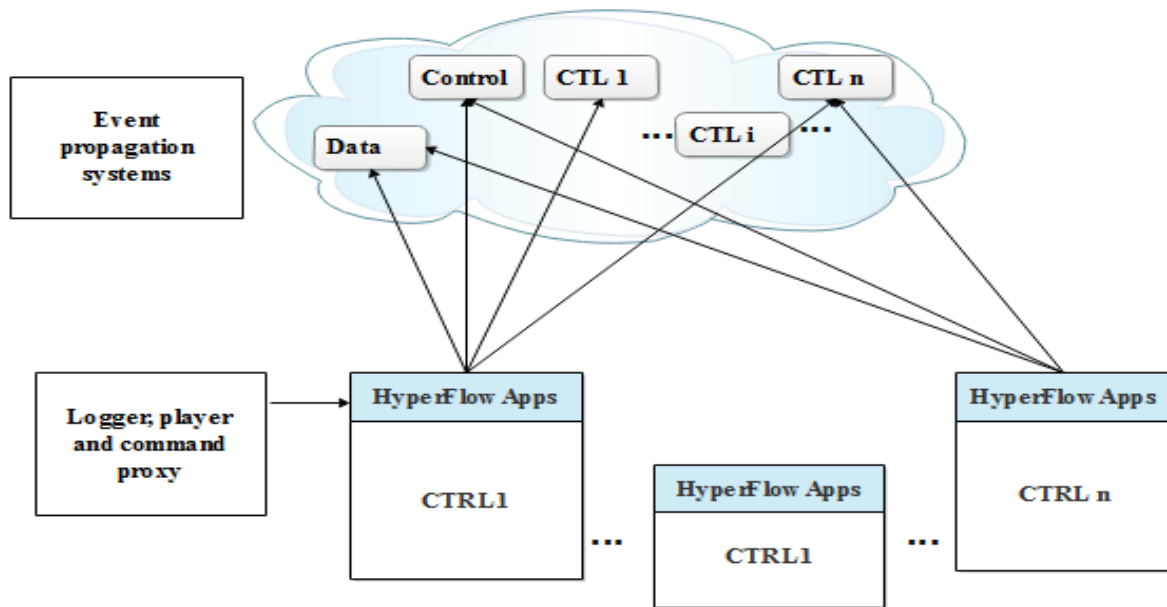


Figure 3-3: HyperFlow Architecture [60].

3.6.4. ONOS

In our case, we have selected SDN ONOS controller because ONOS controller is able to operate and control the network through multiple controllers [5]. ONOS is an SDN network operating system targeted specifically at the service provider and mission critical networks. In addition, ONOS enables easier application development to control OpenFlow ready and legacy devices. It contains a cluster of controllers that work together to manage the applications and the network devices. As the network devices are an increase or network control applications are an increase, ONOS can scale by adding additional controllers into the cluster [16].

3.6.4.1. ONOS Architecture

ONOS is physically distributed and logically centralized controller and it is distributed in a form of cluster. ONOS controllers in the cluster interact with all the other controllers. Controllers are logically connected in a full mesh in a peer to peer way by using a specific TCP port 9876. The controllers send and accept keep alive messages to or from other controllers to monitor the cluster members [47, 64].

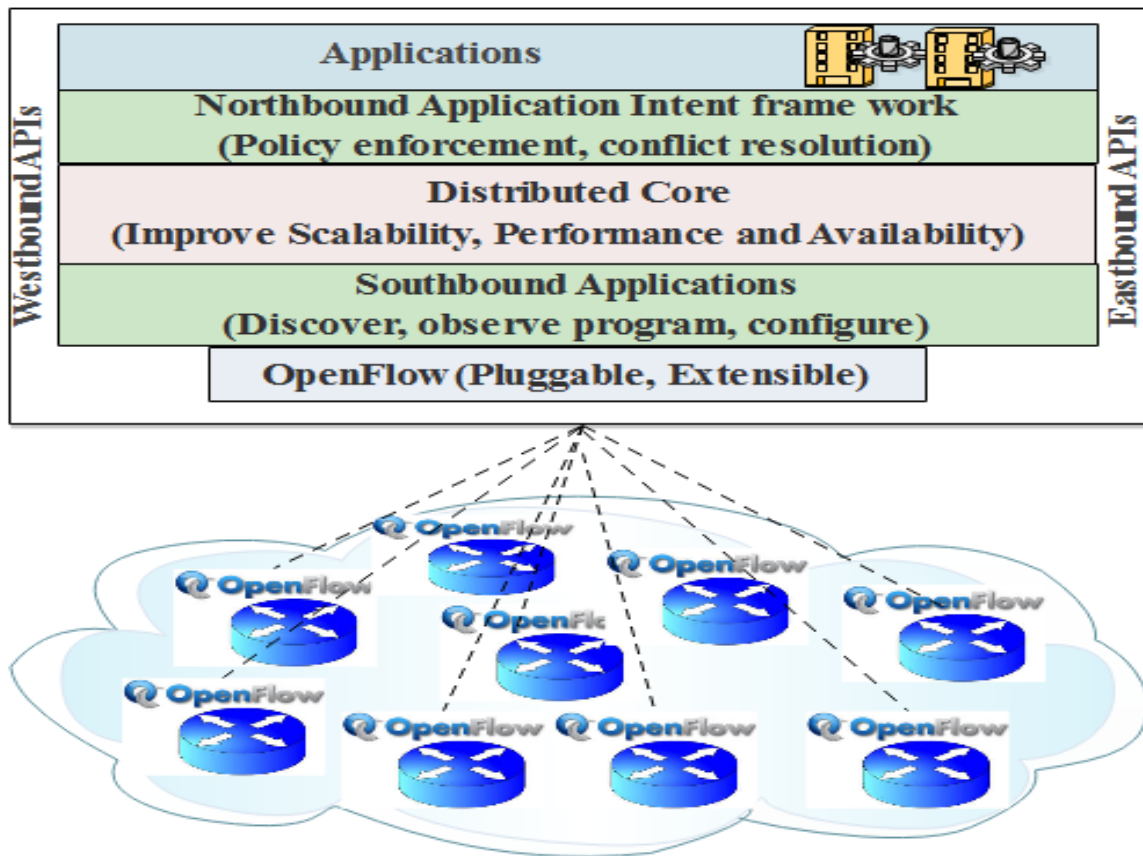


Figure 3-4: Distributed ONOS Architecture [63].

ONOS provides northbound and southbound API and it has four basic features [16, 47, 64].

ONOS Distributed Core: It is ONOS core architecture, which manages topology stores, provision of consistency, availability and cluster state. This provides scalability, high availability, and performance of SDN control plane. The distributed core provides messaging, state management and leader election services to and between controllers.

ONOS Northbound APIs: It provides an abstract view of the network to the application layer. In addition, it provides information related to network topology, devices, and links. Also, it provides application intent framework and global network view to network applications. The application intent framework allows the application to specify high-level intents without underlying rules. The global network view provides the application with a view of the network, hosts, switches, links and any other state associated with the network such as utilization.

ONOS Southbound APIs: This API module contains the interface which abstracts the messages and operations required to control a device. It allows plugin for various southbound protocols and devices, where a plugin maps or translates general network element description or operation on the device to the specific and vice versa. ONOS southbound API uses OpenFlow, NetConf, OSPF, OVSD, PCEP and BGP protocols. OpenFlow is the best protocol for ONOS controller.

ONOS Software Modularity: It makes easy to develop, debug, maintain and upgrade ONOS as a software system by a community of developers and by the providers. It is how the software is

structured into components and how those components relate to one another. It enables new applications or new protocol adapters can be added as needed.

3.6.4.2. ONOS Controller Clustering

Clustering is SDN controllers distributing method, which makes a set of strongly connected SDN controllers work together as a single controller. It helps us to create and run multiple ONOS controllers. When one of the ONOS controllers fails in a cluster, resources are redirected and the workload is redistributed to other working ONOS controller. In the cluster, OpenFlow switches are connected to more than one ONOS controller in a cluster. Then the switch determines which controller should be the master and which should be the standby or slave. This is very useful for fault tolerance and high availability purposes [22, 45, 63].

ONOS cluster must detect and handle controllers joining and leaving the cluster and delegate control over devices, such that any given device has one primary controller. When running multiple ONOS in a cluster, each ONOS has three roles, with respect to forwarding devices. The master roles of ONOS has knowledge of the devices and has full control of or read, write access to all of the connected devices. The standby role of ONOS it has a knowledge of the device and can read the device's state, but not manage or write to the device. None role of ONOS that may or may not have knowledge of the device and cannot interact with it. [10, 22, 63].

3.7. Emulator and Simulator Tool Comparison

Building real network devices, real operating systems, and applications is very expensive to conduct a large test bed environment [65]. This problem is solved by simulation and emulation tools because they have a low cost to build, inexpensive, controllable and flexible to any topology, scalable, repeatable and accessible to many users. Simulator tools have a reality problem to solve this problem emulation tools are developed. The emulator tool uses some real devices running, real applications to and interacts with some simulated devices [66].

Today many SDN network simulator and emulator tools exist such as OMNeT++ [67], Mininet [68], EstiNet [69] and NS-3 and so on. The question is, which of the tool is to be selected and used? To solve this issue, we studied literature, websites, blogs and any available resource providing information about the emulator and simulator tools. We analyzed several existing simulator and emulator tools to collect their properties. We ranked the simulator and emulator tools based on their criteria. Among them, the top three emulators and simulator have been selected for the study to gather their properties. The selected tools are OMNeT++ [67], Mininet [68] and EstiNet [69]. The table below shows the comparison of OMNET++, EstiNet and Mininet tools.

Table 3-2: Comparison of OMNET++, EstiNet, and Mininet [67, 68, 69].

Criteria	OMNeT++ [69]	Mininet [70]	EstiNet [71]
OpenFlow version	1.2.0	1.0 – 1.4	1.0, 1.1, 1.3, 1.3.2
Language	C++	Python	C/C++
Operating System	Windows, Linux	Linux	Linux Fedora 17
Simulation	Yes	No	Yes
Emulation	No	Yes	Yes
Use real Controller	No	Yes	Yes
Result Repeatable	Yes	No	Yes
Scalability	Medium	High	High
Latest version	5.1.2	2.2.1	1.8
Result correctness	No real controller	High	High
Developer	OMNeT++ Team	ON. Lab	EstiNet Technologies
Web site	www.OMNeT++.org	www.mininet.	www.estinet. Com
GUI	Configuration	VND, Miniedit	EstiNet GUI
Open Source	Yes	Yes	No
Accuracy	Yes	Yes	Yes

In the above Table 3-2, we have compared three SDN network emulator and simulator tools. Because this helps us to identify the most appropriate SDN network emulator tool to this thesis. Depend on the above evaluation criteria, we have selected Mininet SDN network emulator for this thesis. Since, Mininet emulator use a real controller, scalable, supports OpenFlow 1.0- 1.5 versions, python based, correct result, middle multiple processors and support GUI. In addition, Mininet supports reference controller (ref), OpenvSwitch controller (OVSC) and remote controller. Also, it supports emulation i.e. it uses real OpenFlow switches, real operating system, real host, real network and real link [67]. Regarding these criteria, we have selected and used Mininet SDN emulator.

3.7.1. OMNeT++

OMNeT++ is an open source modular network simulation tool. OMNeT++ has a generic and flexible architecture. OMNeT++ is a component-based architecture. The components are then assembled into larger components and models by using a high-level language [67]. We have compared this tool with Mininet and EstiNet because it supports OpenFlow switches, OpenFlow controllers, and links. We have not used this tool in this thesis since it does not support emulation, not use a real controller and not support OpenFlow version more than 1.2.0 this version.

3.7.2. EstiNet

Another OpenFlow network simulator and emulator tool are EstiNet. EstiNet uses a unique approach to testing the functions and performances of OpenFlow controllers. By using an innovative simulation methodology, which is called kernel reentering, it combines the advantages of both the simulation and emulation approaches. In a network simulated by EstiNet, each simulated host can run the real Linux operating system and any UNIX based real application program can readily run on a simulated

host without any modification. Although EstiNet also supports the OpenFlow protocol and can be used to simulate SDNs, easy sharing results, and tools with researchers [69]. We have not used this tool in our work because it is not open source and not support 64-bit operating system.

3.7.3. Mininet

In this research, we have used python based Mininet emulation tool to emulate the network topology. It enables to build and configure the network topology using python language. Mininet is a network emulation platform that supports rapid development in SDN using OpenFlow protocol. Mininet is freely available open source software that emulates OpenFlow devices and connects with SDN controllers [17].

Mininet runs a set of end hosts, switches, routers and links on a single Linux kernel by using lightweight virtualization and it allows emulating real world network scenarios. It helps us to scale up to hundreds or thousands of controllers depend on our desired. Mininet also provides a smooth path to running on hardware and surprise is everything can be done in single PC. It is correct, accurate and repeatable regarding their capabilities, performance, and functionalities. It supports single, tree, linear and customized topologies [18, 67].

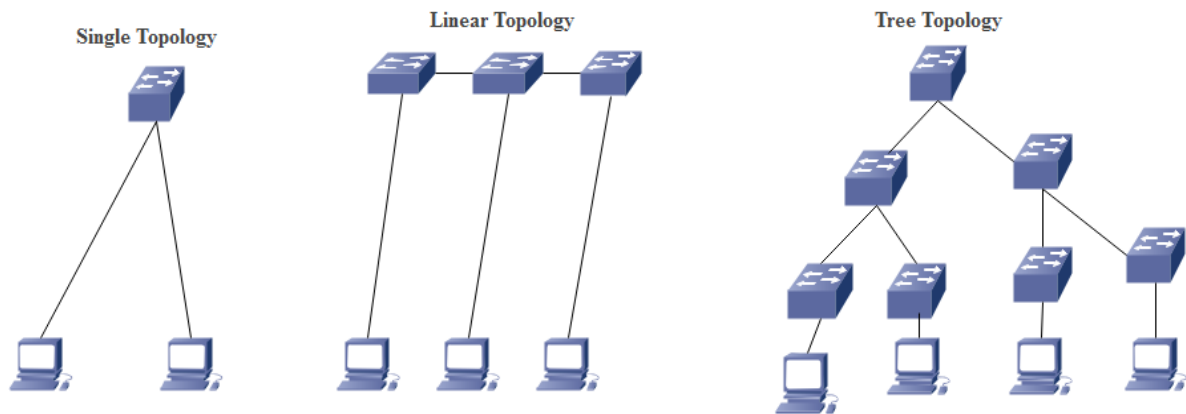


Figure 3-5: Three common topologies in Mininet [18].

As we can see in the Figure 3-5 a single topology has 1 controller, 1 switch, and a definable number of hosts. A linear topology has serial connections with a definable number of switches and hosts. A tree topology consists of multiple topology levels with a definable number of levels such as depth and fan-out. Mininet allows using remote controllers, reference controllers, and inbound controllers. Some other advantages of using Mininet are system level regression test supporting, complex topology testing and CLI which is topology aware and OpenFlow aware. Developed code for an OpenFlow controller, modified switch, or host on Mininet, can move to a real system with minimal deployment. So a network design in Mininet can usually move directly to hardware switches for line rate packet forwarding [18].

```
distributed@mininet-vm:~/topo/Tree TOPO$ sudo mn --controller remote,ip=10.0.3.11 --controller remote,ip=10.0.3.12 --controller remote,ip=10.0.3.13 --topo tree --custom tree3.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s3) (s2, s4) (s2, s5) (s3, s6) (s3, s7) (s4, h1) (s4, h2) (s5, h3) (s5, h4) (s6, h5) (s6, h6) (s7, h7) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0 c1 c2
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet>
```

Figure 3-6: Mininet CLI with switches, hosts, links and three remote ONOS controllers.

Where `sudo mn --custom tree3.py --topo tree --controller remote,ip=10.0.3.11 --controller remote,ip=10.0.3.12 --controller remote,ip=10.0.3.13` is:

Table 3-3: List of Mininet commands to run the topology and to connect with the controller.

<i>Commands</i>	<i>Description</i>
<code>sudo</code>	It starts the Mininet emulator from Ubuntu terminal root.
<code>Mn</code>	It runs the Mininet emulator.
<code>--custom</code>	It used to call the designed topology by using source code file name.
<code>tree3.py</code>	It is source code file name with python language extension.
<code>--topo</code>	It specify the topology type and call the designed topology by its name in this case the topology name is tree.
<code>--controller</code>	It used to specify the controller type and connect the controller with the topology that run on Mininet.
<code>remote</code>	This command states the controller run at remote and control the entire devices.
<code>ip</code>	It identifies the controller IP address like 10.0.3.11, 10.0.3.12 and 10.0.3.13

3.8. Cbench

Cbench is OpenFlow controller performance testing tool used to evaluate OpenFlow-based SDN controllers. Cbench runs in either latency or in throughput mode. In latency mode, Cbench sends a packet in a message to the controller and waits for the response and estimate the amount of response per ms. In throughput mode, Cbench sends a stream of the packet in messages to the controller for specified period of time. In both cases, Cbench counts a number of responses from the controller [18] [35]. In general, cbench measure the performance of an SDN controller with three main algorithms. First, emulate a network topology with a set of OpenFlow switches and hosts. Second, Generate a sequence of packet-in messages from each switch to the controller and capture their responses, i.e., packet-out or flow-mod messages. Third, record the respective statistical information, and calculate the performance of the controller.

Table 3-4: Cbench OpenFlow controller benchmarking tool running options [35].

<i>Option</i>	<i>Description</i>	<i>Default value</i>
-c=controller	STR hostname of controller to connect to	Controllers IP address
-l=loops	INT loops per test	16
-M=MAC addresses	INT unique source MAC addresses per switch	100000
-m=ms-per test	INT test length in ms	1000
-p=port	INT controller port	6633
-r=ranged-test	test range of 1..\$n switches	Off
-s=switches	INT fake \$n switches	16
-t=throughput	Test throughput instead of latency	0
_i=connectdelay	INT delay between groups of switches connecting to the controller	in ms

3.9. Working Environment Setup Challenges

In this thesis, we used SDN controller, SDN emulator, and SDN controller benchmarking tools to meet the objective the research. To setup the working environment requires a lot of dependencies most of these dependency sites have been blocked; this was an obstacle to completed the thesis. This problem was taken a lot of time because there is no manual option to setup these dependencies. The only option is command installation and configuration in Ubuntu terminal. We have tried to install and add different repositories, oracle java PPA repositories, git, LXC, OpenFlow switch images and ONOS controller packages and libraries in Wolkite University, AA university, and private networks. However, like ASTU all the sites were blocked in these universities and private internet cafes. Additionally, dependency applications, libraries, and images were needed more than 3.2 mb/s internet speed and high-performance computers. The above listed things were our challenge.

The other challenge was ONOS releases quarterly at the end of February, May, August, and November. At the beginning of a release cycle, they have a planning meeting and two weeks before the end of the release they freeze integration of new features. When the new version is released most of the previous version commands are not functional to the newly released ONOS version. Even though, with these challenges, we have setup the environment as followed.

3.10. Research Environment Setup Main Processes

In this section, described the main procedures to install and set our working environment under specified hardware requirements. The main procedures are first installed oracle VirtualBox on the host machine and install Ubuntu operating system as a guest machine. Second, installed ONOS controller and dependency applications such as oracle java, ONOS packages, apache karaf, apache

maven, OpenFlow switch, LXC, OpenSSH server, clone the ONOS source code from Github, and crul. Third, installed Mininet SDN network emulator tool and its dependency to emulate the designed network topologies. Then created three containers using Linux container to distributed ONOS controller. Fourth, created distributed three ONOS controllers in a single cluster using Linux container. Fifth, installed cbench and Iperf tools and their dependencies. Then, we have started to understand the ONOS architecture, Mininet architecture and the cbench architecture and their commands. Finally, we have started the experiment.

Finally, in the next chapter, we have presented the expected solutions and contributions of this thesis.

CHAPTER FOUR

4. Proposed Solution

In the first chapter, described a problem, in this chapter presented the solution of the problem. The main contribution of this study was created three distributed ONOS controllers, adapted distributed ONOS in a data center networks, tree, and customized topologies and measured the throughput, latency and TCP bandwidth of ONOS controller. Then analyzed and evaluated the experiment results of distributed ONOS controllers in a cluster.

4.1. Performance Analysis Procedures

Performance is a critical issue in computer networking especially in SDN it needed a research and analysis. In order to achieve the objectives of this research the following listed procedures are followed. The necessary processes that are performed in this research show the separated control plane and data plane, created three ONOS controllers and integrated three ONOS controllers with Mininet and analyzed and evaluated the controller's performance.

To distribute ONOS controller used clustering network distributing techniques, because, it is fully supported by ONOS controller. Clustering is SDN controllers distributing method, which makes a set of strongly connected SDN controllers work together in a logically centralized manner [22]. Clustering ONOS controller helped us to provide high performance, high available, reliable and scalable ONOS controller.

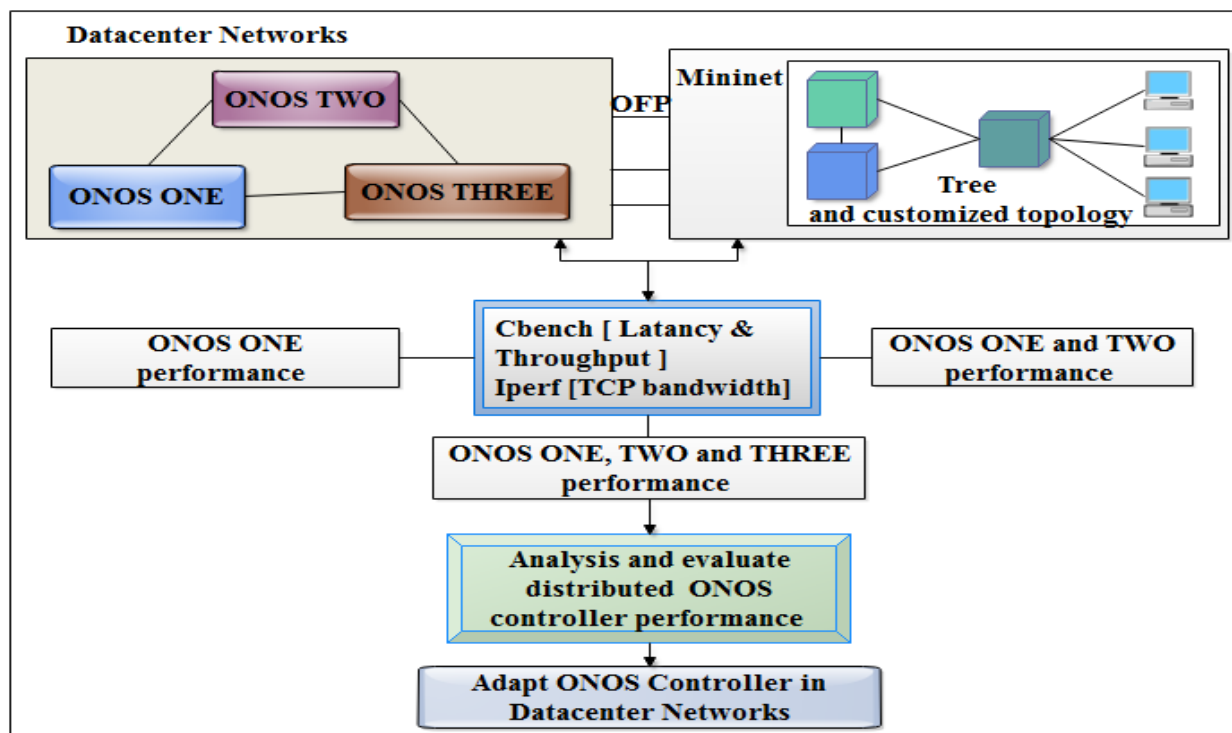


Figure 4-1: Performance analysis procedures.

4.2. Adapt ONOS Controller in a Data center Network

Constructing data center network needs, high-speed 1GB to the 100GB connection [70]. To meet this requirement DCNs needs very expensive vertically integrated control plane and data plane switches and routers. This traditional vendor locked network devices have fixed performance, bandwidth and only accessed by commands, so it is difficult to scale the network and improve the performance of the network in DCN [18].

A large DCN contains tens of thousands of physical servers and hundreds of switches and routers. Such large DCN requires thousands of physical network devices and tens of thousands of switch devices for interconnection purpose. So, in DCN tens of thousands or more network devices control plane and data plane are vertically integrated. Due to this all DCN devices are installed and configured manually and individually this takes a lot of operational time. Therefore, DCN needs a solution to these problems, the only solution is DCNs must use SDN network.

The SDN controller controls and manages network devices remotely. Implementation of SDN in a DCN enables centralized network monitoring, improve performance and higher routing efficiency. In addition, improves bandwidth usage, reduce device cost, and save operational time.

Today modern DCN needed high performance, high available and scalable SDN controller. Depend on different papers that we have reviewed ONOS controller is the best SDN controller. However, ONOS was designed for internet service providers and optical networks. DCN is different from service provider networks in terms of devices arrangement since it requires a lot of servers, network devices, high availability, scalability, reliability and high security than ISPs. In addition, DCN is different from ISP in the network topology. Several DCN topologies exist; even though DCN uses a tree topology. Because tree topology has been applied as the best topology for data center networks this has been approved by several researchers.

Therefore, we have motivated to use SDN ONOS controller in a data center networks. This helps to improve the data center networks performance, availability, scalability and save operational time. In order to use distributed ONOS controller in a data center network, we have contributed the following solutions and procedures.

- ❖ Compared four SDN distributed OpenFlow controllers and selected the best ONOS controller that meets DCN requirements.
- ❖ Compared three emulators and simulator and selected the best Mininet SDN emulator tool.
- ❖ Identified the topologies that are most suitable for DCNs.
- ❖ Designed a tree and customized topology.
- ❖ Selected SDN OpenFlow controller benchmarking tool.
- ❖ Then measured the throughput, latency and TCP bandwidth of distributed ONOS controllers.

- ❖ Then analyzed and evaluated the throughput, latency, and TCP bandwidth results of distributed ONOS controllers.
- ❖ Compared the DCN bandwidth requirement and ONOS controller bandwidth.
- ❖ Finally, we have adapted ONOS controller in a DCN networks.

4.3. Measuring ONOS Controller Performance

Different researchers do their research on the performance of Ryu, Floodlight, OpenDaylight, POX, and NOX SDN centralized controllers. So, have motivated to do performance analysis on a distributed ONOS controller in a data center network. The analysis process of the study is conducted using three ONOS controllers in a cluster. Distributed ONOS controller in the cluster communicates each other using gossip and RAFT protocols [64].

4.3.1. Single ONOS Controller

In the first scenario, we have created a tree and customized topologies using python language. After this run single ONOS controller at remote and run tree topology on Mininet SDN emulator. A single ONOS controller at the remote connected with Mininet topologies and controlled entire network devices. Then using cbench measured and analyzed the throughput and latency results of single ONOS controllers per tree topology. After this killed the existing tree topology and run customized topology. Then performed the above procedure on the single ONOS controller per customized topology and analyzed the results.

Then analyzed each topology throughput and latency result variations in case of the single ONOS controller. After this discussed the result of the two topologies and identified which topology is most suitable for the single ONOS controller. Finally, evaluated the throughput and latency results of single ONOS controller per both topologies. Also, we have measured the bandwidth of ONOS controller using Iperf tool. This enabled us to determine the available TCP bandwidth of ONOS controller.

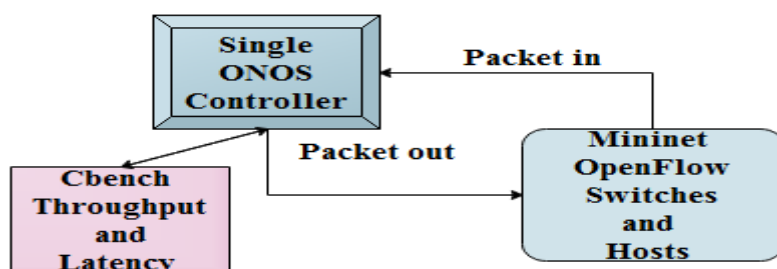


Figure 4-2: Single ONOS controller.

4.3.2. Two ONOS Controllers

In the second setup, we have been created and run two distributed ONOS controllers in a cluster and connected with tree and customized topologies in Mininet. Then measured the throughput and latency of the two distributed ONOS controllers. Like the single ONOS controller, we have analyzed and

discussed the throughput and latency results of two ONOS controllers in a cluster per tree and customized topologies and identified the appropriate topology. Finally, evaluated the performance of a single ONOS controller per two topologies and discussed the results. In addition, we have measured the bandwidth of two ONOS controllers in a cluster using Iperf tool. This used us to adapt distributed ONOS controller in data center networks.

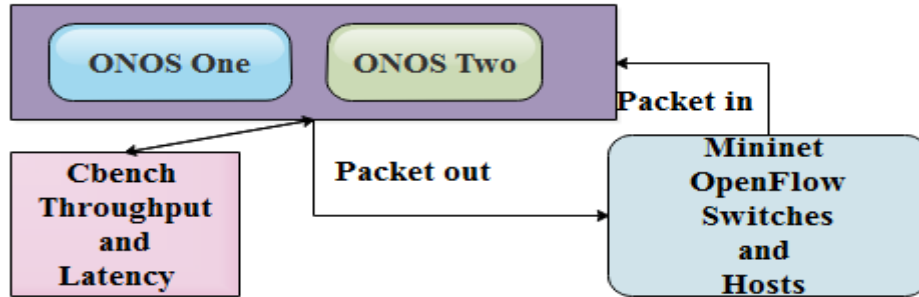


Figure 4-3: Two ONOS Controllers in a cluster.

4.3.3. Three ONOS Controllers

In this scenario, we have been created and run three distributed ONOS controllers in the cluster at the remote. Then connected with the designed tree and customized topologies in Mininet. Then measured the throughput and latency of a three distributed ONOS controllers. Like the single and two ONOS controllers analyzed the throughput and latency results of three ONOS controllers. Next, to this, identified the most appropriate topology for a three distributed ONOS controllers.

Finally, we have taken the average throughput and latency results of single, two and three ONOS controllers per tree and customized topologies and analyzed their performance differences. In addition, we have measured the bandwidth of three ONOS controllers in a cluster using Iperf tool. This used us to adapt distributed ONOS controller in data center networks.

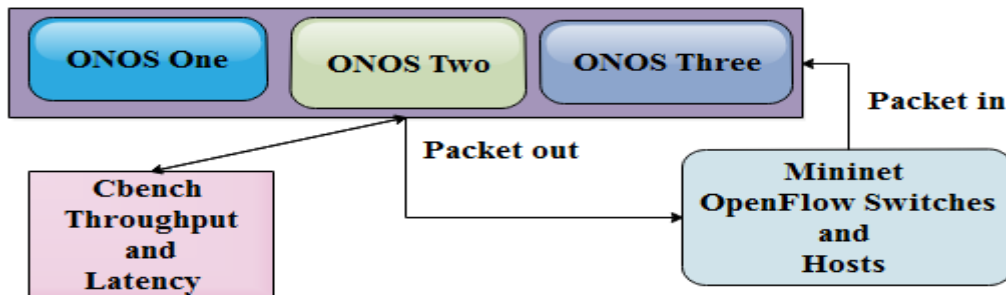


Figure 4-4: Three ONOS Controllers in a cluster.

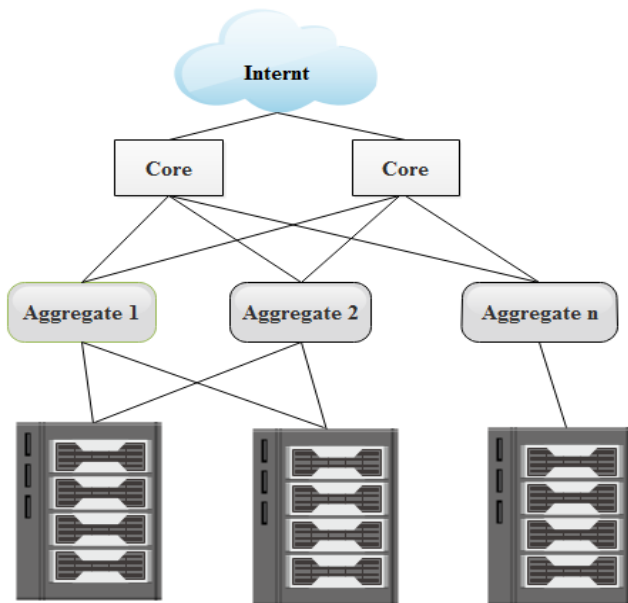
4.4. SDN Network Topologies

SDN network topology is the physical and logical arrangement of OpenFlow SDN controllers, OpenFlow forwarding devices, links, and hosts. SDN network emulator supports single, linear, tree and customized network topologies. So, we have designed tree and customized topologies using python language[22].

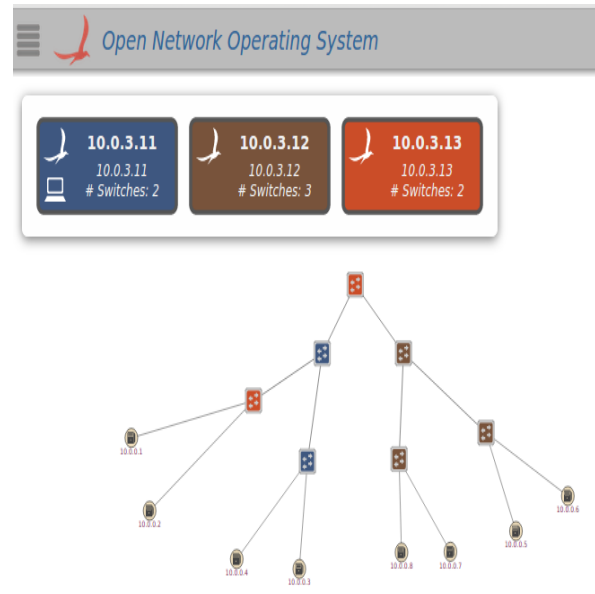
4.4.1. Tree Topology

A tree topology is a network topology that includes at least three levels in a topology hierarchy. It contains k levels and 2 hosts are attached to per switch [22]. We used and designed tree topology because it is most appropriate for data center networks. Data center network has a hierarchical architecture like that of tree topology. Mininet command supports only K level 2 number of switches in each level; this not meets the data center network requirement. Due to these problems we designed tree topology to build K level and X number of switches and hosts.

In this thesis, we have been designed three tree topology. The first topology was designed with 5 levels that contain 30 number of switches and hosts. The second topology was designed with 4 levels that contain 20 number of switches and hosts. And the third topology was designed with 3 levels that contain 10 number of switches and hosts.



A. Data center network architecture [11]



B. Simple tree topology GUI view.

Figure 4-5: Comparison of data center network architecture and tree topology.

4.4.2. Customized Topology

Customized topology is SDN network topology that we design as we like. Unlike a tree, single and linear topology it hasn't a specified design and shape. Custom topology is supported by Mininet emulator but it is not built in Mininet [18]. To design our customized topology, we have been referred and considered different data center network topology architectures. Then, we have designed three customized topologies that meet the data center network topology requirements.

In this thesis, we have been designed three customized topologies. The first topology was designed with 30 number of switches, 180 links, and 30 hosts. The second topology was designed with 20 switches, 100 links, and 20 hosts. And the third topology was designed with 10 switches, 48 links, and 10 hosts.

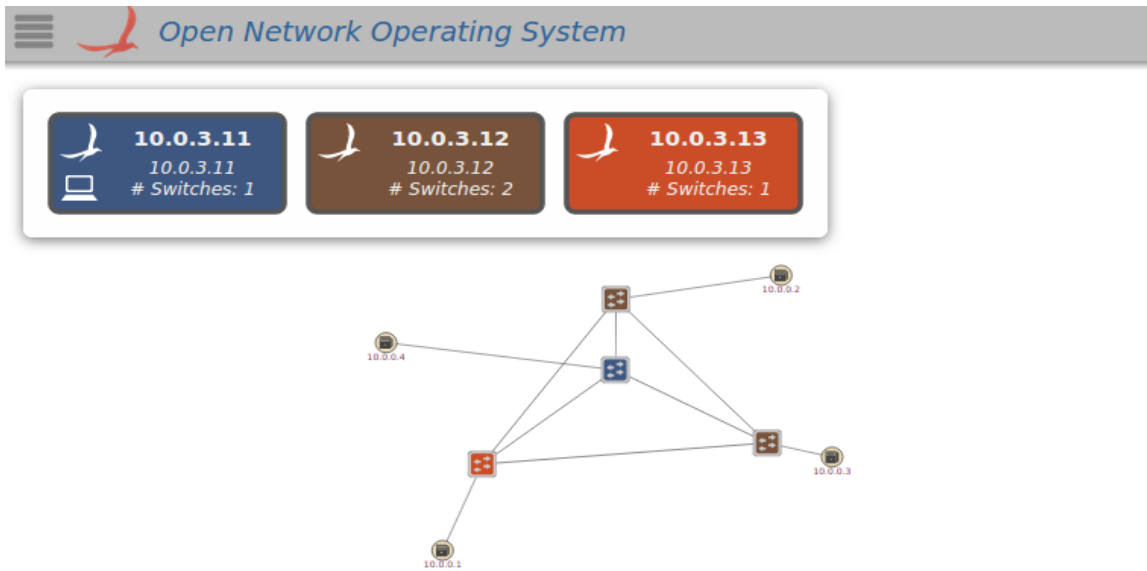


Figure 4-6: Sample customized topology source code GUI view.

4.5. SDN OpenFlow Controller Performance Parameters

We have used throughput and latency performance metrics to perform experiment, analysis and evaluate distributed ONOS controllers in a cluster. These parameters are the two key performance parameters for SDN OpenFlow controllers. These parameters are measured by cbench OpenFlow controller benchmarking tool. In addition, we have been used TCP bandwidth between hosts as a parameter.

4.5.1. Throughput

Throughput is the actual measured performance of the controller. In the concept of cbench, throughput is the total amount of response per second or flows per second that is provided by the controller. Also, it is the number of requests per second that can be processed by the controllers. Throughput can be measured in packet per second, responses per second and flow per second [71]. In this thesis, we used flow per second and measured the throughput of distributed ONOS controllers. The throughput is calculated as when ONOS controller controls all forwarding devices, the packet flows from ONOS to devices this the actual flow per second is the throughput of ONOS controller.

4.5.2. Latency

In the concept of cbench benchmarking tool latency is the amount of response per millisecond that is processed by a controller. The total number of responses received from the controller at the end of the time period can be used to calculate the average time it took the controller to process each packet in initiated from the switches [71]. The latency in cbench measured in responses per nanosecond (ns), responses per microseconds (μ s) and responses per milliseconds (ms). In this thesis, we used

responses per milliseconds (ms) and measured the number of responses per millisecond provided by ONOS controller.

4.5.3. TCP Bandwidth

Bandwidth is the maximum amount of data that can travel through OpenFlow channel, which is provided by SDN controllers. In this thesis, we have measured TCP bandwidth between hosts by using Iperf Mininet tool. We have measured the maximum and minimum TCP bandwidth of ONOS controllers that can generate between all hosts in the network. According to [70], today data center networks needed 1GB up to 100 GB network speed. Regarding this standard, we have used bandwidth as a parameter to use ONOS controller that meets data center networks requirement. In this thesis, we have been measured and analyzed TCP bandwidth in Gigabyte per second.

Finally, in the next chapter, we have presented the performance analysis and result evaluation of this thesis.

CHAPTER FIVE

5. Performance Analysis and Evaluation

We have selected best controller and emulator tools, setup the environment, designed tree and customized topologies and created three ONOS controllers in a cluster. Also, we have selected cbench SDN controllers benchmarking tool and the key SDN controller performance parameters. In this chapter, we have performed distributed ONOS controller performance analysis and evaluation under different scenarios.

5.1. ONOS Controller GUI

The ONOS GUI is a single web page web application provide a visual interface to the ONOS controller or a cluster of multiple ONOS controllers. The GUI must be installed in ONOS and the GUI listen on the port 8181. The ONOS GUI has been developed to work on Google chrome and to access the ONOS GUI we have used <http://10.0.3.11:8181/onos/ui> [73].

5.1.1. Single ONOS Controller GUI

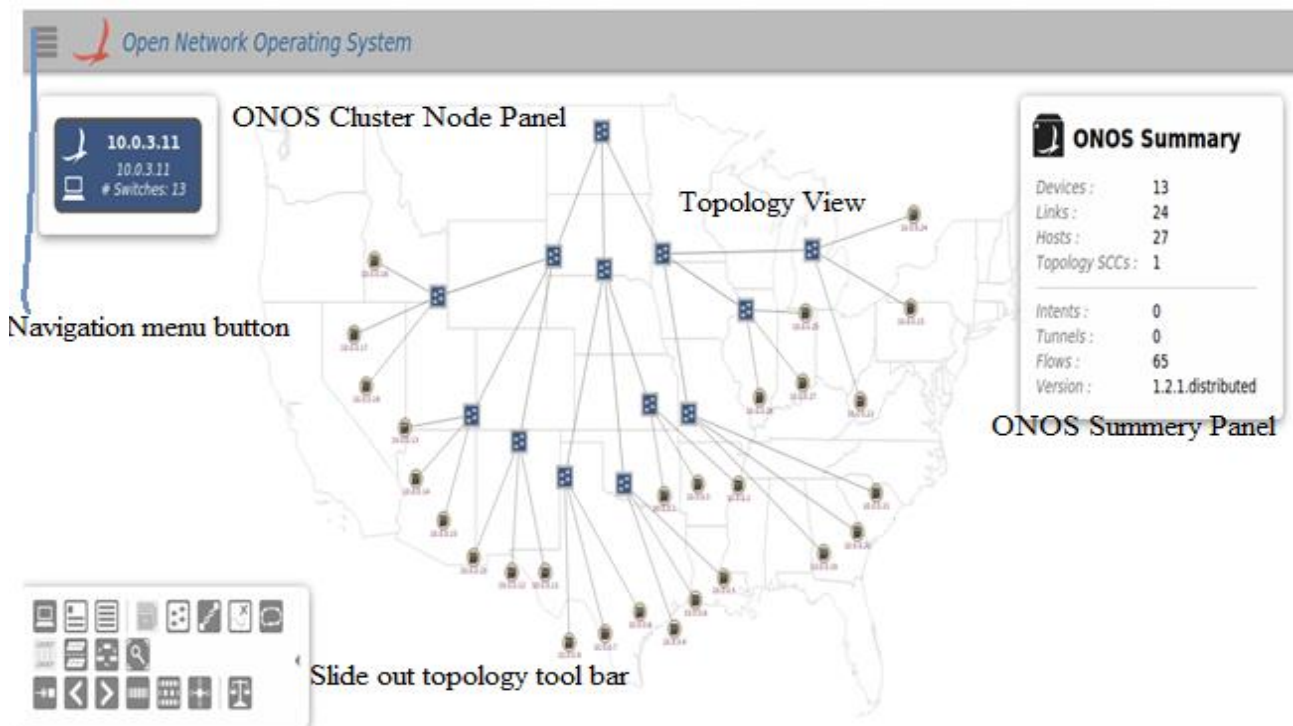


Figure 5-1: Single ONOS controller with tree topology

The above Figure 5-1 shows the GUI of ONOS controller that contains ONOS cluster node panel, ONOS summary panel, background, topology view, navigation menu button and slide out topology toolbar. The GUI shows single ONOS controller with the IP address 10.0.3.11 and Mininet tree topology with 13 devices, 27 hosts, and 24 links. Also, the figure shows the main SDN concept, which

is the separated control plane and data plane. As we can see in the figure, the single ONOS controller control the entire network topology, if the controller stop working, the network is totally failing. The single controller has not only a single point of failure but also decrease the performance. As we can see on the ONOS summary, with the same topology a single ONOS controller can only provide 65 flows but two and three ONOS controllers can handle 3,035 flows.

5.1.2. Two ONOS Controller GUI

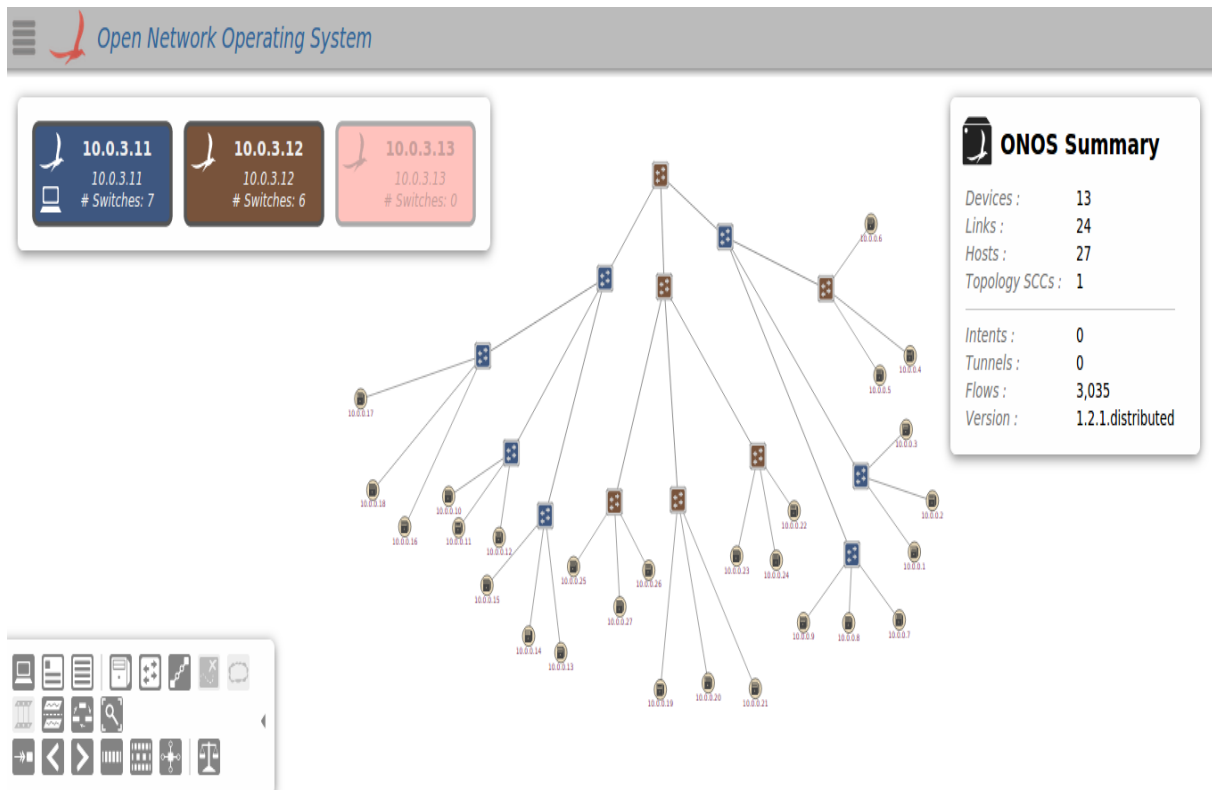


Figure 5-2: Two ONOS controller with tree topology

Figure 5-2 shows two ONOS controllers in a cluster with the IP address 10.0.3.11, 10.0.3.12 and Mininet tree topology with 13 devices, 27 hosts, and 24 links. This figure shows 13 switches, 27 hosts and 24 links controlled by distributed two ONOS controllers in a cluster, but in Figure 5-1 run only a single ONOS controller and controls the entire topology.

As we can see in Figure 5-2 two ONOS controllers control the entire tree topology in physically distributed and logically centralized way. Due to this, the two ONOS controllers such as 10.0.3.11 and 10.0.3.12 are controls 7 and 6 switches respectively. This avoids a single point of failure and provides high network availability. In the figure 5-2 ONOS controller with the IP address 10.0.3.13 not installed in the cluster, so it does not control any device in the network. In the figure, there are two ONOS controllers and switch colors when the switch color similar with ONOS controller color the switch is controlled by that controller.

5.1.3. Three ONOS Controller GUI

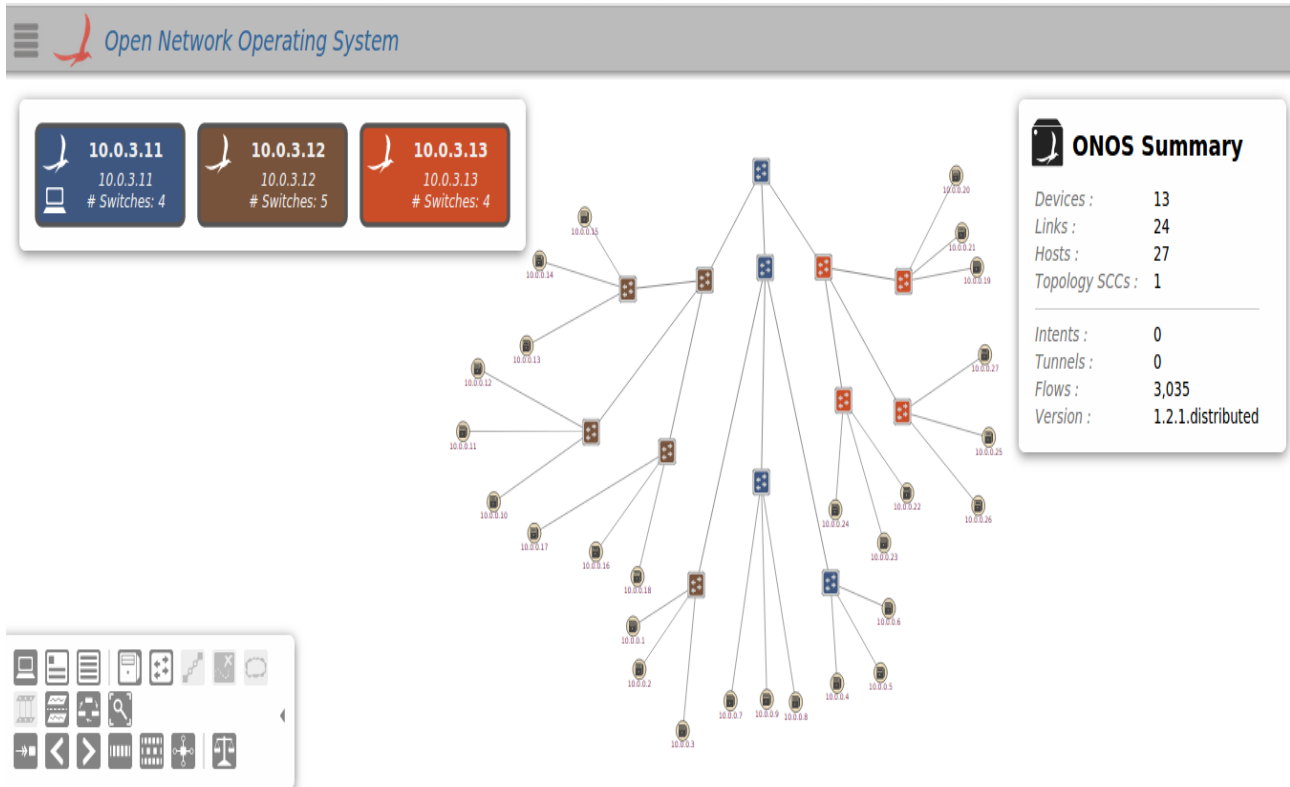


Figure 5-3: Three ONOS controller with tree topology

Figure 5-3 shows three ONOS controllers in a cluster with the IP address of 10.0.3.11, 10.0.3.12, and 10.0.3.13 and Mininet tree topology with 13 devices, 27 hosts, and 24 links. This figure shows the 13 switches, 27 hosts and 24 links controlled by three ONOS controllers. Also, the figure shows the main SDN and the distributed controller concept, which is the separated three distributed ONOS control plane and data plane.

As we can see in Figure 5-3 three ONOS controllers control the entire network in physically distributed and logically centralized way. Due to this, the three ONOS controllers such as 10.0.3.11, 10.0.3.12 and 10.0.3.13 are controls 4, 5 and 4 switches respectively. This increases the network availability and avoids a single point of failure. In the figure, there are three ONOS controllers and switch colors when the switch color similar with ONOS controller color the switch is controlled by that controller or the controller is the master of that device. In general, if one controller in the cluster failed the other controller start to control the devices that were controlled by the failed controller.

5.2. Designed Tree and Customized Topology GUI

In this section, we have presented the graphical view of three ONOS controllers in a cluster, which is connected with our designed tree and customized topology.

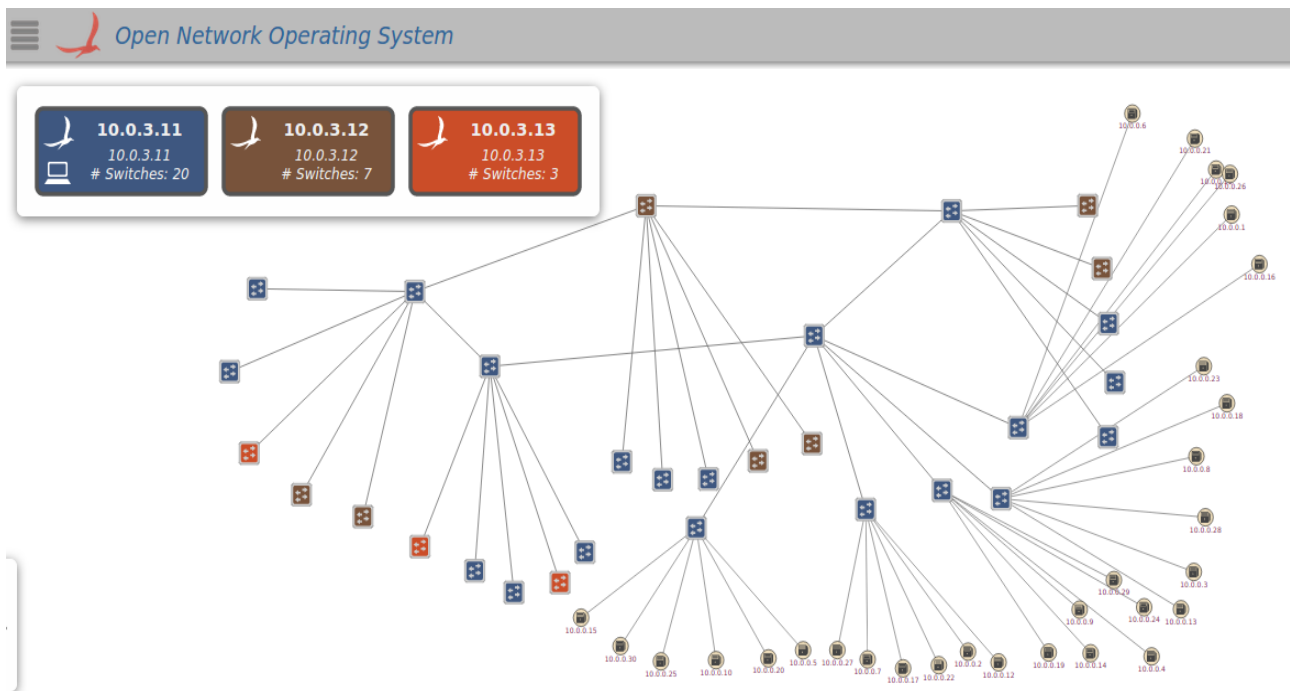


Figure 5-4: The designed customized topology graphical view with 30 switches and hosts.

Figure 5-4 shows our customized topology that contains three ONOS controllers, 5 data center core switches, 25 data center switches, 30 hosts and 180 links. Each 5 data center core switches represent 5 data centers at a different location. The network operators no need to configure each device in each 5 data centers. Because ONOS controller is control plane that is separated from each forwarding devices and placed at remote to controller these 30 switches and 30 hosts. This enables three distributed ONOS controller control each 5 data center networks in one single room. The network operator only configures, improve and manage the three ONOS controllers in a cluster. This topology helped us to measure the amount of flows per second that is provided by ONOS controller and a number of responses per ms.

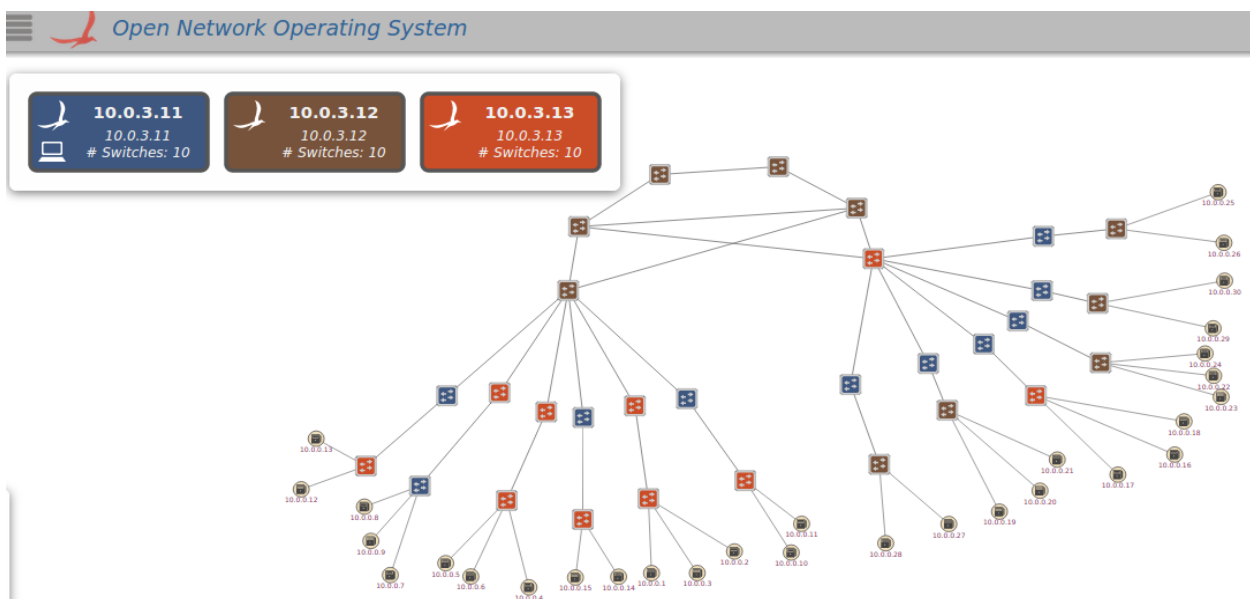


Figure 5-5: The designed tree topology graphical view with 30 switches and hosts.

The above Figure 5-5 shows our tree topology that contains three ONOS controllers, 6 core switches, 12 aggregate switches, 12 edge switches 30 hosts and 62 links. This helped us to measure the amount of flows per second that is handled by ONOS controller and a number of responses in a millisecond per tree topology. If one core switch failed in tree topology, ONOS controller used the remaining switches and another network path.

5.3. ONOS Controller Performance Analysis

In this section, we have analyzed the distributed ONOS controller throughput and latency. To analysis the throughput and latency of ONOS controller we have built the following experiment scenarios.

First, we have specified the number of switches and hosts in the tree and customized topology. In both topologies, we have used 10, 20 and 30 numbers of switches and hosts and a different number of links. We have designed and used 6 topologies and each 6 topologies connected with single, two and three ONOS controllers. In another word, we have used 36 scenarios for both throughput and latency experiment.

Each single, two and three ONOS controllers connected with each 3 tree and 3 customized topologies has been tested 10 times because the minimum standard is 5 times. In addition, each throughput and latency scenario has been individually tested 16 loops per test. This enabled us to get reliable and accurate throughput and latency results of distributed ONOS controllers and to determine the best topology. We have considered the average throughput and latency results to analysis and evaluate the performance of ONOS controller.

5.3.1. Throughput Analysis

In the throughput mode cbench test, distributed ONOS controller in a cluster is analyzed for how many total flows it can process in a second. We tested a single, two and three ONOS controller throughput performance with respect to two topologies under a different number of switches, hosts, and link. We have calculated the average throughput differences to compare the throughput between three controllers and two topologies. We have been run the cbench command at Ubuntu terminal in the cbench directory and measured the throughput of ONOS controller in a cluster. We have run the cbench line of command to measure the throughput of ONOS controller in a cluster with -t option:

```
distributed@mininet-vm:/home/mininet/oflops/cbench$ taskset -c 0-1 cbench -c 10.0.3.11 -p 6633 -m 1000 -l 16 -M 100000 -i 70 -s 10 -t
cbench: controller benchmarking tool
  running in mode 'throughput'
  connecting to controller at 10.0.3.11:6633
  faking 10 switches offset 1 :: 16 tests each; 1000 ms per test
  with 100000 unique source MACs per switch
  learning destination mac addresses before the test
  starting test with 0 ms delay after features_reply
  ignoring first 1 "warmup" and last 0 "cooldown" loops
  connection delay of 70ms per 1 switch(es)
  debugging info is off
```

Figure 5-6: Cbench running in throughput mode.

5.3.1.1. Single ONOS Controller Throughput Analysis

To analysis a single ONOS controller, we have been run cbench in a throughput mode and cbench emulates a 10, 20, and 30 number of OpenFlow switches that all communicate with a single ONOS controller. Each emulated switch sends a number of new flow messages to a single ONOS controller and ONOS sends a response to each emulated switch requests. The cbench records the total amount of single ONOS controller response. Finally, at the end of the loop, the cbench calculated and displayed the maximum, minimum and average throughput results of a single ONOS controller.

Table 5-1: Single ONOS controller throughput results in flow per second.

<i>Single ONOS controller throughput results</i>						
Throughput	Tree Topology			Customized Topology		
	10 switches 10 Hosts 20 Links	20 switches 20 Hosts 41 Links	30 switches 30 Hosts 62 Links	10 switches 10 Hosts 48 Links	20 switches 20 Hosts 100 Links	30 switches 30 Hosts 180 Links
Maximum	64584.40	79987.80	82849.50	64544.94	78967.50	82698.87
Minimum	61785.60	78694.49	82284.20	56785.49	77684.26	81244.64
Average	63185.00	79341.15	82566.85	60665.22	78325.88	81971.76

The above Table 5:1 shows the maximum, minimum and the average throughput results of the single ONOS controller under three customized and three tree topologies with a different number of switches, hosts, and links.

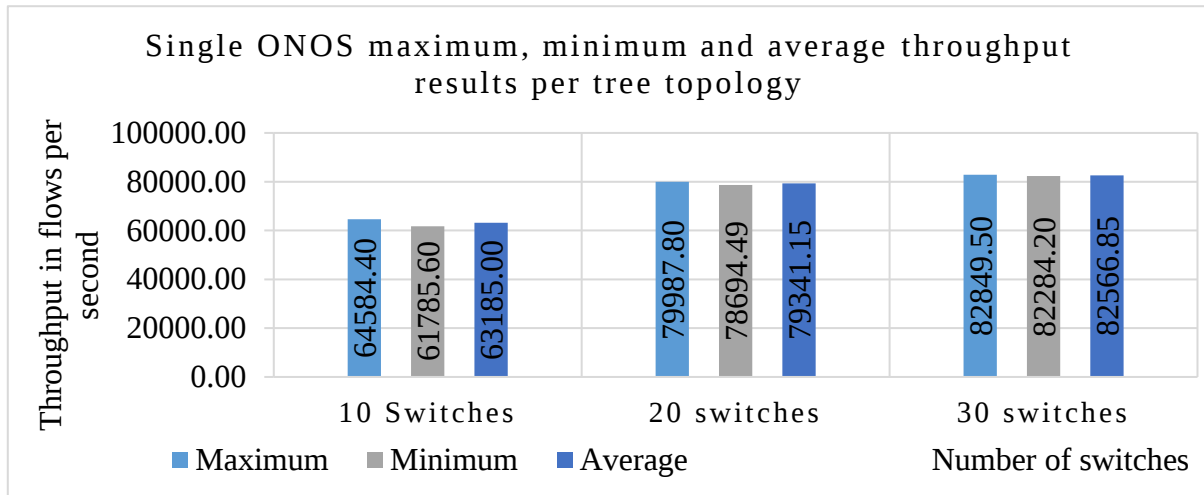


Figure 5-7: Single ONOS throughput results per tree topology.

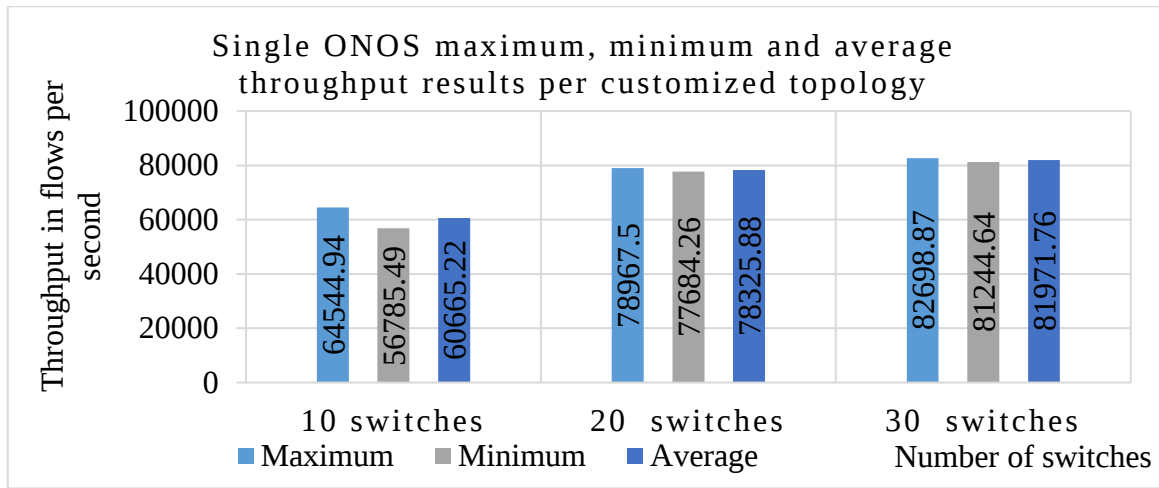


Figure 5-8: Single ONOS throughput results per customized topology.

The above Figure 5-7 and Figure 5-8 shows single ONOS controller throughput results in terms of tree and customized topologies respectively. In both topology, when the number of devices increases the throughput of the single ONOS controller also increase. As we increased the number of switches to 10, 20 and 30 the number of average flows per second increase to 63185, 79341.15 and 82566.85 respectively in tree topology and 60665.22, 78325.88 and 81971.76 in customized topology.

In the Figure 5-7, a single ONOS controller has high throughput per tree topology than customized topology. The single ONOS controller in a tree topology with 10, 20 and 30 switches has 2519.78, 1015.27 and 595.09 more flows per second than customized topology respectively. So, in this thesis for single ONOS controller tree topology is better topology than customized topology. As we have observed throughput tests more than 30 switches single ONOS controller generates inconsistency throughput results.

5.3.1.2. Two ONOS Controller Throughput Analysis

To analysis the throughput of two distributed ONOS controller, we have been run cbench in a throughput mode and cbench emulates 10, 20, and 30 number of OpenFlow switches that all communicate with two ONOS controllers in a cluster. Each emulated switch sends a number of new flow messages to the two ONOS controller and ONOS sends a response to each emulated switch message. The cbench records the total amount of two ONOS controller response. Finally, at the end of the loop, the cbench calculate and display the maximum, minimum and average throughput results of a two ONOS controller.

Table 5-2: Two ONOS controller throughput results in flow per second.

<i>Two ONOS controller throughput results</i>						
Throughput	Tree Topology			Customized Topology		
	10 switches 10 Hosts 20 Links	20 switches 20 Hosts 41 Links	30 switches 30 Hosts 62 Links	10 switches 10 Hosts 48 Links	20 switches 20 Hosts 100 Links	30 switches 30 Hosts 180 Links
Maximum	105948.42	109846.17	118978.56	105647.42	108829.00	118246.47
Minimum	105887.26	109685.50	118843.78	105481.20	108686.40	118123.60
Average	105917.80	109765.80	118911.20	105564.30	108757.70	118185.00

The above Table 5:2 shows the maximum, minimum and the average throughput results of two ONOS controllers under three customized and three tree topologies with a different number of switches, hosts, and links.

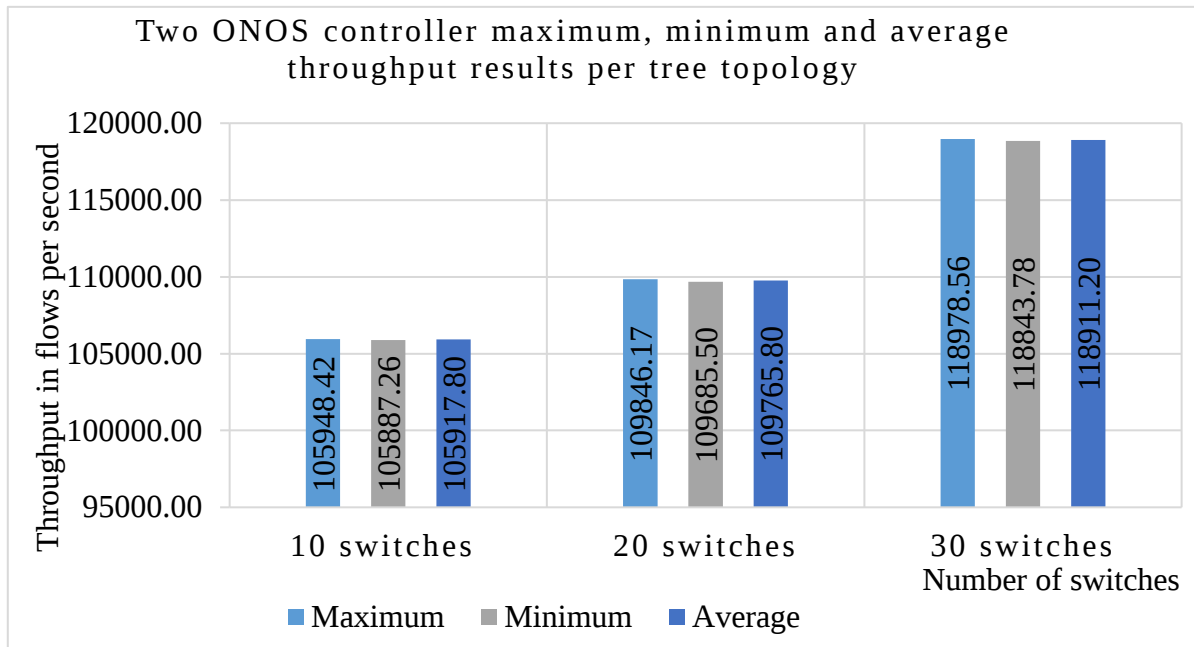


Figure 5-9: Two ONOS controller throughput results per tree topology.

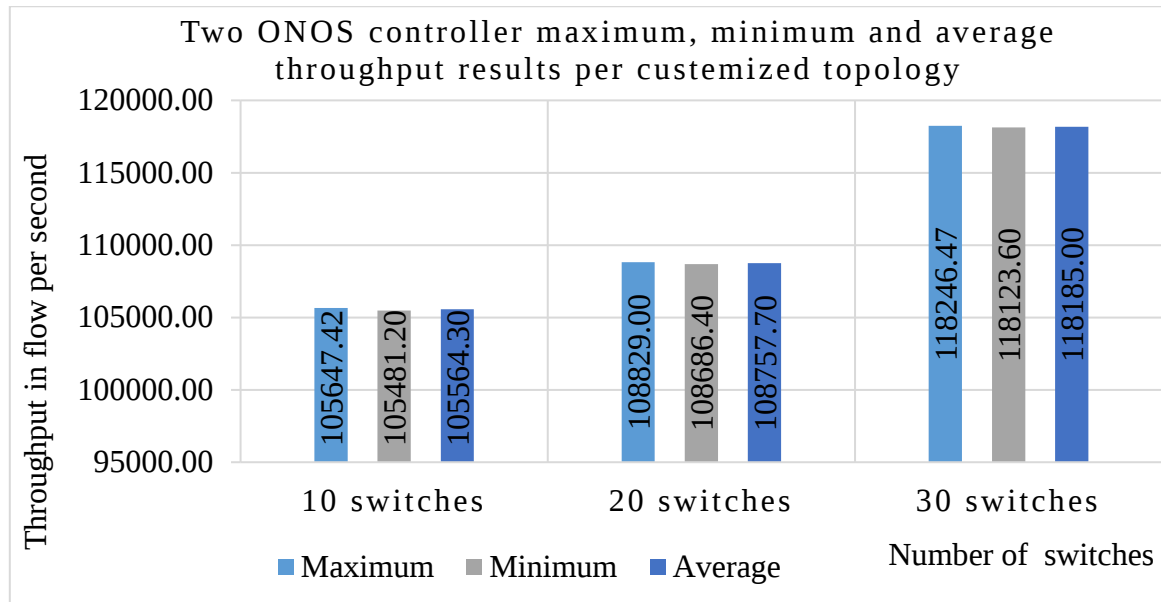


Figure 5-10: Two ONOS controller throughput results per customized topology.

The above Figure 5-9 and Figure 5-10 shows two ONOS controller throughput results in terms of tree and customized topologies respectively. Like single ONOS controller, in both topologies, the throughput of two ONOS controllers is increased, when the number of switches is increased. In two ONOS controllers as we increased the number of switches to 10, 20 and 30 the number of average flows per second increase to 105917.80, 109765.80 and 118911.20 respectively in tree topology and 105564.30, 108757.70 and 118185.00 in customized topology.

As we can see in the Figure 5-9, in two ONOS controller's tree topology has higher throughput than customized topology. Two ONOS controller in a tree topology with 10, 20 and 30 switches has 353.50, 1008.10 and 726.20 more flows per second than customized topology respectively. So, in this thesis for two ONOS controller tree topology is better topology than customized topology.

5.3.1.3. Three ONOS Controller Throughput Analysis

To analysis the throughput of three ONOS controllers in a cluster, we have been run cbench in a throughput mode and cbench emulates 10, 20 and 30 number of OpenFlow switches that all communicate with three ONOS controllers in a cluster. Each emulated switch sends a number of new flow messages to the three ONOS controllers and ONOS sends a response to each emulated switch message. The cbench records the total amount of three ONOS controller response. Finally, at the end of the loop, the cbench calculate and display the maximum, minimum and average throughput results of a three ONOS controller.

Table 5-3: Three ONOS controller throughput results in flow per second.

<i>Three ONOS controller throughput results</i>						
Throughput	Tree Topology			Customized Topology		
	10 switches 10 Hosts 20 Links	20 switches 20 Hosts 41 Links	30 switches 30 Hosts 62 Links	10 switches 10 Hosts 48 Links	20 switches 20 Hosts 100 Links	30 switches 30 Hosts 180 Links
Maximum	217943.23	220989.45	229785.24	216843.62	219978.92	228749.24
Minimum	217889.67	220897.96	228992.60	216789.98	219846.82	228692.60
Average	217916.50	220943.70	229388.90	216816.80	219912.90	228720.90

The above Table 5:3 shows the maximum, minimum and the average throughput results of three ONOS controllers under three customized and three tree topologies with a different number of switches, hosts, and links.

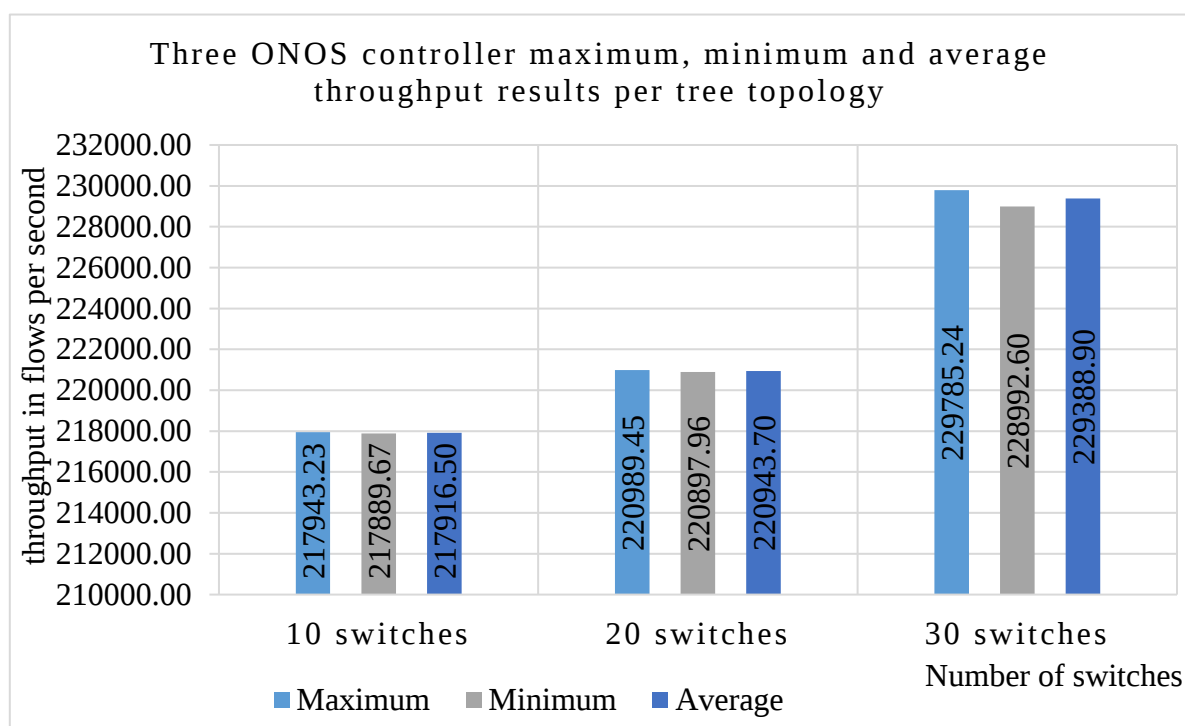


Figure 5-11: Three ONOS controller throughput results per tree topology.

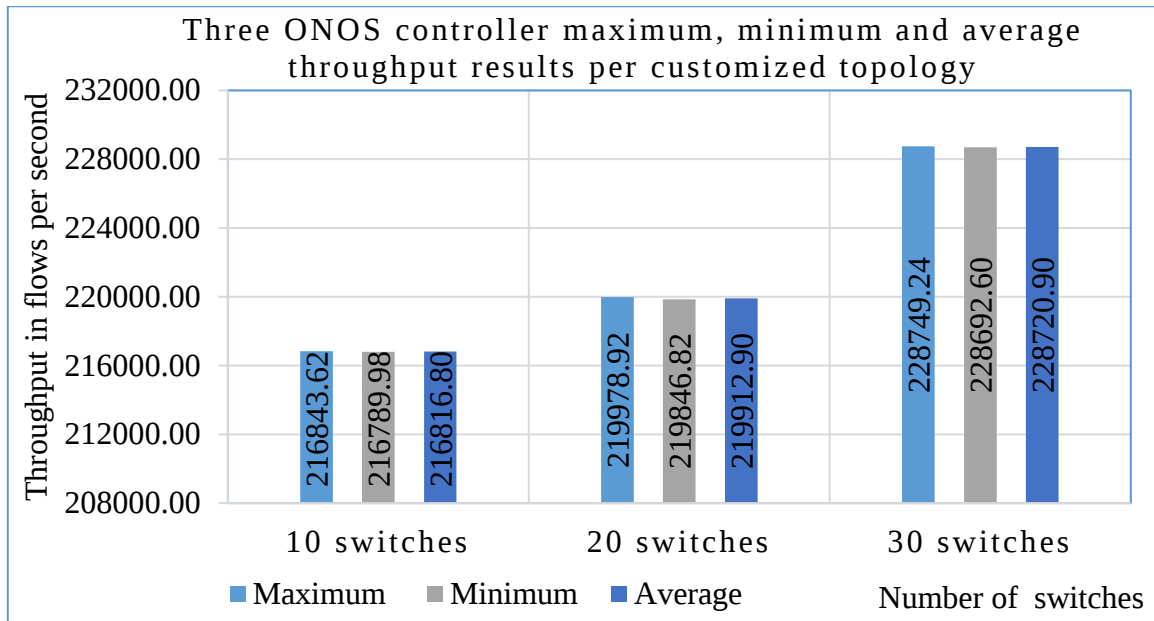


Figure 5-12: Three ONOS controller throughput results per customized topology.

The above Figure 5-11 and Figure 5-12 shows three ONOS controller throughput results in terms of tree and customized topologies respectively. Like single and two ONOS controllers, in both topology the throughput of two ONOS controllers is increased, when the number of switches is increased. In three ONOS controller as we increased the number of switches to 10, 20 and 30 the number of average flows per second increase to 217916.5, 220943.70 and 229388.90 respectively in tree topology and 216816.80, 219912.90 and 228720.90 in customized topology.

As we can see in the Figure 5-11, three ONOS controllers per tree topology have higher throughput than customized topology. Three ONOS controller in a tree topology with 10, 20 and 30 switches has 1099.70, 1030.80 and 668.00 more flows per second than customized topology respectively. So, in this thesis for three ONOS controller tree topology is better topology than customized topology.

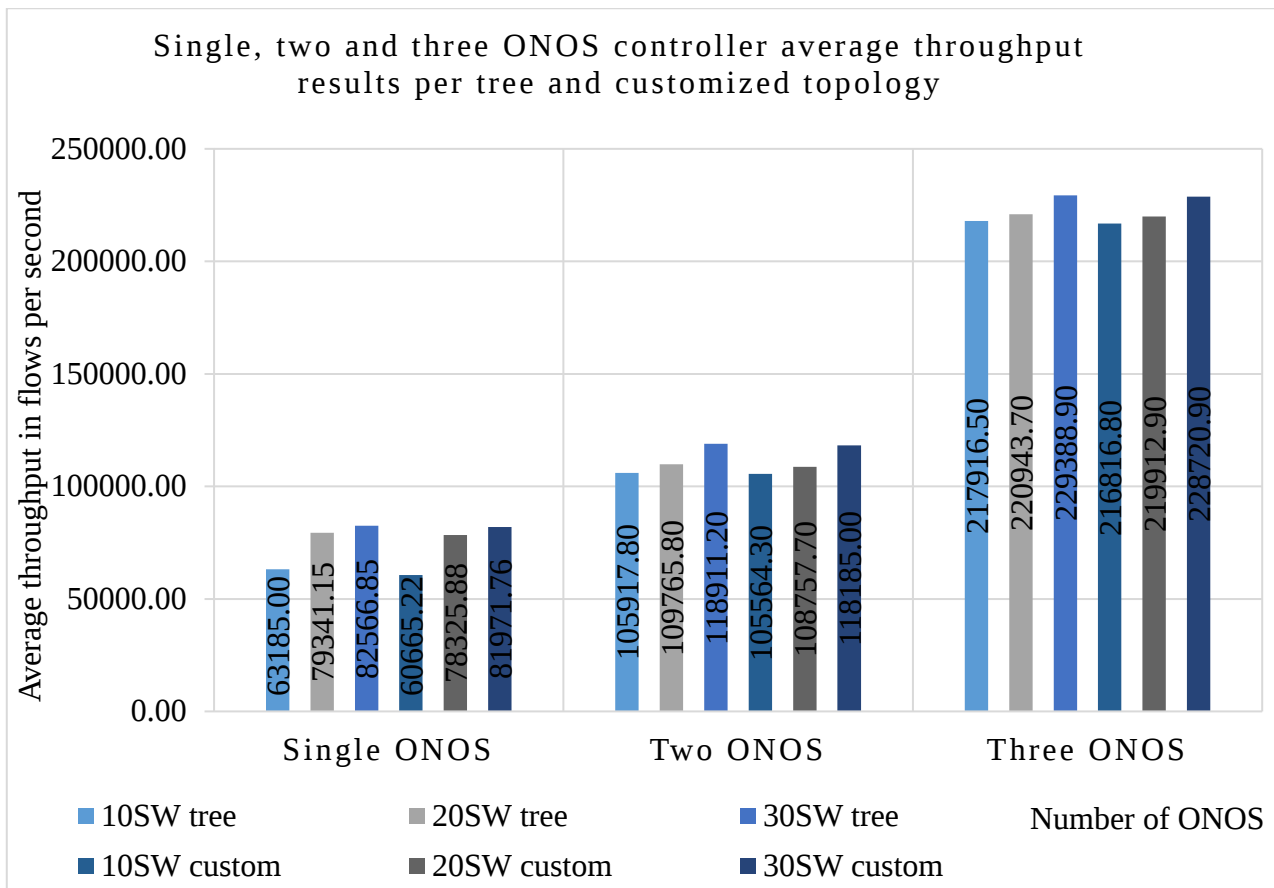


Figure 5-13: Single, two and three ONOS controller average throughput results per tree and customized topology.

As we can see in the Figure 5-13 the single, two and three ONOS controller throughput results increase when the number of switches is increased. Three ONOS controller provides very high throughput than single and two ONOS controllers in both topologies.

The single ONOS controller has average throughput results from 63185.22 to 82566.85 flow per second in tree topology and 60665.22 to 81971.76 flow per second in customized topologies under a different number of devices. Two ONOS controller has average throughput results from 105917.80 to 118911.20 flow per second in tree topology and 105564.3 to 118185 flow per second in customized topologies under a different number of devices. Three ONOS controller has average throughput results from 217916.50 to 229388.90 flow per second in tree topology and 216816.80 to 228720.90 flow per second in customized topologies under a different number of devices.

Finally, as we can see in Figure 5-13, three distributed ONOS controller provides high throughput than single and two ONOS controllers. This enabled to increase the throughput of ONOS controller by increasing the number of ONOS controller in a cluster. Using distributed multiple ONOS controllers in a cluster can provide a large amount of throughput than single and two ONOS controllers in a cluster.

5.3.2. Latency Analysis

The latency of ONOS controller means that how much time ONOS takes to process a single packet. We tested the distributed ONOS controller performance latency. We used the same number of switches, hosts, and links with throughput analysis, to measure and analysis the latency. This enabled to measure a number of responses that flowed in 1 millisecond (ms). We have calculated the average response per second result differences to compare the latency between three ONOS controllers per two topologies. In order to measure the latency of ONOS controller in the cluster, we have been run the cbench line of command at Ubuntu terminal in the cbench directory without -t option:

```
distributed@mininet-vm:/home/mininet/oflops/cbench$ taskset -c 0-1 cbench -c 10.0.3.11 -p 6633 -m 1000 -l 16 -M 100000 -i 70 -s 10
cbench: controller benchmarking tool
  running in mode 'latency'
  connecting to controller at 10.0.3.11:6633
  faking 10 switches offset 1 :: 16 tests each; 1000 ms per test
  with 100000 unique source MACs per switch
  learning destination mac addresses before the test
  starting test with 0 ms delay after features_reply
  ignoring first 1 "warmup" and last 0 "cooldown" loops
  connection delay of 70ms per 1 switch(es)
  debugging info is off
```

Figure 5-14: Cbench running in latency mode.

5.3.2.1. Single ONOS Controller Latency Analysis

To analysis the latency of single ONOS controller, we have been run cbench in a latency mode and cbench emulates 10, 20 and 30 number of OpenFlow switches that all communicate with the single ONOS controller. Each emulated switch sends a number of new flow messages to a single ONOS controller, waits for the appropriate responses and cbench records a number of responses per ms of the single ONOS controller. Finally, at the end of the loop, the cbench calculate and display the maximum, minimum and average responses per second of the single ONOS controller.

Table 5-4: Single ONOS controller latency results in response per ms.

<i>Single ONOS controller latency results</i>						
Latency	Tree Topology			Customized Topology		
	10 switches 10 Hosts 20 Links	20 switches 20 Hosts 41 Links	30 switches 30 Hosts 62 Links	10 switches 10 Hosts 48 Links	20 switches 20 Hosts 100 Links	30 switches 30 Hosts 180 Links
Maximum	68.34	82.62	96.55	65.45	80.36	90.78
Minimum	63.24	79.23	92.35	60.33	77.64	88.25
Average	65.79	80.93	94.45	62.89	79.00	89.51

The above Table 5:4 shows the maximum, minimum and the average latency results of the single ONOS controller under three customized and three tree topologies with a different number of switches, hosts, and links. Single ONOS controller latency results in Table 5-4 are a number of responses per ms.

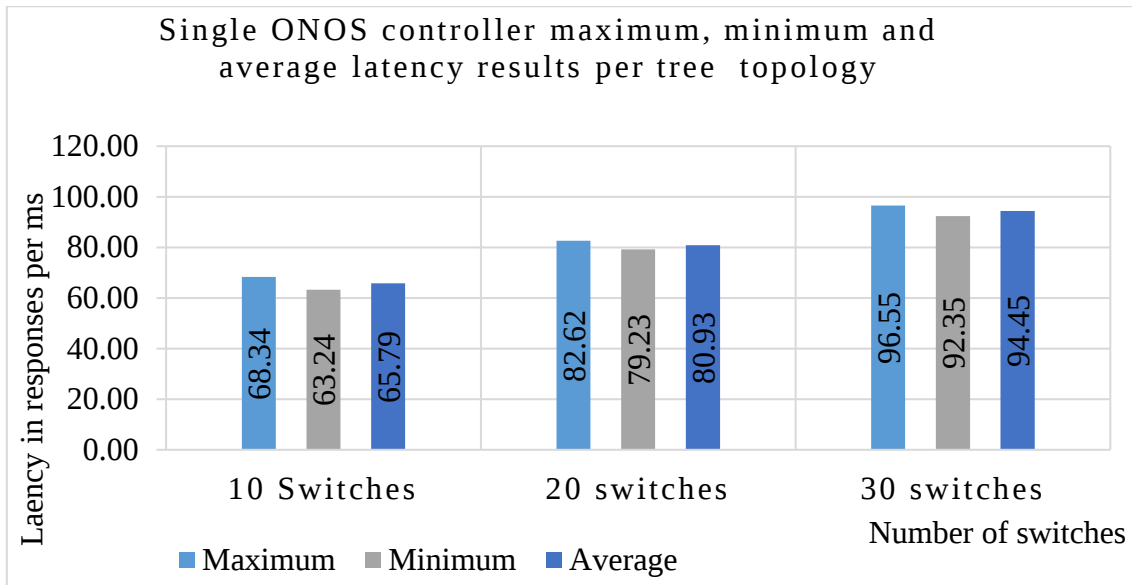


Figure 5-15: Single ONOS controller latency results per tree topology.

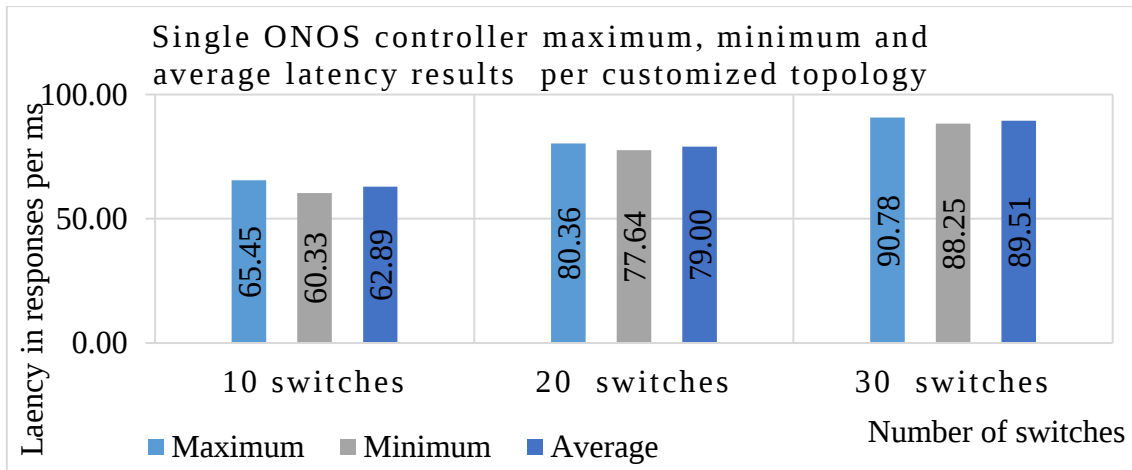


Figure 5-16: Single ONOS controller latency results per customized topology.

The above Figure 5-15 and Figure 5-16 shows the minimum, maximum and average latency results of the single ONOS controller in terms of tree and customized topology respectively. In single ONOS controller under both topologies as we increased the number of switches resulted into slight increase in the number of responses per ms.

As we observed that single ONOS controller provides 65.79 with 10 switches, 80.93 with 20 switches and 94.45 with 30 switches in tree topology and 62.89 with 10 switches, 79.00 with 20 switches and 89.51 with 30 switches in customized topologies.

As we can see in the Figure 5-15, single ONOS controller per tree topology has higher responses per ms than customized topology. The single ONOS controller in a tree topology with 10, 20 and 30 switches has 2.90, 1.93 and 4.93 more responses per ms than customized topology respectively. So, in this thesis for single ONOS controller tree topology is better topology than customized topology. As we have observed latency tests more than 30 switches single ONOS controller generates inconsistency latency results.

5.3.2.2. Two ONOS Controller Latency Analysis

To analysis the latency of two ONOS controllers, we have been run cbench in a latency mode and cbench emulates 10, 20 and 30 number of OpenFlow switches that all communicate with two ONOS controllers. Each emulated switch sends a number of new flow messages to a two ONOS controller, waits for the appropriate flow responses and cbench records a number of responses per ms of two ONOS controllers. Finally, at the end of the loop, the cbench calculate and display the maximum, minimum and average responses per second of two ONOS controllers.

Table 5-5: Two ONOS controller latency results in responses per ms.

<i>Two ONOS controller latency results</i>						
Latency	Tree Topology			Customized Topology		
	10 switches 10 Hosts 20 Links	20 switches 20 Hosts 41 Links	30 switches 30 Hosts 62 Links	10 switches 10 Hosts 48 Links	20 switches 20 Hosts 100 Links	30 switches 30 Hosts 180 Links
Maximum	125.94	133.62	148.98	118.57	127.83	144.25
Minimum	122.87	133.62	146.84	116.48	125.69	139.99
Average	124.41	133.62	147.91	117.53	126.76	142.12

The above Table 5:5 shows the maximum, minimum and the average latency of two ONOS controllers under three customized and tree topologies with a different number of switches, hosts, and links. A two ONOS controller latency results in Table 5-5 are a number of responses per ms.

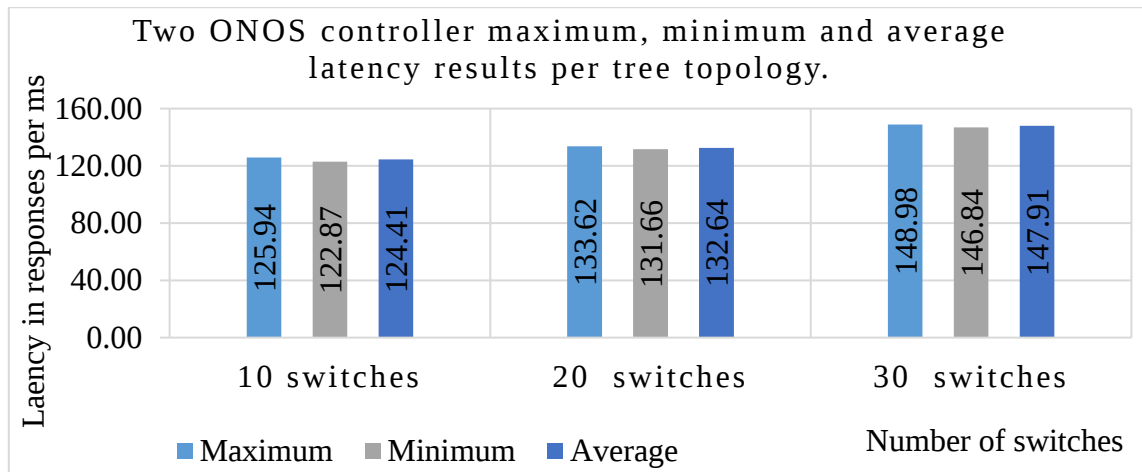


Figure 5-17: Two ONOS controller latency results per tree topology.

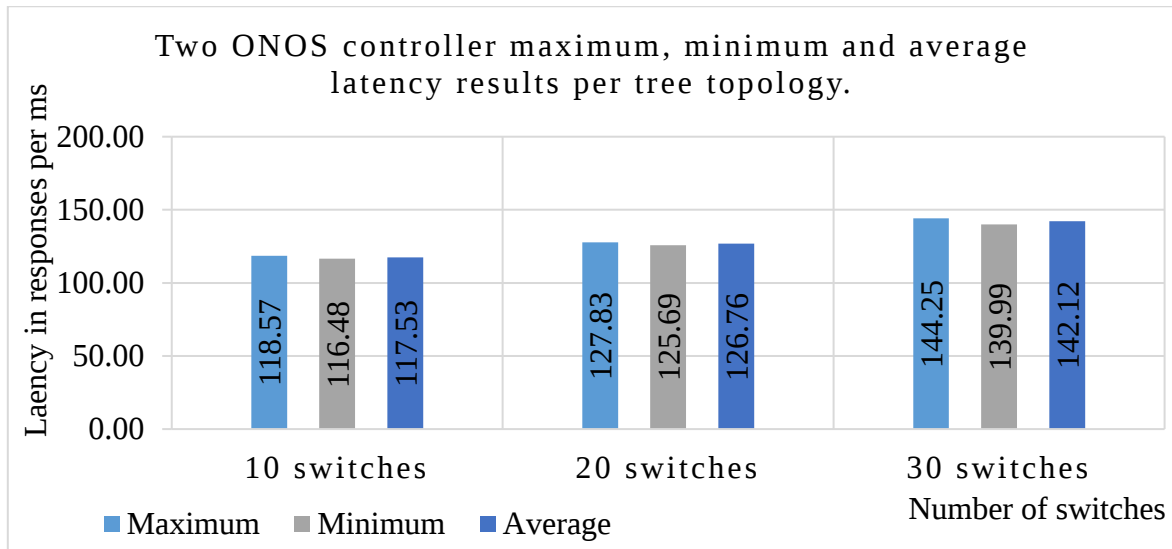


Figure 5-18: Two ONOS controller latency results per customized topology.

The above Figure 5-17 and Figure 5-18 shows the minimum, maximum and average latency results of two ONOS controllers in terms of tree and customized topology respectively. In two ONOS controller under both topologies as we increased the number of switches resulted into slight increase in the number of responses per ms.

As we observed that two ONOS controller provides 124.41 with 10 switches, 132.64 with 20 switches and 147.91 with 30 switches in tree topology and 117.53 with 10 switches, 126.76 with 20 switches and 142.12 with 30 switches in customized topologies responses per ms.

As we can see in the Figure 5-17, two ONOS controllers per tree topology have higher responses in ms than customized topology. Two ONOS controller in a tree topology with 10, 20 and 30 switches has 6.88, 5.88 and 5.79 more responses per ms than customized topology respectively. So, in this thesis for two ONOS controller tree topology is better topology than customized topology.

5.3.2.3. Three ONOS Controller Latency Analysis

To analysis the latency of three ONOS controllers, we have been run cbench in a latency mode and cbench emulates 10, 20 and 30 number of OpenFlow switches that all communicate with three ONOS controllers. Each emulated switch sends a number of new flow messages to a three ONOS controller, waits for the appropriate flow responses and cbench records a number of responses per ms of three ONOS controllers. Finally, at the end of the loop, the cbench calculate and display the maximum, minimum and average responses per second of three ONOS controllers.

Table 5-6: Three ONOS controller latency results in responses per ms.

<i>Three ONOS controller latency results per tree topology</i>						
Latency	Tree Topology			Customized Topology		
	10 switches 10 Hosts 20 Links	20 switches 20 Hosts 41 Links	30 switches 30 Hosts 62 Links	10 switches 10 Hosts 48 Links	20 switches 20 Hosts 100 Links	30 switches 30 Hosts 180 Links
Maximum	252.94	270.98	298.79	246.85	266.98	290.75
Minimum	249.90	267.90	296.43	243.79	263.85	287.69
Average	251.42	269.44	297.61	245.32	265.41	289.22

The above Table 5:6 shows the maximum, minimum and the average latency of three ONOS controllers under three customized and tree topologies with a different number of switches, hosts, and links. A three ONOS controller latency results in Table 5-6 are a number of responses per ms.

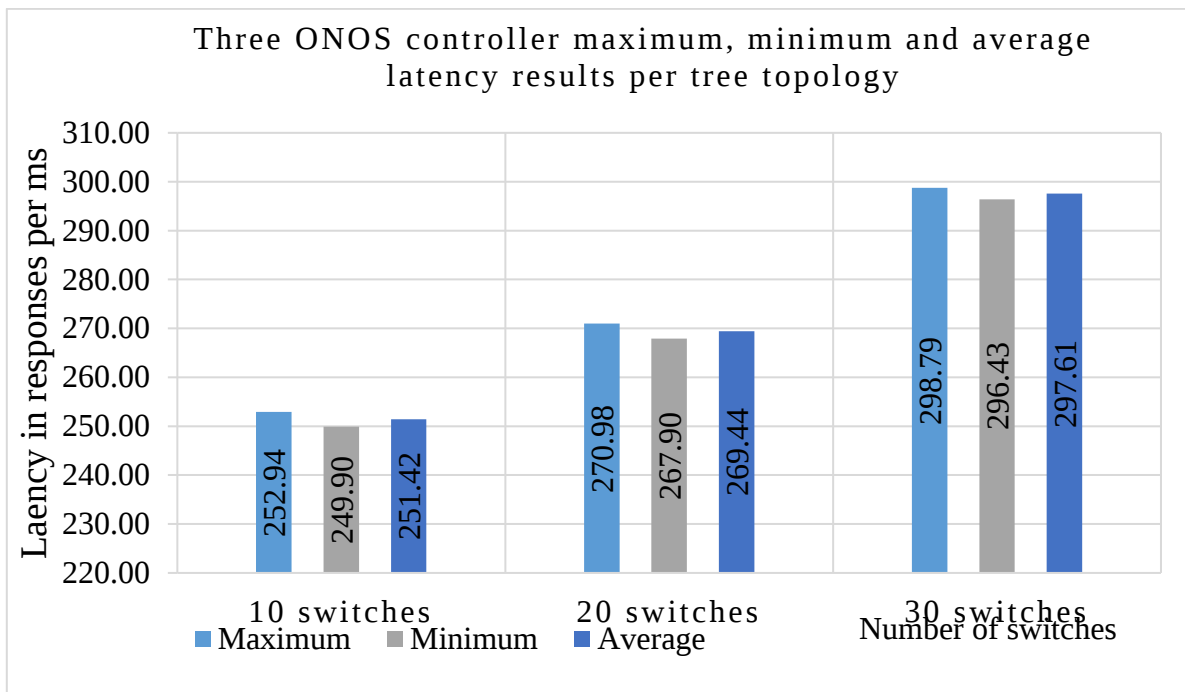


Figure 5-19: Three ONOS controller latency result per tree topology.

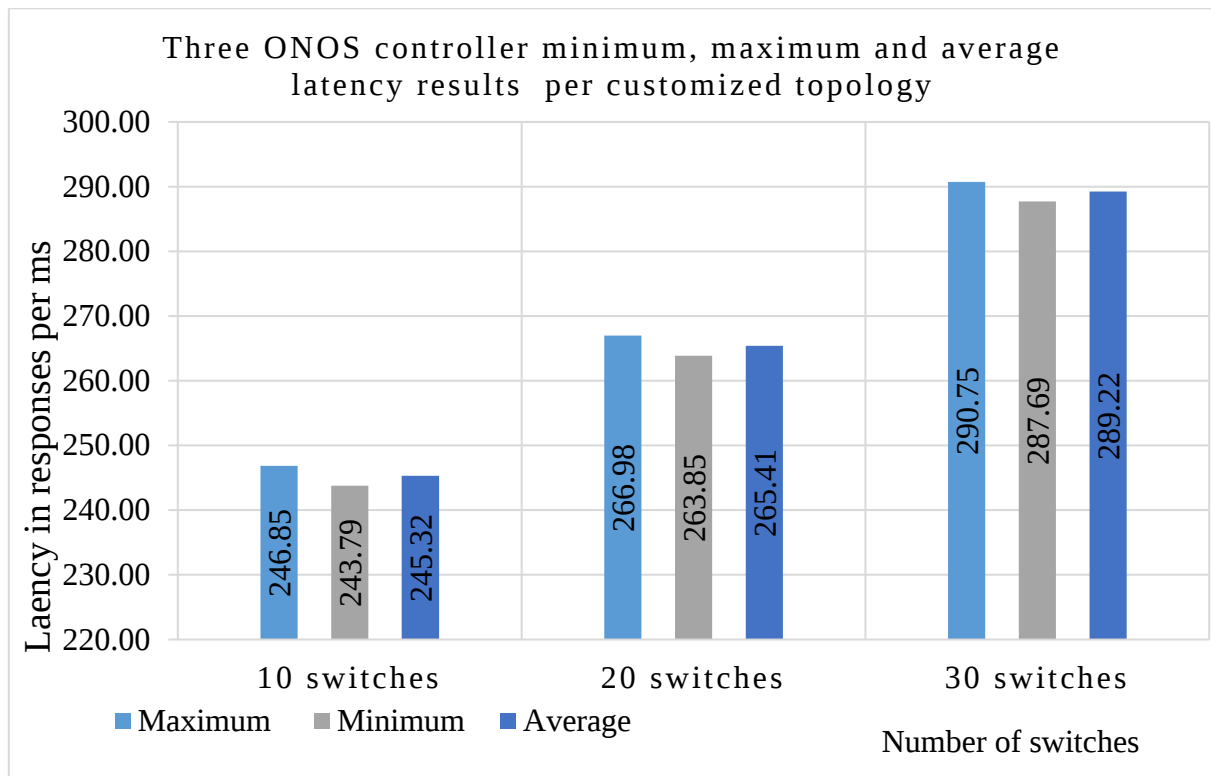


Figure 5-20: Three ONOS controller latency results per customized topology.

The above Figure 5-19 and Figure 5-20 shows the minimum, maximum and average latency results of three ONOS controllers in terms of tree and customized topology respectively. In three ONOS controller under both topologies as we increased the number of switches resulted into slight increase in the number of responses per ms.

As we observed that three ONOS controller provides 124.41 with 10 switches, 132.64 with 20 switches and 147.91 with 30 switches in tree topology and 117.53 with 10 switches, 126.76 with 20 switches and 142.12 with 30 switches in customized topologies.

As we can see in the Figure 5-19, three ONOS controllers per tree topology have higher responses in ms than customized topology. Three ONOS controllers per tree topology with 10, 20 and 30 switches has 6.10, 4.03 and 8.39 more responses per ms than customized topology respectively. So, in this thesis for three ONOS controller tree topology is better topology than customized topology.

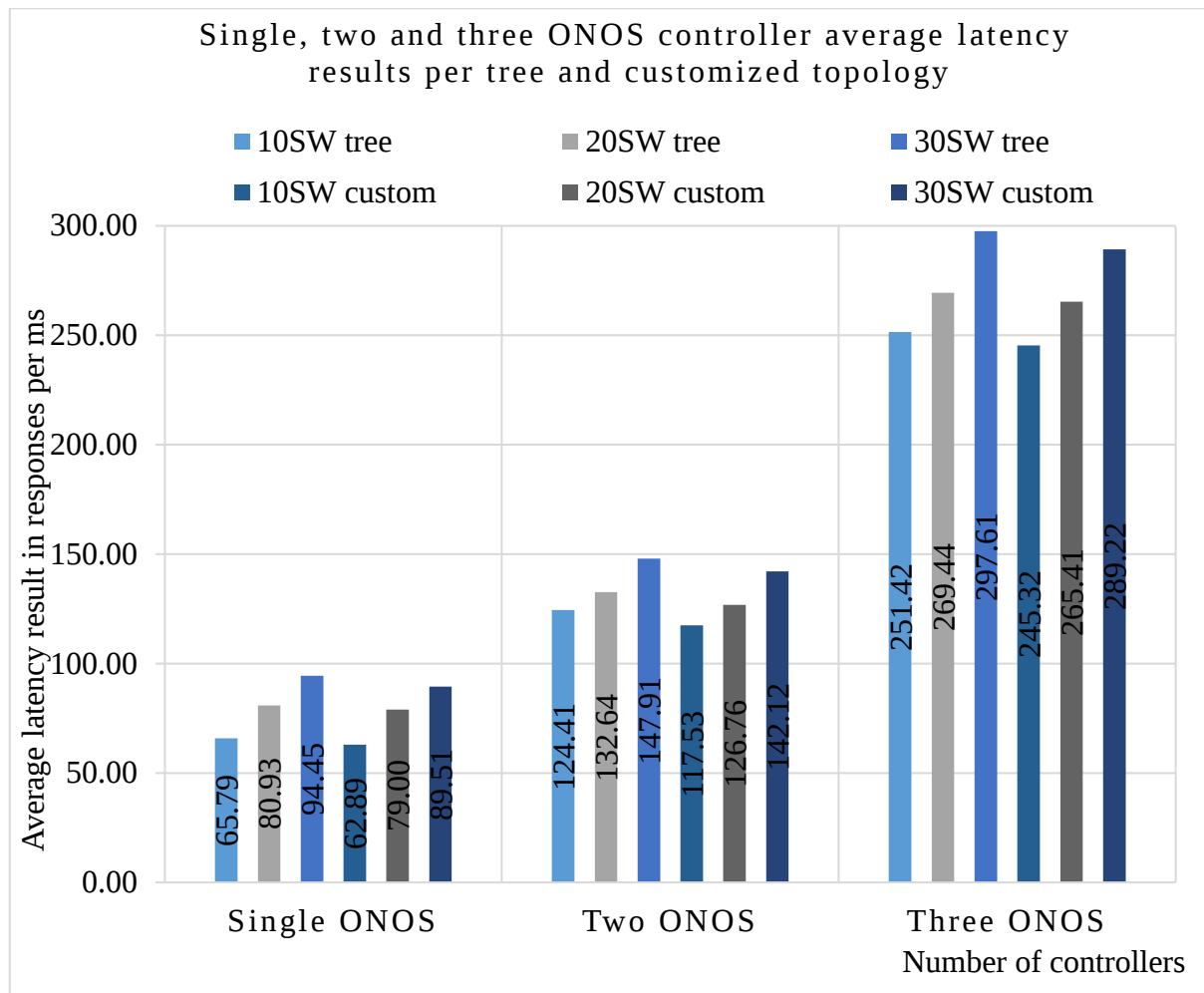


Figure 5-21: Single, two and three ONOS controller average latency results per tree and customized topologies.

As we can see in the Figure 5-21 the single, two and three ONOS controllers the average response per ms results are increase when the number of switches is increased. As we can see in the Figure 5-21 three ONOS controller provides high responses per ms than single and two ONOS controllers in both topologies under a different number of devices.

The single ONOS controller generates from 65.79 to 94.45 average responses per ms in tree topology and 62.89 to 89.51 responses per ms in customized topologies. Two ONOS controller generates from 124.41 to 147.91 average responses per ms in tree topology and 117.53 to 142.12 average responses per ms in customized topologies. Three ONOS controller generates from 251.42 to 297.61 average responses per ms in tree topology and 245.32 to 289.22 average responses per ms in customized topologies.

Finally, three distributed ONOS controller provides high responses per ms than single and two ONOS controllers. The single ONOS controller provides minimum responses per ms and when we add more than 30 devices in the topology it generates less and inconsistence responses. The single and two

ONOS controllers had provided very low responses as compared to three ONOS controllers. So if we used more than one controller, we can increase the number of responses per ms.

5.4. ONOS TCP Bandwidth between Hosts

In this section, we have compared the TCP bandwidth speed of single, two and three ONOS controllers in a cluster. Iperf Mininet tool has been measured and displayed the TCP bandwidth speed of the ONOS controller that provides between hosts in the network. Then we have analyzed and evaluated the TCP bandwidth speed of distributed ONOS controllers in a cluster. The Table 5-7 below shows the bandwidth speed of three ONOS controllers in a cluster using 10, 20 and 30 hosts in both topologies.

```
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20
*** Starting controller
c0 c1
*** Starting 20 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 ...
*** Starting CLI:
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h20
*** Results: ['18.0 Gbits/sec', '18.1 Gbits/se' ]
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h20
*** Results: ['18.4 Gbits/sec', '18.4 Gbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h20
*** Results: ['18.6 Gbits/sec', '18.6 Gbits/sec']
mininet>
```

Figure 5-22: Two ONOS controller TCP bandwidth results between 20 hosts.

The above snapshot Figure 5-22 shows the TCP bandwidth speeds of two ONOS controllers between 20 hosts in GB/s. Iperf Mininet tool runs three times and displays three slight different TCP bandwidth range results. Two ONOS controllers can provide to 18.0 GB/s to 18.6 GB/s bandwidth speed for 20 hosts in the network.

Table 5-7: Distributed ONOS controller TCP bandwidth between hosts in GB/s.

<i>ONOS Controller TCP Bandwidth between Hosts</i>				
No. Controllers	TCP Bandwidth in GB/s	Number of Hosts		
		10 Hosts	20 Hosts	30 Hosts
Single ONOS	Maximum	14.40	14.00	13.60
	Minimum	14.00	13.80	13.40
	Average	14.20	13.90	13.50
Two ONOS	Maximum	18.70	18.60	18.40
	Minimum	18.20	18.00	18.10
	Average	18.45	18.30	18.25
Three ONOS	Maximum	20.60	19.80	19.60
	Minimum	20.00	19.40	19.40
	Average	20.30	19.60	19.50

Table 5-7 shows the maximum, minimum and the average TCP bandwidth results of a single, two and three ONOS controller between 10, 20 and 30 hosts. We have tested 10 times the TCP bandwidth

of single, two and three ONOS controllers connected with 10, 20, 30 number of switches and hosts. Then we have taken the maximum and minimum TCP bandwidth of single, two and three ONOS controllers that provides for each host. Next, to this, we have been calculated the average TCP bandwidth of 10, 20 and 30 hosts. As shown in the table single, two and three ONOS controller has TCP bandwidth difference. The chart below shows the average TCP bandwidth of single, two and three ONOS controller that provides between hosts.

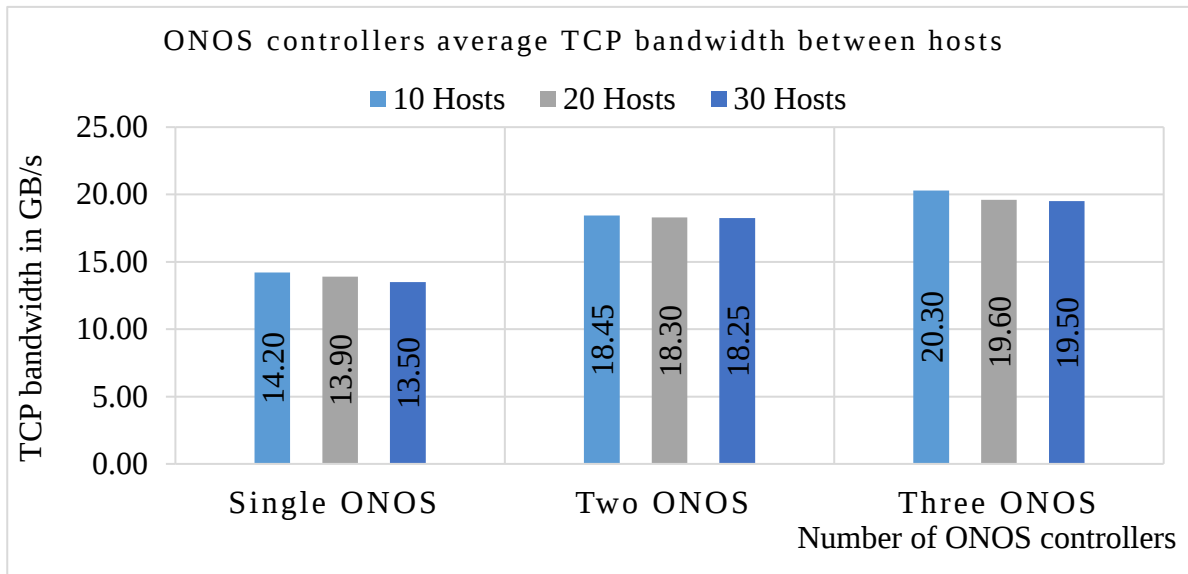


Figure 5-23: ONOS Controller TCP bandwidth between hosts.

As we can see in the Figure 5-23 when the number of ONOS controllers increase the average TCP bandwidth also increase since ONOS controller control each device in a distributed mode. When one ONOS controller highly loaded the devices move to less loaded ONOS controller in a cluster, this enables each device gets high TCP bandwidth. In addition, when the host numbers increase the TCP bandwidth slightly decrease. This can be solved by adding an additional ONOS controller in a cluster. As illustrated in the Figure 5-23 single ONOS controller provides less bandwidth than two ONOS controllers and two ONOS controllers provide less TCP bandwidth than three ONOS controllers in the cluster under the same number of hosts.

5.5. Evaluation

Now a day several researchers set numerical standards for SDN OpenFlow controller throughput and latency. In [47] SDN OpenFlow controller's performance standard in terms of throughput and latency were discussed. Regarding this, the OpenFlow controllers support from 30,000 to 1 million requests per second has high throughput and the OpenFlow controllers that support from 10 to 100 ms delay has a low latency. This offered us a high advantage to analysis and evaluate three ONOS controllers performance in a data center networks. Also today modern data center network needs 1GB to 100 GB network speed, we have used this range network speed as a standard to evaluate the ONOS controller

TCP bandwidth between hosts [70].

In this section, we have evaluated distributed ONOS controller throughput, latency and TCP bandwidth analyzed results. We have considered the performance standards of SDN OpenFlow controllers to evaluate and decide our work results. In addition, we have compared and evaluated the throughput and latency results of this work with other previous work results. Regarding SDN OpenFlow controller standards, the single ONOS controller has a high throughput and low latency. In addition, two ONOS controller has high throughput and low latency. Also, three ONOS controller highly meets the standard and it has high throughput and low latency.

We have also evaluated the TCP bandwidth analyzed results, regarding this single, two and three ONOS controllers meet the modern data center network speed requirements. According to our TCP bandwidth analyzed results, distributed ONOS controller can provide 13.4 to 20.6 gigabyte per second TCP bandwidth between hosts. This meets the network speed requirements of today modern data center networks.

The authors in [52] have been compared and evaluated single floodlight and OpenDaylight controller throughput and latency in a data center networks. In their OpenDaylight controller test, obtained 5500 to 45000 average response per second and 0.005 to 0.45 average responses per ms. In their floodlight controller test, obtained 1000 to 65000 average response per second and 1.2 to 1.45 average responses per ms. And they conclude that floodlight controller has high throughput and low latency than an OpenDaylight controller. 0

According to [52] this result, the single ONOS controller has high throughput and low latency than floodlight and OpenDaylight controllers. Also, two and three ONOS controller generated high amount throughput and low latency than floodlight and OpenDaylight. Due to this, distributed ONOS controller is the best controller than floodlight and OpenDaylight controllers. Therefore, based on distributed ONOS controller performance and TCP bandwidth results, distributed ONOS controller is the best controller in a data center networks than OpenDaylight and floodlight controllers.

Finally, we have evaluated the tree and customized topologies. According to the throughput and latency result of a single, two and three ONOS controllers, tree topology has been the most appropriate topology for single, two and three ONOS controllers.

CHAPTER SIX

6. Conclusion and Future Work

6.1. Conclusion

Software defined networking is a new network architecture that separates the control plane and the data plane. Today SDN network is a hot research area due to this researcher and network organizations highly contributed to this network. In this thesis, we have reviewed literature, compared and selected SDN distributed controllers and emulator tools. We have created three ONOS controllers in a cluster, designed tree, and customized Mininet topologies. In this thesis, performance analysis and evaluation of distributed ONOS OpenFlow controllers by running single, two and three ONOS controller in a cluster has been made. We used throughput and latency key performance parameters to analysis and evaluate ONOS controller performance and used TCP bandwidth to evaluate ONOS controller bandwidth between hosts. Also, analysis of tree and customized topology and analysis of TCP bandwidth between hosts has been performed. We have adapted SDN distributed ONOS controller in a data center networks.

We have presented the throughput, latency, and TCP bandwidth results of single, two and three ONOS controllers. The single ONOS controller has 60665.22 to 82566.85 flow/s throughput, 62.89 to 94.45 responses/ms latency and 13.5 GB/s TCP bandwidth. Two ONOS controller has 105564.30 to 118911.20 flow/s throughput, 117.53 to 147.91 responses/ms latency and 18.25 GB/s TCP bandwidth. Three ONOS controller has 216816.80 to 229388.90 flows/s throughput, 297.61 to 245.32 responses/ms latency and 19.40 GB/s TCP bandwidth. Single, two and three ONOS controllers per tree topology has high throughput and high response per ms than customized topology. Tree topology was better topology than customized topology for single, two and three ONOS controllers.

Distributed ONOS controller performance analysis and evaluation enable to improve the controller performance, identify the best topology and to improve the bandwidth of the controller. Distributed ONOS controller can be a best SDN controller for data center networks. Distributed ONOS controller has high throughput and low latency SDN controller. However, when one controller failed in the cluster the devices controlled by the failed controller start to discover another controller in the cluster this takes a time. In addition, when the number of devices is increasing ONOS graphical user interface not properly show the designed topology and displayed killed devices.

Cbench is SDN OpenFlow controller benchmarking tool. It measures the throughput and latency of OpenFlow controllers. However, in latency mode cbench measured a number of responses/ms but not the amount of time that controller takes to provide a response. In addition, cbench have not the graphical user interface.

6.2. Future Work

OpenFlow controller performance parameters such as packet loss, Jitter, topology discovery time, sojourn time should be tested. Single cluster, like that of the single controller, has a single point failure problem. So, we have recommended ONOS controller performance evaluation using two clusters each with three ONOS controllers per tree and customized topologies. Also, we have recommended improving distributed ONOS controller performance as future work.

Implementing distributed ONOS controller in a real data center networks leads to distributed ONOS controller placement problem. To solve this problem, different researchers have been work distributed controller placement mathematical model, even though the problem still now not solved. Distributed controller placement problem affects ONOS controller performance. This problem needs study and mathematical model design.

To create multiple controllers and to distribute the controllers, ONOS uses clustering method, because ONOS is logically centralized SDN controller. So ONOS should be logically and physically distributed. This requires study and design logically distributed ONOS controller architecture. We have recommended this as a future work.

References

- [1] B. Forouzan, *Data Communication And Network*, 5th ed., Amsterdam: Alan R. Apt, 2007.
- [2] I. Ahokas, "Software-defined networking," *Computer network and communication*, pp. 2-3, 23 September 2014.
- [3] Citrix, "webtorials," Webtorials, 8 January 2016. [Online]. Available: <http://www.webtorials.com/content/2014/01/2013-guide-to-network-virtualization-sdn-3.html>. [Accessed 24 May 2016].
- [4] W. M. L. F. K. Wenjuan Li, "A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures," *Journal of Network and Computer Applications*, pp. 3-4, 10 May 2016.
- [5] F. V. P. C. R. S. S. Diego Kreutz, "Software-Defined Networking: A Comprehensive Survey," *Software-defined networking*, vol. 103, no. 1, pp. 1-61, 31 January 2015.
- [6] N. McKeown, "Stanford," Stanford university, 15 January 2016. [Online]. Available: <http://yuba.stanford.edu/>. [Accessed 23 June 2016].
- [7] ONF, "Listing mambers (Ouer Members)," ONF, Monday November 2016. [Online]. Available: <https://www.opennetworking.org/our-members>. [Accessed 17 November 2016].
- [8] SDX, "sdxcentral," SDX site, 12 Janury 2016. [Online]. Available: <https://www.sdxcentral.com/resources/sdn/who-is-open-networking-foundation-onf/>. [Accessed 25 June 2016].
- [9] H. K. N. Feamster, "Improving Network Management with Software Defined Networking," *IEEE Communications*, pp. 114-115, 23 February 2013.
- [10] D. G. Z. H. T. Q. P. L. Junjie Xie, "Control plane of software defined networks: A survey," *Computer Communication*, pp. 1-10, June 2015.
- [11] M. E. M. L. H. M. Rodrigo S.Couto, "A Reliability Analysis of Datacenter Topologies," *Communication QoS, Reliability and Modelling Symposium*, pp. 1890-1895, 26 September 2014.
- [12] F. L. J. G. H. X. Tao Wang, "Dynamic SDN Controller Assignment in Data Center Networks: Stable Matching with Transfers," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, Hong Kong, 2016.
- [13] Techopedia, "Distributed network," Techopedia site, 26 December 2016. [Online]. Available: <https://www.techopedia.com/definition/27788/distributed-network>. [Accessed 12 Agusut 2016].

- [14] F. H. S. M. T. R. R. K. Advait Dixit, "ElastiCon: Towards An Elastic Distributed SDN Controller," *Software defined network*, pp. 7-12, 20–21 October 2014.
- [15] V. C. L. I. B. L. A. Leonardo, "Evolution and Challenges of Software Defined Networking," *Group of Analysis, Security and Systems*, pp. 1-6, 2012.
- [16] ON.LAB, "Introducing ONOS an SDN network operating system for Service Providers," ON.Lab, England, 2015.
- [17] J. S. N. S. G. Karamjeet Kaur, "Mininet as Software Defined Networking," in *International Conference on Communication*, Ferozepur, 2014.
- [18] V. A. M. K. Shiva Rowshanrad, "Performance Evaluation of SDN Controllers: Floodlight and OpenDaylight," *IIUM Engineering*, vol. 17, no. 2, pp. 47-56, 30 November 2016.
- [19] VirtualBox.org, "VirtualBox," Oracle, 09 September 2016. [Online]. Available: <https://www.virtualbox.org/>. [Accessed 19 October 2016].
- [20] CGManager, "Linux Containers," Canonical Ltd, 10 November 2016. [Online]. Available: <https://linuxcontainers.org/>. [Accessed 22 November 2016].
- [21] P. Module, Python Module of the Week, London: Creative commons, September 10, 2016.
- [22] Brocade, "Software Defined Networking (SDN)," *Brocade Communications Systems*, pp. 1-8, 18 August 2015.
- [23] K. H. Matthew, *Where Wizards Stay Up Late the Origins of The Internet*, New York: Simon & Schuster, 1998.
- [24] F. P. E. S. S. D. Kreutz, "Software-defined networking: A comprehensive survey," *proceedings of the IEEE*, vol. 103, no. 1, p. 14–76, 2015.
- [25] S. Shenker, "The future of networking, and the past of protocols," *The Open Networking Summit*, pp. 1-5, 10 December 2011.
- [26] T. S. A. S. R. A. Y. L. Manar Jammal, "Software-defined networking: State of the art and research challenges," *Computer Networks*, pp. 75- 79, 25 July 2014.
- [27] H. K. T. B. A. A. N. Feamster, "The Evolution of Network Configuration: A Tale of Two Campuses," in *ACM SIGCOMM Internet Measurement conference (IMC)*, pp. 499-514, 2-4 November 2011.
- [28] N. McKeown, "OpenFlow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, p. 69–74, March 2008.
- [29] N. H. Feamster, "Detecting BGP Configuration Faults with Static Analysis," *computer network*, pp. 3-9, 26 May 2010.

- [30] W. D. R. Mahajan, "Understanding BGP misconfiguration," *Border gateway protocols*, p. 3–17, 19-23 August 2002.
- [31] O. ONF, "Software-defined networking: The new norm for networks.," *ONF White Paper*, pp. 2-8, 13 April 2012.
- [32] G. Z. Kyriakos, "Open flow Virtual Networking: A Flow-Based Network Virtualization Architecture," *Royal Institute of Technology (KTH)*, pp. 5-7, 12 December 2009.
- [33] J. R. E. Z. N. Feamster, "The Road to SDN: An intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 1-11, 14 April 2014.
- [34] M. M. X. N. N. K. O. T. Bruno Nunes Auto, "A Survey of Software-Defined Networking: The Past, Present, and Future of Programmable Networks," *Communications Surveys and Tutorials*, vol. 16, no. 3, pp. 1617-1634, 19 January 2014.
- [35] ONF, "Open Networking Foundation," ONF, 2 May 2016. [Online]. Available: <https://www.opennetworking.org/>. [Accessed 14 May 2016].
- [36] Y. W. C. H. F. Wenfeng Xia, "A Survey on Software-Defined Networking," *IEEE Communication Surveys And Tutorials*, vol. 17, no. 1, pp. 1-25, 24 September 2015.
- [37] A. G. Rahim Masoudi, "Software defined networks: A survey," *Journal of Network and Computer Application*, pp. 1-25, 26 March 2016.
- [38] Sdncentral, "Traditonal network," sdncentral, 28 April 2016. [Online]. Available: <https://www.sdncentral.com>. [Accessed 12 May 2016].
- [39] T. Slattery, "nojitter," Jitter, 24 March 2016. [Online]. Available: <http://www.nojitter.com>. [Accessed 14 May 2016].
- [40] F. P. D. Kreutz, "Towards Secure and Dependable Software Defined Networks," *Proceedings of the second workshop on Hot Topics in Software Defined Networking*, pp. 1-6, 2013.
- [41] SDNCentral, "SDN Controllers Report," SDNCentral LLC, Chicago, 2015.
- [42] M. B. M. R. B. F. Benamrane, "Performances of OpenFlow-Based Software Defined Networks: An overview," *Journal Of Networks*, vol. 10, no. 6, pp. 2-4, 24 JUNE 2015.
- [43] T. J. B. M. N. S. N. Gude, "NOX: Towards an Operating System for Networks," *Computer Communication Review*, vol. 38, p. 106–109, July 2008.
- [44] Nox, "About POX," noxrepo, 23 September 2016. [Online]. Available: <http://www.noxrepo.org/pox/about-pox/>. [Accessed 03 October 2016].
- [45] R. S. F. Community, "An Operating System for Software Defined Network.," 22 January

2016. [Online]. Available: <http://osrg.github.io/ryu/>.. [Accessed 22 October 2016].
- [46] Github, "floodlight," loxigen, 23 May 2016. [Online]. Available: <https://github.com/floodlight/>. [Accessed 25 June 2016].
- [47] M. G. J. H. Y. H. Pankaj Berde, "ONOS: Towards an Open, Distributed SDN OS," *Computer-Communication Networks*, pp. 1-6, 22 August 2014.
- [48] MD-SAL, "OpenDaylight Controller," MD-SAL corporation, 19 April 2016. [Online]. Available: https://wiki.opendaylight.org/view/OpenDaylight_Controller. [Accessed 14 May 2016].
- [49] W. is.com, "Distributed Control Plane Architecture (DCPA)," Techtarget, 18 09 2016. [Online]. Available: <http://searchsdn.techtarget.com/definition/Distributed-Control-Plane-Architecture-DCPA>. [Accessed 22 09 2016].
- [50] H. C. Z. W. S. C. Guozhen Cheng, "DHA: Distributed Decisions on the Switch Migration Toward a Scalable SDN Control Plane," *National Digital Switching System Engineering & Technological R&D Center*, pp. 1-9, 03 December 2015.
- [51] D. Building, "Enterprise Networking vs Data Center Networking," YourDailyTech, 27 December 2016 . [Online]. Available: <https://yourdailytech.com/networking/enterprise-networking-vs-data-center-networking/>. [Accessed 27 12 2016].
- [52] M. A. A. I. Zuhra Khan Khattak, "Performance Evaluation of OpenDaylight SDN Controller," *Software Defined Networking Benchmarking*, pp. 1-6, 23 December 2014.
- [53] S. Y. M. R. A. Tootoonchian, "On controller performance in software-defined networks," *Cloud, and Enterprise Networks and Services (Hot-ICE)*, pp. 1-5, 2012.
- [54] A. J. Wang, "Scalable software defined network controllers," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, 2012.
- [55] A. Z. R. M. K. B. Rahamatullah Khondoker, "Feature-based Comparison and Selection of Software Defined Networking (SDN) Controllers," *Controller Comparison*, vol. 10, no. 03, pp. 1-7, 2014.
- [56] F. F. P. C. G. S. V. L. Zuccaro, "Distributed control in virtualized networks," *The 10th International Conference on Future Networks and Communications (FNC 2015)*, p. 276 –283, 2015.
- [57] K. Y. J. Z. W. L. K. L. Bing Xiong, "Performance evaluation of OpenFlow-based software-defined networks based on queueing model," *Computer Networks*, pp. 172-184, 4 April 2016.
- [58] D. G. Z. H. T. Q. P. L. Junjie Xie, "Control plane of software defined networks: A survey,"

Computer Communications, pp. 1-10, 18 June 2015.

- [59] M. N. J. L. M. R. Y. H. T. T. Koponen, "Onix: A distributed control platform for large-scale production networks.," *International*, vol. 10, p. 1–6, 2010.
- [60] A. Y. Ganjali, "Hyperflow: a distributed control plane for OpenFlow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking (INM)*, 2010.
- [61] L. Foundation, "OpenDaylight Home page," Linux foundation, 12 November 2016. [Online]. Available: <https://www.opendaylight.org/home>. [Accessed 24 November 2016].
- [62] M. B. J. L. Kevin Phemius, "DISCO: Distributed SDN Controllers in a Multi-domain Environment," *Thales Communications & Security*, pp. 1-8, 23 March 2014.
- [63] O. Group, "Cluster Coordination," wiki, 10 November 2016. [Online]. Available: <https://wiki.onosproject.org/display/ONOS/Cluster+Coordination>. [Accessed 15 November 2016].
- [64] A. B. P. G. G. M. Abubakar Siddique Muqaddas, "Inter-controller Traffic in ONOS Clusters for SDN Networks," *Next-Generation Networking and Internet Symposium*, vol. 9, no. 05, pp. 1-6, 2016.
- [65] B. Lantz, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pp. 18-20, 2010.
- [66] S. Y. Wang, "Comparison of SDN OpenFlow Network Simulator and Emulators: EstiNet vs. Mininet," *The nineteenth IEEE Symposium on Computers and Communications*, pp. 1-6, 23-26 June 2014.
- [67] A. B. A. A. S. R. B. P. Pupatwibul, "A Comparative Review: Accurate OpenFlow Simulation Tools for Prototyping," *Journal of Networks*, vol. 10, no. 5, pp. 1-6, 20 MAY 2015.
- [68] F. K. S. Askar, "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments," *Emulation*, vol. 20, no. 03, pp. 1-6, 21 November 2016.
- [69] E. B. Y. Mohammed Basheer Al-Somaidai, "Survey of Software Components to Emulate OpenFlow Protocol as an SDN Implementation.," *Emulate OpenFlow Protocol*, vol. 3, no. 6, pp. 1-6, 10 11 2016.
- [70] Farmingiham, "Datacenter Networks," IDC, 05 January 2017. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=IDC_P13. [Accessed 05 January 2017].
- [71] P. Z. L. X. X. L. C. H. Huanzhao Wang, "A secure and high-performance multi-controller architecture for software-defined networking," *Security, Multi-controller, Distributed rule*

store, pp. 634-646, 8 August 2016.

- [72] I. Merriam-Webster, "Synonym Discussion of standard," Merriam-Webster, 06 January 2017. [Online]. Available: <https://www.merriam-webster.com/dictionary/standard>. [Accessed 07 January 2017].
- [73] A. K. Simon Hunt, "The ONOS GUI," onosproject, 30 December 2016. [Online]. Available: [https://wiki.onosproject.org/display/ONOS/The ONOS web GUI.html](https://wiki.onosproject.org/display/ONOS/The+ONOS+web+GUI.html). [Accessed 21 January 2017].
- [74] D. Erickson, "What is Beacon? Home," OpenFlow, 4 August 2016. [Online]. Available: <https://openflow.stanford.edu/display/Beacon/Home>. [Accessed 9 December 2016].
- [75] E. Ng, "Maestro: A System for Scalable OpenFlow Control TSEN Maestro Technical report.," *sdn controller*, pp. 1-6, 2010.

Appendices

Appendix A

Most frequently used commands

ONOS Commands

In this section, we have listed commands that were frequently used in the thesis progress.

```
distributed@mininet-vm:~$ sudo lxc-create -n onos1 -t Ubuntu # to create onos1 LXC container.
```

```
distributed@mininet-vm:~$ sudo lxc-ls-fancy # to verify the created containers.
```

```
distributed@mininet-vm:~$ sudo lxc-start -n onos1 -d # to start LXC container onos1.
```

```
distributed@mininet-vm:~$ export ONOS_ROOT=~/.onos # to develop ONOS environment setup
```

```
distributed@mininet-vm:~$ cd /home/Mininet/onos #
```

```
distributed@mininet-vm:~$ source $ONOS_ROOT/tools/dev/bash_profile
```

```
distributed@mininet-vm:/home/Mininet/onos$ mvn clean install # to build and package ONOS
```

```
distributed@mininet-vm:~$ cells # to install and start ONOS on 3 LXC containers.
```

```
distributed@mininet-vm:~$onos-service $OC1 stop # to stop the running ONOS one
```

```
distributed@mininet-vm:~$onos-uninstall $OC1 # to uninstall ONOS one
```

```
distributed@mininet-vm:~$ onos-package # to package ONOS
```

```
-rw-rw-r--1 distributed distributed 82M Jan 20 11:44 /tmp/onos-1.2.1.distributed.tar.gz
```

```
distributed@mininet-vm:~$ onos-install -f $OC1 #this has now installed ONOS on LXC container.
```

```
Onos start/running, process 734
```

```
distributed@mininet-vm:~$ onos-install -f $OC2 #this has now installed ONOS on LXC container.
```

```
Onos start/running, process 717
```

```
distributed@mininet-vm:~$ onos-install -f $OC3 #this has now installed ONOS on LXC container.
```

```
Onos start/running, process 721
```

```
distributed@mininet-vm:~$ onos -w $OC1 #start and manage multiple ONOS by connecting
```

```
Onos> # this indicates we are in the ONOS command line. In the ONOS command line interface, there are 455 commands.
```

```
Onos> app activate org.onosproject.fwd # Reactive forwarding application using flow subsystem
```

```
onos> app activate org.onosproject.openflow # OpenFlow protocol southbound providers
```

```
onos> app activate org.onosproject.drivers # Builtin device drivers
```

```
onos> app activate org.onosproject.mobility # Host mobility application
```

```
onos> apps -a -s # to display and view activated ONOS apps
```

```
onos> nodes #to display all installed ONOS in a cluster.
```

```
Id=10.0.3.11, address=10.0.3.11:9876, state=ACTIVE, updated=8m ago *
```

```
id=10.0.3.12, address=10.0.3.12:9876, state=ACTIVE, updated=6m ago
```

```

id=10.0.3.13, address=10.0.3.13:9876, state=ACTIVE, updated=6m ago
onos>masters # to view all switches connected with its master ONOS controller in a cluster.
onos> masters
10.0.3.11: 1 devices  of:00000000000000010
10.0.3.12: 1 devices  of:0000000000000000e
10.0.3.13: 1 devices  of:00000000000000014
onos> balance-masters # to balance the number of switches in each ONOS.
onos> devices # to view all OpenFlow switches run in Mininet that is controlled by ONOS controller.
id=of:0000000000000000e, available=true, role=STANDBY, type=SWITCH, mfr=Nicira, Inc.,
hw=Open vSwitch, sw=2.1.3, serial=None, protocol=OF_10, channelId=10.0.3.1:55171
id=of:00000000000000010, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc.,
hw=Open vSwitch, sw=2.1.3, serial=None, protocol=OF_10, channelId=10.0.3.1:55165
onos> hosts # to view all hosts run in Mininet that is controlled by ONOS controller.
id=BA:83:89:D1:F7:11/-1, mac=BA:83:89:D1:F7:11, location=of:0000000000000003/1, vlan=-1,
ip(s)=[10.0.0.1]
id=DA:20:04:0B:9F:97/-1, mac=DA:20:04:0B:9F:97, location=of:0000000000000003/2, vlan=-1,
ip(s)=[10.0.0.2]

```

Mininet Commands

```

mininet> # indicates that you are in Mininet command line interface.
mininet> pingall #to testing ping reachability between all hosts.
mininet> links # to view all links
mininet> net #display all node interfaces
mininet> nodes #to view total number of controllers, switches and hosts.
mininet> iperfudp # to test UDP bandwidth between h1 and h4.
mininet> xterm h1 h2 #to open host1 host2 ... hn and switches in root mode.

```

Appendix B

Simple tree topology sample python source code with three switches, six hosts and remote controllers.

```

from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.net import Mininet
from mininet.node import RemoteController, OVSSwitch
from mininet.topo import Topo
class Tree( Topo ):

```

"Tree topology with 7 switch and 8 hosts"

```
def build( self ):
```

```
    # Create 8 hosts.
```

```
    h1 = self.addHost( 'h1' )
```

```
    h2 = self.addHost( 'h2' )
```

```
    h3 = self.addHost( 'h3' )
```

```
    h4 = self.addHost( 'h4' )
```

```
    h5 = self.addHost( 'h5' )
```

```
    h6 = self.addHost( 'h6' )
```

```
    h7 = self.addHost( 'h7' )
```

```
    h8 = self.addHost( 'h8' )
```

```
    # Create 7 switches
```

```
    s1 = self.addSwitch( 's1' )
```

```
    s2 = self.addSwitch( 's2' )
```

```
    s3 = self.addSwitch( 's3' )
```

```
    s4 = self.addSwitch( 's4' )
```

```
    s5 = self.addSwitch( 's5' )
```

```
    s6 = self.addSwitch( 's6' )
```

```
    s7 = self.addSwitch( 's7' )
```

```
# Add links between the 7 switch and 8 host
```

```
    self.addLink( s4, h1 )
```

```
    self.addLink( s4, h2 )
```

```
    self.addLink( s5, h3 )
```

```
    self.addLink( s5, h4 )
```

```
    self.addLink( s6, h5 )
```

```
    self.addLink( s6, h6 )
```

```
    self.addLink( s7, h7 )
```

```
    self.addLink( s7, h8 )
```

```
    self.addLink( s1, s2 )
```

```
    self.addLink( s1, s3 )
```

```
    self.addLink( s2, s4 )
```

```
    self.addLink( s2, s5 )
```

```
    self.addLink( s3, s6 )
```

```
self.addLink( s3, s7 )
```

```
# Allows the file to be imported using `mn --custom tree7.py --topo tree
```

```
topos = { 'tree': Tree }
```

Sample python source code of customized topology with four switches, for hosts and three remote controllers.

```
#Simple customized topology
```

```
from mininet.topo import Topo
```

```
from mininet.cli import CLI
```

```
from mininet.net import Mininet
```

```
from mininet.node import RemoteController, OVSKernelSwitch
```

```
class Custom( Topo ):
```

```
    def __init__( self ):
```

```
        "Create customized topology"
```

```
        # Initialize custom topology
```

```
        Topo.__init__( self )
```

```
        # Add 4 hosts and 4 switches
```

```
        host1 = self.addHost( 'h1' )
```

```
        host2 = self.addHost( 'h2' )
```

```
        host3 = self.addHost( 'h3' )
```

```
        host4 = self.addHost( 'h4' )
```

```
        sw1 = self.addSwitch('s1')
```

```
        sw2 = self.addSwitch('s2')
```

```
        sw3 = self.addSwitch('s3')
```

```
        sw4 = self.addSwitch('s4')
```

```
        # Add links between 4 switches and 4 hosts
```

```
        self.addLink( sw1, host1 )
```

```
        self.addLink( sw2, host2 )
```

```
        self.addLink( sw3, host3 )
```

```
        self.addLink( sw4, host4 )
```

```
        self.addLink( sw1, sw2 )
```

```
        self.addLink( sw2, sw3 )
```

```
        self.addLink( sw3, sw4 )
```

```
        self.addLink( sw4, sw2 )
```

```
        self.addLink( sw1, sw4 )
```

```
        self.addLink( sw3, sw1 )
```

```
# Allows the file to be imported using `mn --custom Custom.py --topo customized
```

```
topos = { 'customized': Custom }
```

Network Topology Code

```
import os
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.net import Mininet
from mininet.node import RemoteController, OVSSwitch
from mininet.topo import Topo
from mininet.util import irange

class DatacenterCustomtopo( Topo ):
    "Design Datacenter Network Topology"

    def build( self, DatacentersCoreSWs=5, numHosts=6, numDatacenterSWs =5 ):
        # We assign the number of datacenter switches less than the number of core switches
        if numDatacenterSWs >= 30:
            raise Exception( "Please use less than the number of datacenter switches" )

        self.blocks = []
        coreSwitches = []
        last datacenterSwitch = None

        # Create and link all the core switches
        for i in irange( 1, numDatacenterSWs ):

            datacenterSwitch = self.addSwitch( 's%s' % i )
            coreSwitches.append( coreSwitch )

            # initialized at least two CoreSwitches.
            if lastDatacenterSwitch:
                self.addLink( lastDatacenterSwitch, datacenterSwitch )

            lastDatacenterSwitch = coreSwitch

        # Make the final link from the datacenterSwitch to the first switch
        if numDatacenterSWs > 1:
            self.addLink( lastDatacenterSwitch, coreSwitches[0] )
            for i in irange( 1, DatacentersCoreSW ):
                rack = self.buildBlocks( i, numHosts=numHosts,
                                         numDatacenterSWs= numDatacenterSWs )
                self.blocks.append( block )

        # For every numDatacenterSWs, add a link between the datacenter switch and
        #core datacenter switch of the same ID
        for j in range(numDatacenterSWs):
```

```

        self.addLink(coreSwitches[j], block[j] )

def buildBlock( self, loc, numHosts, numDatacenterSWs ):
    "Build a block of hosts with a top block switch"

    switches = []
    for n in xrange( 1, numDatacenterSWs):
# Make sure each switch gets a unique DPID based on the location
# in the block for easy decoding when looking at logs.
        dpid = ( loc * 16 ) + y
        switch = self.addSwitch( 's%s1%s' % (y, loc), dpid='%x' % dpid )
        switches.append( switch )

    for n in xrange( 1, numHosts ):
        host = self.addHost( 'h%s1%s' % ( y, loc ) )

        # Add a link from every datacenter switch to the host
        for switch in switches:
            self.addLink( switch, host )

    # Return list of datacenter switches for this blocks return switches
# Allows the file to be imported using `mn --custom <filename> --topo dafull`
topos = {
    'custom': DatacenterCustomtopo
}

```

Tree Topology Code

```

import os
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.net import Mininet
from mininet.node import RemoteController, OVSSwitch
from mininet.topo import Topo
from mininet.util import xrange

class DtacenterTreeTopo(Topo):

    numP = 8    # Number of ports.
    numL = 4    # Number of switch layers.
    numcore = 4 # Number of core switches.
    numedge = 8 # Number of switches per layer.
    numhost = 30 # Number of hosts.
    hnodes = [] # host list.
    snodes = [] # switch list.
    def __init__(self, numP, numL):
        self.numP=numP
        self.numL = numL
        # Maximum number of switches and hosts.

```

```

# Core switch and Edge switch have the same port number.
self.numhost = 2 * (self.numP / 2) * self.numL
self.numedge = self.numhost / (self.numP / 2)
self.numcore = self.numedge / 2
Topo.__init__(self)
# Create datacenter tree topology.
self.addHosts()
self.addSwitches()
self.addLinks()

def addHosts(self):
    for i in range(0, self.numhost):
        self.hnodes.append(self.addHost("h" + str(i)))
def addSwitches(self):
    # Add core switches.
    core = []
    for i in range(0, self.numcore):
        core.append(self.addSwitch("c" + str(i)))
    self.snodes.append(core)
    # Add edge switches.
    for i in range(1, self.numL):
        edge = []
        for j in range(0, self.numedge):
            edge.append(self.addSwitch("e" + str(i) + str(j)))
        self.snodes.append(edge)
def addLinks(self):
    # Add links between core and edge switches.
    for i in range(0, self.numedge):
        for j in range(0, self.numP / 2):
            idx = i * (self.numP / 2) % self.numcore + j
            self.addLink(self.snodes[1][i], self.snodes[0][idx])
# Add links between each layer of edge switches.
    for i in range(1, self.numL - 1):
        for j in range(0, self.numedge):
            for k in range(0, self.numP / 2):
                tmp = self.numP / 2
                self.addLink(self.snodes[i][j],
                             self.snodes[i + 1][(j // tmp) * tmp + k])

# Add links between edge switches and hosts.
    for i in range(0, self.numedge):
        for j in range(0, self.numP / 2):
            self.addLink(self.snodes[self.numL - 1][i],
                         self.hnodes[self.numP / 2 * i + j])

topos = { 'tree' :
(lambda:DatacenterTreeTopo(6, 4))

```

Appendix C

ONOS Throughput sample result with 10 switches

```
distributed@mininet-vm:~$ taskset -c 0-1 cbench -c 10.0.3.11 -p 6633 -m 1000 -l 14 -w 2 -M 100000 -i 50 -I 10 -s 10 -t
cbench: controller benchmarking tool
  running in mode 'throughput'
  connecting to controller at 10.0.3.11:6633
  faking 10 switches offset 1 :: 14 tests each; 1000 ms per test
  with 100000 unique source MACs per switch
  learning destination mac addresses before the test
  starting test with 0 ms delay after features_reply
  ignoring first 2 "warmup" and last 0 "cooldown" loops
  connection delay of 50ms per 10 switch(es)
  debugging info is off
06:13:48.458 10 switches: flows/sec: 4361 4363 4313 4263 4212 4162 4112 4061 4012 3962 total = 41.819913 per ms
06:13:49.559 10 switches: flows/sec: 5581 5580 5580 5579 5579 5578 5578 5577 5577 5577 total = 55.767151 per ms
06:13:50.664 10 switches: flows/sec: 5680 5679 5679 5679 5679 5678 5678 5678 5678 5678 total = 56.785489 per ms
06:13:51.765 10 switches: flows/sec: 6232 6232 6231 6231 6231 6231 6230 6230 6229 6229 total = 62.305003 per ms
06:13:52.867 10 switches: flows/sec: 6341 6341 6340 6340 6340 6340 6340 6340 6339 6338 total = 63.396337 per ms
06:13:53.968 10 switches: flows/sec: 6422 6422 6421 6421 6421 6420 6419 6419 6418 6418 total = 64.200165 per ms
06:13:55.069 10 switches: flows/sec: 6448 6448 6448 6447 6447 6446 6446 6445 6444 6444 total = 64.461904 per ms
06:13:56.170 10 switches: flows/sec: 6347 6347 6346 6346 6345 6345 6343 6343 6343 6342 total = 63.446683 per ms
06:13:57.271 10 switches: flows/sec: 6424 6423 6423 6422 6422 6422 6421 6421 6420 6419 total = 64.215294 per ms
06:13:58.373 10 switches: flows/sec: 6452 6451 6450 6450 6449 6449 6506 6448 6448 6447 total = 64.549935 per ms
06:13:59.474 10 switches: flows/sec: 6442 6441 6441 6439 6439 6438 6438 6438 6437 6437 total = 64.389807 per ms
06:14:00.575 10 switches: flows/sec: 6442 6440 6440 6439 6439 6439 6438 6438 6437 6437 total = 64.387970 per ms
06:14:01.676 10 switches: flows/sec: 6452 6452 6451 6450 6449 6449 6448 6448 6448 6447 total = 64.493420 per ms
06:14:02.777 10 switches: flows/sec: 6433 6432 6432 6430 6430 6430 6429 6429 6428 6428 total = 64.300550 per ms
RESULT: 10 switches 12 tests min/max/avg/stdev = 56785.49/64549.94/63411.05/2095.01 responses/s
```

Figure1 shows the throughput result in flows per second.

```
distributed@mininet-vm:/home/mininet/oflops/cbench$ taskset -c 0-1 cbench -c 10.0.3.11 -p 6633 -m 1000 -l 16 -M 100000 -i 50 -s 10
cbench: controller benchmarking tool
  running in mode 'latency'
  connecting to controller at 10.0.3.11:6633
  faking 10 switches offset 1 :: 16 tests each; 1000 ms per test
  with 100000 unique source MACs per switch
  learning destination mac addresses before the test
  starting test with 0 ms delay after features_reply
  ignoring first 1 "warmup" and last 0 "cooldown" loops
  connection delay of 50ms per 1 switch(es)
  debugging info is off
21:47:44.436 10 switches: flows/sec: 4661 4656 4655 4651 4649 4647 4645 4643 4640 4636 total = 64.218852 per ms
21:47:45.539 10 switches: flows/sec: 4843 4843 4842 4842 4842 4842 4842 4842 4841 4841 total = 67.458909 per ms
21:47:46.643 10 switches: flows/sec: 4649 4647 4647 4646 4646 4646 4646 4646 4643 total = 64.308588 per ms
21:47:47.746 10 switches: flows/sec: 4832 4832 4832 4832 4832 4831 4831 4831 4831 total = 67.292227 per ms
21:47:48.869 10 switches: flows/sec: 4784 4784 4784 4784 4784 4784 4784 4784 4784 total = 66.537923 per ms
21:47:49.971 10 switches: flows/sec: 4705 4703 4701 4698 4694 4690 4688 4685 4682 4681 total = 64.945651 per ms
21:47:50.072 10 switches: flows/sec: 4813 4810 4805 4803 4802 4799 4797 4794 4791 4790 total = 66.659157 per ms
21:47:51.173 10 switches: flows/sec: 4817 4813 4808 4805 4801 4797 4796 4796 4791 4790 total = 66.659463 per ms
21:47:52.275 10 switches: flows/sec: 4676 4674 4669 4667 4666 4662 4660 4657 4654 4651 total = 64.465777 per ms
21:47:53.379 10 switches: flows/sec: 4880 4878 4875 4872 4870 4867 4865 4863 4860 4857 total = 67.762378 per ms
21:47:54.480 10 switches: flows/sec: 4795 4795 4794 4793 4793 4793 4793 4793 4793 4793 total = 76.683003 per ms
21:47:55.436 10 switches: flows/sec: 4868 4868 4868 4867 4867 4867 4866 4866 4865 4865 total = 67.850144 per ms
21:47:56.586 10 switches: flows/sec: 4780 4781 4781 4781 4780 4779 4779 4779 4779 4779 total = 66.463082 per ms
21:47:57.688 10 switches: flows/sec: 4844 4844 4844 4844 4844 4844 4843 4842 4842 4842 total = 67.479845 per ms
21:47:58.766 10 switches: flows/sec: 4847 4847 4847 4847 4848 4848 4848 4848 4847 4847 total = 67.552845 per ms
21:47:59.905 10 switches: flows/sec: 4788 4788 4788 4787 4787 4787 4786 4786 4786 4786 total = 66.574311 per ms
RESULT: 10 switches 16 tests min/max/avg/stdev = 64218.85/67850.14/66432.01/1214.77 responses/s
```

Figure 2 ONOS Latency sample result with 10 switches.