Sudan University of Science and Technology

College of Graduate Studies

School of Electronics Engineering

Performance Evaluation of Software Defined Networking compare to Traditional Networks

تقويم اداء الشبكات المعرفة برمجيا بالمقارنة مع الشبكات التقليدية

A research submitted in partial fulfillment for the requirements of the M.Sc. degree in Computer and Network Engineering.

By:

Mohammed Khalil Abdalla Elmedani.

Supervisor:

Dr. Rashid A. Saeed.

April, 2017

بسم الله الرحمن الرحيم

قال تعالى:

وَقُلِ ٱلۡحَمۡدُ لِلَّهِ ٱلَّذِي لَمۡ يَتَّخِذۡ وَلَدٗا وَلَمۡ يَكُن لَّهُۥ شَرِيكٞ فِي ٱلۡمُلۡكِ وَلَمۡ يَكُن لَّهُۥ وَلِيّٞ مِّنَ ٱلذُّلِّۖ وَكَبِّرۡهُ تَكۡبِيرَۢا ﴿١١١﴾

صدق الله العظيم

سورة الإسراء

الآية (111)

DEDICATION

To My parents, teachers, colleagues,

AND OF COURSE, TO OUR

BELOVED COUNTRY.

Acknowledgements

I wish to express our appreciation and gratitude to **Dr. Rashid A. Saeed.**who, through his ideas, suggestions and advice improved this project. Thanks to him, not only for his help in general but also for his trust and guidance during the revision process.

My deepest thanks to all the staff in electronic department at Sudan University of Science and Technology, who, in many ways contributed in making this project a memorable and an enriching experience.

Finally, I thank my families for their patience and understanding during the days of writing and revising this project.

ABSTRACT

Software-defined network continues to be one of the most hyped technology evolutions in information and communication technology compare to all traditional and perfuse network technologies.

These traditional networks introduce many challengestime-consuming, Multi-vendor environments require a high level of expertise and complicate network segmentation, inconvenience and difficulty of learning to manage such a huge systems and devices and more.

In this study, mininet software is emulated using many different scenarios in order to evaluate the connectivity and performance of SDN networks compare to traditional networks.

Consider the difficulty of SDN network as new technology the performance of these scenarios is evaluated by using iperf tool to investigate that the SDN networks can meet the basic function of traditional networks.

The requirements for the functionalities of the current network are not complex, only basic switching and routing are required. These were simulated with the different topologies.

المستخلص

تعتبر الشبكات المعرفة بالبرمجيات واحدة من أكثر التطورات التكنولوجية تأثيرا في تكنولوجيا المعلومات والاتصالات مقارنة بجميع تكنولوجيات الشبكات التقليدية.
هذه الشبكات التقليدية بها العديد من التحديات حيث انها تستغرق وقتا طويلا، والبيئات المختلفه التي تتطلب مستوى عال من الخبرة وتعقيد الشبكة حيث تكون مجزئة، وصعوبة التعلم لإدارة مثل هذه النظم والأجهزة الضخمة وأكثر من ذلك.
في هذه الدراسة، يتم محاكاة ببرنامج مينينيت باستخدام العديد من السيناريوهات المختلفة من أجل تقييم توصيل وأداء الشبكات المعرفة برمجيا مقارنة بالشبكات التقليدية.
النظر في صعوبة الشبكة باعتبارها التكنولوجيا الجديدة يتم تقييم أداء هذه السيناريوهات باستخدام أداة إيبرف للتحقيق في أن شبكات المعرفة بالبرمجيات يمكنها أن تلبي الوظيفة الأساسية للشبكات

إن متطلبات وظائف الشبكة الحالية ليست معقدة، ولا يلزم سوى التبديل الأساسي والتوجيه وتم عمل محاكاة هذه المتطلبات المختلفه والتوصل الي تحقيق كل متطلبات الشبكات التقليدية.

Table of Content

Dedication II

Acknowledgements III

Abstract IV

المستخلص... V

Table of Contents VI

List of Tables XI

List of Figures XII

Abbreviations.. XV

1. Chapter One: Introduction 2

1.1 Preface 2

1.2 Problem Statement...3

1.3 Proposed Solution 4

1.4 Methodology 4

1.5 Thesis Outlines 5

2. Chapter Two: Literature Review 7

2.1 Overview 7

2.2 Traditional IP Networks..7

2.3 MPLS Networks 8

2.4 SDN Network 9

2.4.1SDN Architecture 10

2.4.1.1 Openflow protocol..13

2.4.1.2 SDN concept...15

2.4.1.3 SDN Applications..16

2.4.1.4 SDN Controller...18

2.4.1.4.1Two sets of SDN controllers 19

2.4.1.4.2Open and community driven initiatives......................................19

2.4.1.5SDN Data path...20

2.4.1.6SDN Control to Data-Plane Interface (CDPI)..................................20

2.4.1.7SDN Northbound Interfaces (NBI)...20

2.4.1.8SDN Southbound Interfaces (SBI)..20

2.5 Traditional Networking to SDN 21

2.6Related Works in SDN...22

2.7differences between traditional and SDN types....................................25

3. Chapter Three: Methodology 27

3.1 Overview...27

3.2 SDN Evaluation..27

3.3The SDN Controllers Considered for the Experiment...............................27

3.3.1Open Daylight Helium 27

3.3.1.1Operation ODL 28

3.3.1.2Available Applications...28

3.3.2Pox SDN Controller 28

3.3.2.1General information about POX...28

3.3.2.2POX components....29
3.4The Simulation Software used for the Experiment 29
3.4.1Mininet Basic Operation 29
3.4.1.1Build SDN networks 30
3.4.1.2Start MiniEdit 30
3.4.1.3Alternative method: Mininet command line....32
3.4.1.4Mininet features....32
3.4.1.5Thereissomelimitationsinmininet....33
3.4.2Packet sniffer (Wireshark) 33
3.4.2.1Features 34
3.4.2.2Live capture and offline analysis....34
3.5Description of the Experiment....35
3.6The Setup of the Experiment 36
3.6.1Setting up ODL Helium 36
3.7The Experiment....36
3.8 The Topologies Used in the Experiment....38
3.8.1First Scenario Connectivity Test 38
3.8.2Second Scenario using Looped Topology 40
3.8.3Third Scenario using A Larger Number of Nodes 41
3.8.4Fourth Scenario using Utilizing Flows [Appendix I]....42
3.8.4.1Running the POX 43
3.8.4.2Script Explanation (important parts)....43

3.8.4.3First step install POX controller 45

3.8.4.4Second step run the script [Appendix I] 45

3.8.4.5Third step add feature to learn layer 3 routing 46

3.8.4.6Fourth step add default routing in mininet software 47

3.8.4.7Fifth step ping all hosts 47

4. Chapter Four: Results and Discussions 49

4.1 Overview 49

4.2 Simulation 1 Linear network 49

4.3 connectivity between hosts 51

4.4 Performance and Bandwidth 54

4.5 Simulation 2 Looped Topology 54

4.5.1 Connectivity between hosts 56

4.5.1.1 Performance and Bandwidth 56

4.5.1.2 Simulation 3 A Larger Number of Nodes 56

4.5.1.3 Connectivity between hosts 57

4.5.1.4Performance and Bandwidth 57

4.5.2 Simulation 4 Utilizing Flows [Appendix I] 59

4.5.2.1 Connectivity between hosts 60

4.5.2.2 Performance and Bandwidth 60

4.5.3 Analysis the Results 61

4.6 Transitioning to SDN 62

5. Chapter Five: Conclusion and Future Work 64

5.1Conclusion 64

5.2 Recommendations 64

References 66

LIST OF TABLES

Table No.	**Table Title**	Page No.
1	difference between traditional and software defined networking types	25

LIST OF FIGURES

Figure No.	Figure Title	Page No.
1	Open Network Foundation's software-defined network architecture	2
2	difference between traditional networking and software defined networking	7
3	MPLS in ISP environment	9
4	Traditional Architecture	11
5	SDN Architecture	12
6	Open Flow instruction set	14
7	SDN Controller	18
8	Transitional Models from Traditional Networking to SDN	21
9	Simple tree with three switches	30
10	enable CLI in miniedit	31
11	configure the controller as a remote controller	31
12	flow chare	35
13	install the image of Mininet	37
14	After installing Ubuntu 14.04 64-bit in VMware	37
15	Xterm to access to mininet	38
16	one controller and two switches	39
17	start the Linear topology	40
18	Linear topology looks like in Open daylight controller	40
19	Looped Topology	41

20	Larger Number of Nodes	42
21	SDN controller works as router	43
22	install POX controller in ubuntu	45
23	run the python script	45
24	testing the reachability	46
25	add feature to POX controller	46
26	add default routing	47
27	reach the h4 host	47
28	testing the hole network	47
29	start Open daylight controller	50
30	two switches in ODL controller	50
31	Test connectivity	51
32	topology in ODL controller after ping	51
33	start Wireshark	52
34	start Wireshark capture	52
35	ARP within Open flow	53
36	ICMP within open flow	53
37	open flow 1.3	53
38	Node Traffic statistics	54
39	Bandwidth with host 1 and host 2	54
40	wireshark with looped topology	54
41	Looped topology in ODL Controller	55
42	Looped topology in controller after send some traffic	55
43	connectivity testing for looped topology	56
44	Testing bandwidth between h1 and h2 for looped	56

	topology	
45	Testing bandwidth between h1 and h4 for looped topology	56
46	large number of nodes topology in ODL controller	57
47	bandwidth between h1 and h5 for large number of nodes	57
48	bandwidth between h1 and h7 for large number of nodes	58
49	The node connector statistics in controller in large node topology	58
50	Open flow in Wireshark for large node topology	58
51	The details of open flow for large node topology	59
52	starting the code in mininet software	60
53	Testing connectivity between h1 and h4	60
54	bandwidth between hosts in utilization topology	61

Abbreviations

SDN	Software Define Network
API	Application Programming Interface
COTS	Commercial of the Shelf
CPE	Customer Premises Equipment
DDoS	Distributed Denial-of-Service (attack)
GUI	Graphical User Interface
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPTV	Internet Protocol Television
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISP	Internet Service Provider
L2	Layer 2 (of The OSI model)
L3	Layer 3 (of The OSI model)
LLDP	Link Layer Discovery Protocol
MPLS	Multiprotocol Label Switching
NaaS	Network as a Service
NAT	Network Address Translation
NFV	Network Functions Virtualization
ONF	Open Networking Foundation
OSI	Open Systems Interconnection

OSS	Operations Support System
QoS	Quality of Service
RAM	Random Access Memory
REST	Representational State Transfer
SDAN	Software Defined Access Network
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
VM	Virtual Machine
WLAN	Wireless Local Area Network

Chapter One:

Introduction

1. Introduction

1.1 Preface

Software-defined networking continues to be one of the most hyped technology evolutions in information and communication technology.

Software-defined networking (SDN) centralizes network control, moving it from switches and routers to SDN controllers. This allows network traffic to be managed in the context of an entire network rather than from interconnected but locally controlled devices. SDN controllers use a standard interface, often Open Flow, to program tables in controlled network elements. These tables, called flow tables, allow very granular control of network traffic, much more so than Ethernet based switching or IP based routing.

Finally, SDN allows network operators to programmatically interface with controllers. See Figure 1 [1].

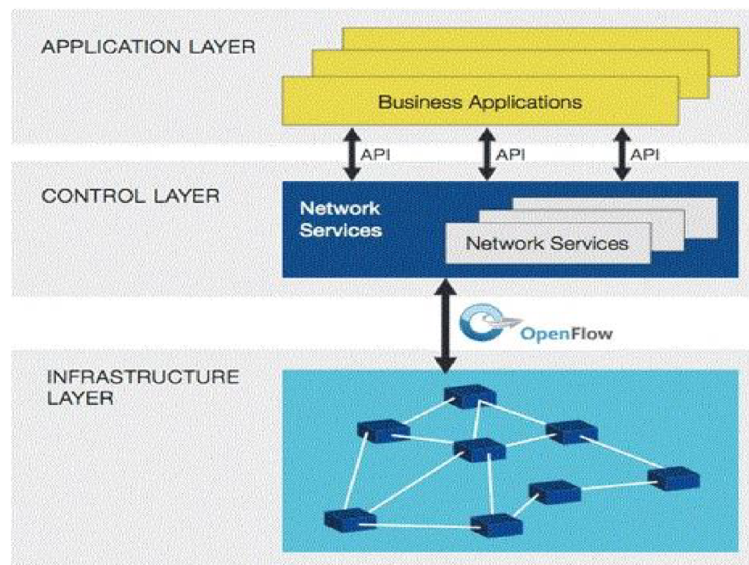


Figure 1: Open Network Foundation's software-defined network architecture

SDN is widely seen as a sign can't step forward towards a completely re-envisioned paradigm for modern packet-switched networks, current incarnations (most notably, openflow) appear to fall short on these promises.

In these days, network providers want to simplify a network management. This is done by decoupling the legacy network system that is composed of a control plane and a data plane.

Software Defined Networking (SDN) divides a network system into a decision plane (control plane) and a forwarding plane (data plane) and it is an approach to computer networking that allows network administrators to manage network services through abstraction of higher-level functionality. It has attracted attentions for even transport networks [2].

The purpose of a transport network is to provide a reliable aggregation and transport infrastructure for any client traffic type. With the growth of packet-based services, operators are transforming their network infrastructures while looking at reducing capital and operational expenditures.

1.2 Problem Statement

Most of companies have been using old technologies in smart grid networks and are clearly in need of new communication techniques. Most companies are still relying on point-to-point radio wave links and leased lines for communication. These technologies do not provide adequate performance, security, and cost-effectiveness for the time critical control signals from the substation.

There are limitations associated with traditional networkingtime-consuming, Multi-vendor environments require a high level of expertise and complicate network segmentation and also the inconvenience and difficulty of learning to manage such a huge systems and devices.

In conclusion, to overcome these and other traditional networking limitations, the time has come to introduce a new perspective on network management.

1.3 Proposed Solutions

Software Defined Networking (SDN) is rapidly becoming the new buzzword in the networking business. Expectations are that this emerging technology will play an important role in overcoming the limitations associated with traditional networking.

This study in SDN was conducted to help devise alternatives for the future development of the network. Not to necessarily offer a ready solution but to see what the state of the art is and if it would be a viable option for such a network in the future; can it do what is required in the traditional network's current state and how could it make it better. The format and style of the thesis have been chosen to provide some clarity between the promises of SDN, what it currently is and how it works technically speaking.

1.4 Methodology

Our goal is to bring and test the SDN Basic Function compare to the Traditional networkand evaluate it. To achieve this goal, we need in order to examine if SDN network can be utilized in traditional network environment, can support the existing legacy applications and co-exist with

the traditional network, we implemented a network using open-source is used as SDN controller, and the network is emulated using Mininet software to implement the basic function of the traditional networks and evaluate the performance, Bandwidth and packet loss of the new technology in different scenarios.

1.5 Thesis Outlines

The reminder of the document is organized in the following manner: Chapter Two provides technical background research relevant to SDN networks in Traditional networks. Chapter Three describes the methodology and emulation tool that used in the research. Chapter Four presents the results and discussion of the data collected. Chapter Five describes the conclusions and areas for recommendations.

Chapter Two:

Literature Review

2. Literature Review

2.1 Overview

This chapter describe briefly traditional network, MPLS network, the architecture of SDN network and Previous Research in SDN with technical background.

2.2 Traditional IP Networks

In traditional IP networks, routing protocols are used to distribute Layer 3 routing information. Regardless of the routing protocol, packet forwarding is based on the destination address alone. Therefore, when a packet is received by the router, it determines the next-hop address using the packet's destination IP address along with the information from its own forwarding/routing table. This process of determining the next hop is repeated at each hop (router) from the source to the destination [11].

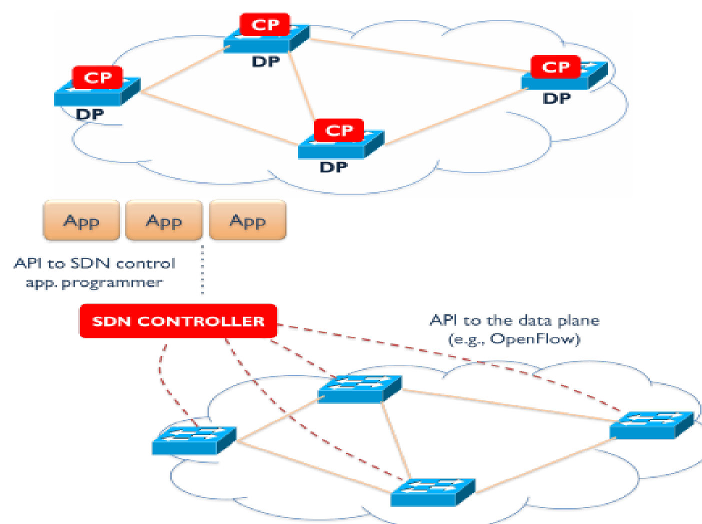


Figure 2: difference between traditional networking and software defined networking [11]

In Figure 2,it consists of control plane, management plane and data plane. This are referred as static kind of networks and also depicts layers within software defined networks. It consists of one layer of data plane along with open flow API. This is interfaced with controller housing both the control and management plane. Above which on both there exists application layer [11].

2.3 MPLS Networks

MPLS is a latest technology before SDN technology that optimizes the traffic forwarding in a network by avoiding complex lookups in the routing table. The traffic is directed based on labels contained in an MPLS packet header. The labels define only the local node to node communication and are swapped on every node. This process allows very fast switching through the MPLS core. MPLS relies on traditional IP routing protocols to determine the best routes and to receive topology updates and predetermines the path the packet will take through the network. This process is performed by the MPLS edge router and thus reduces the processing requirements for the core switching routers. These paths are called Label-Switched Paths (LSPs).

Most of MPLS networks using in ISP environment as presented in Figure 3.

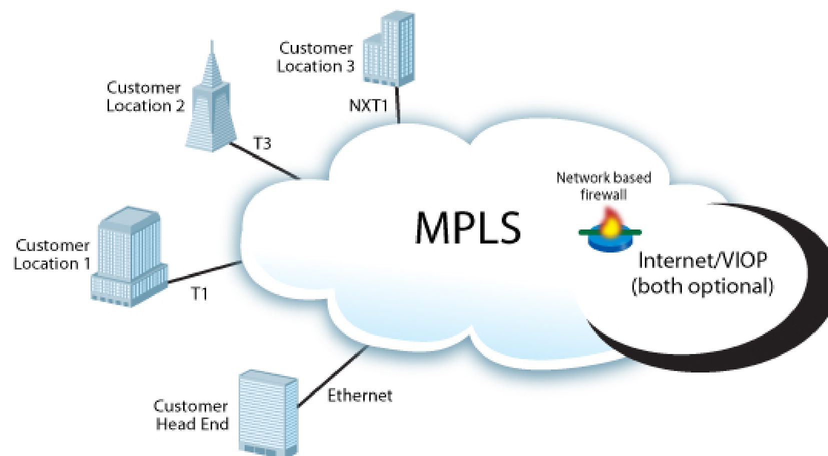


Figure 3: MPLS in ISP environment

2.4 SDN Network

Software-defined networking (SDN) is a new networking architecture that comes after MPLS Technology is proposed as a facilitating technology for network evolution and network virtualization. It has attracted significant attention from both academic researchers and industry. One the main organizations that contribute to the development of SDN is the Open Network Foundation (ONF) which is a non-profit industry consortium of network operators, service providers and vendors that promotes the SDN architecture and drives the standardization process of its major elements [16].

ONF defines SDN as a technology where "network control is decoupled from forwarding and is directly programmable". It concentrates

the network intelligence in software-based central controllers, which aims to bring better and more efficient control, customizability and adaptability. The main benefits that the SDN technology might offer are listed below:

- Centralized unified control of network devices from different vendors
- Better automation and control, as an abstraction of the real network is created
- Simplified and quicker implementation of innovations, as the network control is centralized and there is no need every individual device to be reconfigured
- Improved network reliability and security, because of fewer configuration errors and unified policy enforcement, provided by the automated management and the centralized control
- Ability to easily adapt the network operation to changing user needs, as centralized network state information is available and can be exploited

2.4.1SDN Architecture

Software-defined networking (SDN) has been primarily discussed as network architecture where Layer2 technologies implemented. However, the network, like the economy, is global and the enterprise wide area network (WAN) becomes an essential component of that global network. SDN programmability within the datacenter will only solve one aspect of the larger issue. That programmability needs to extend all the way across the WAN to realize true benefits of software defined networks. As they say, you are as good as your weakest link [15].

Let us first try and peel back the layers of SDN and how it impacts networking. Networking typically involves a collection of switches and routers that work in harmony to achieve end to end communication. The key functions of these network elements can be segmented into layers of management, data plane and control plane. The traditional way of making these nodes work with each other is by implementing protocols running at each of these nodes to exchange information. This creates a distributed architecture, where every node across the network needs to be at a similar state to get the desired end result. In addition, these protocols are very rigid in what they can and cannot do. The result is a very static network architecture that is not adaptive to change as presented in Figure 4 [15].

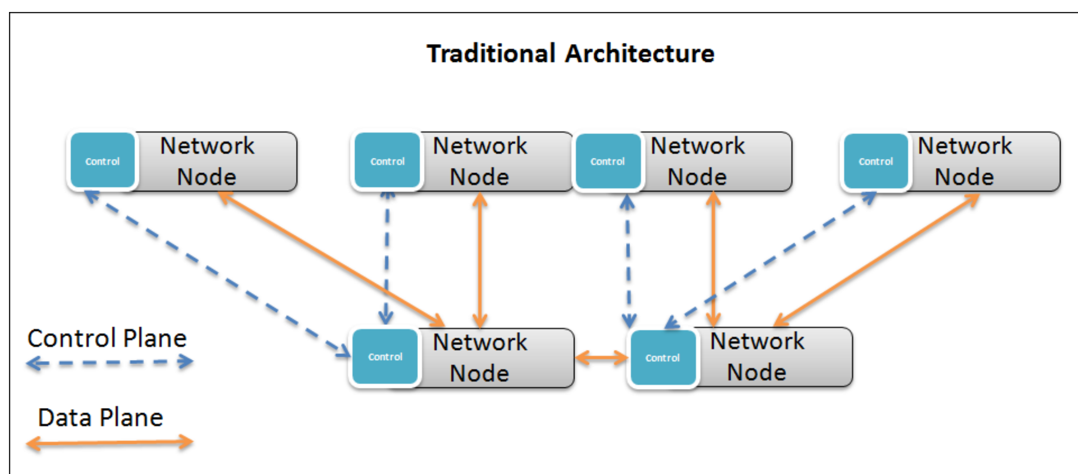


Figure 4: Traditional Architecture [15].

Now consider what would happen if we remove the protocols and instead open up a standard set of APIs. Then, build a centralized control plane that uses these APIs to program the network elements. This control plane will have a global view of the network and can make smart decisions. For

example, how can one carve out a dedicated path between 2 servers? If we had switches opening up APIs indicating the flow to the output port mapping it is a matter of programming all the elements with that information. Imagine trying to do that with the spanning tree protocol instead! This is just a very high level concept, but the fundamental idea is that network elements need to be programmable and cannot be static within a fluid environment like the Cloud, where provisioning needs to happen on demand and elasticity is a key requirementas presented in Figure 5[15].

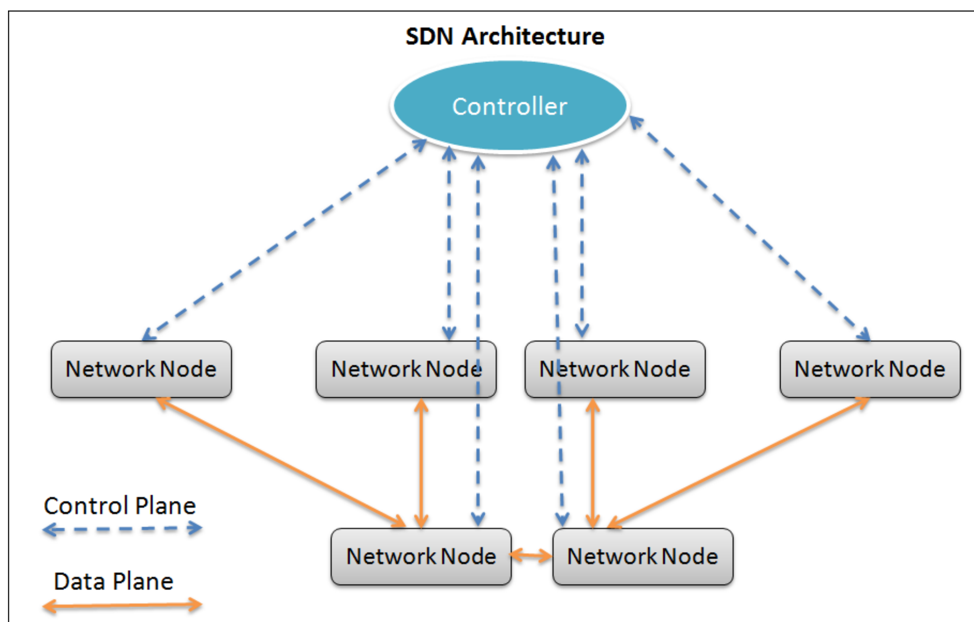


Figure 5: SDN Architecture [15]

Moving the same concept into enterprise networking, Firewalls, VPN, WAN optimization solutions and, QoS are some of the aspects of WAN technologies built on a foundation of L3 routing. L3 routing is destination based and is not flow aware. It does have significant benefits over L2

networks, like support for multi pathing, VPNs but is built on protocols running in a distributed manner and lacking programmability [15].

SDN has been designed to simplify network configuration and facilitate innovation. SDN paradigm decouples the control plane and the data plane and concentrates the data forwarding decisions into a centralized software controller. As a result, the underlying network devices' functions are reduced to simple data forwarding. Instead of programming thousands of devices the network configuration is performed on simplified network abstraction. This allows the implementation of various software modules that can exert dynamic control on the network functions [15], also The centralized control function of the SDN architecture allows consistent policies to be enforced with ease. Common networking functionalities can also be configured via the supported APIs. The deployment of services, such as routing, security, access control, bandwidth management, traffic engineering, quality of service, energy optimization can be configured much easily. The goal of the SDN developers is to ensure multi-vendor support [15].

2.4.1.1Open FlowProtocol

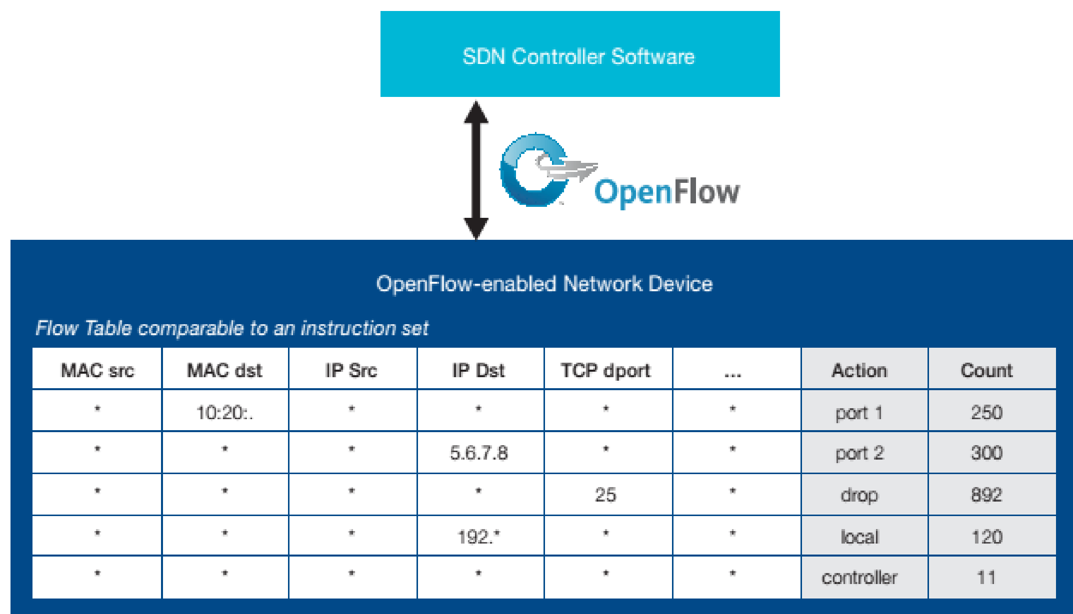
Open Flow is currently the only open standard for implementing SDN and it is a standardized protocol that defines the communication between the control and the data forwarding plane in the SDN architecture. It moves the control out of the networking devices (routers, switches, etc.) into the centralized controller. The protocol uses the concept of flows that use match rules to determine how the packets will be handled. The protocol

is configured on both sides – the device and the controller. The forwarding device in an Open Flow scenario is an Open Flow switch that contains one or more flow tables and an abstraction layer that communicates with the controller. The flow tables are filled with flow entries which define how the packet will be forwarded, depending on the particular flow they are part of [14].

The flow entries have the following fields:

- match fields – might contain information from the packet headers, ingress port or metadata and matches the packets to a certain flow
- counters – collect statistic about the particular flow
- actions – define how the incoming packets to be handled

An example of the Open Flow instruction set is presented on Figure 6.



MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Figure 6: Open Flow instruction set [14]

SDN is possible without using the Open Flow standard, but proprietary

alternatives would lock an operator into vendor-defined solutions, capabilities and pricing. This would greatly reduce the value of SDN as it would result in the loss of both device interoperability and multi-network interoperability [14].

An Open Flow switch essentially receives data packets, extracts the packet header and matches the value to the entries in the flow table. If the value is found the packet is forwarded according to the instructions in the actions fields. In case the value does not match any of the entries, the packet is handled according to the instructions defined in the table-miss entry. The packet can be either dropped, forwarded to the next flow table or send to the Open Flow controller via the control channel. Another possibility, employed in switches that have both Open Flow and non-Open Flow ports, is to forward the packet using standard IP-forwarding schemes. The Open Flow switch communicates with the controller over a secure channel. The controller adds, removes or updates the entries in the flow table [14].

2.4.1.2 SDN Concept

Software-defined networking (SDN) is an architecture purporting to be dynamic, manageable, cost-effective, and adaptable, seeking to be suitable for the high-bandwidth, dynamic nature of today's applications. SDN architectures decouple network control and forwarding functions, enabling network control to become directly programmable and the underlying infrastructure to be abstracted from applications and network services [13].

The Open Flow protocol can be used in SDN technologies. The SDN architecture is:

- Directly programmable: Network control is directly programmable because it is decoupled from forwarding functions.
- Agile: Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs.
- Centrally managed: Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch.
- Programmatically configured: SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves because the programs do not depend on proprietary software.
- Open standards-based and vendor-neutral: When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols.

2.4.1.3SDN Applications

The SDN architecture is claimed to greatly simplify network management and provide an immense number of new services via the programmable software modules. A summary of the application scenario that will benefit from employing the Open Flow architecture are described in and briefly summarized as following [13].

- Enterprise networks – the centralized control function of SDN can be particularly beneficial for enterprise networks in different ways. For example, network complexity can be reduced by removing middle boxes and configuring their functionality within the network controller. Different network functions implemented via SDN include NAT, firewalls, load balancers and network access control. An approach for realizing consistent network upgrade, using high-level abstractions is described in [13].
- Data centers – power consumption management is a major issue in data centers, as they often operate below capacity in order to be able to meet peak demands. a network power manager is described that turns off a subset of switches in a way to minimize power consumption while ensuring the required traffic conditions. A real life example of SDN application in the context of data centers is presented. They describe SDN-based network connecting Google data centers worldwide. The deployment was motivated by the need of customized routing and traffic engineering, as well as scalability, fault tolerance and control that could not be achieved with traditional WAN networks [13].
- Infrastructure-based wireless access networks – an SDN solution for enterprise wireless LAN networks is proposed. The solution builds an abstraction of the access point infrastructure that separates the association state from the physical access point. The purpose is to ensure proactive mobility management and load balancing [13].

2.4.1.4SDN Controller

The controller is the core of an SDN network. It lies between network devices at one end and applications at the other end. Any communications between applications and devices have to go through the controller [17].

SDN controllers are based on protocols, such as OpenFlow to configure network devices and choose the optimal network path for application traffic and to allow servers to tell switches where to send packets as presented in Figure 7.

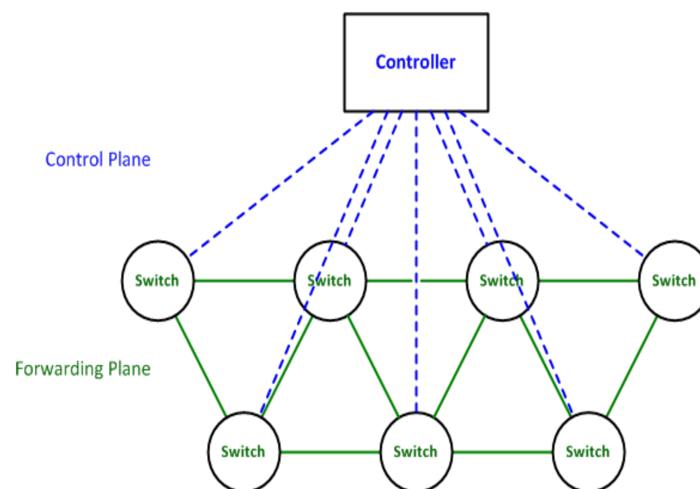


Figure 7: SDN Controller

2.4.1.4.1 Two Sets Of SDN Controllers

1. SDN controllers for the NFV Infrastructure of a datacentre,
2. Historical SDN controllers for managing the programmable switches of the network [17].

In case of SDN controllers for the NFV Infrastructure of a datacentre, they are mostly designed to provide some policy and centralized managements for the Open stack Neutron networking layer that shall provide inter-working between the virtual ports created by Nova. The defacto technology of the SDN controllers is to manage the Linux kernel features made of L3 IP routing, Linux bridges, iptables or ebtables, network namespaces and Open vSwitch [17].

2.4.1.4.2 Open and Community Driven Initiatives

Open Daylight controller baseline project upon which many other controllers are built [17].

- ONOS
- Project Calico
- The Fast Data Project
- Project Floodlight
- Beacon
- NOX/POX
- Open vSwitch
- vneio/sdnc (SDN Controller from vne.io)
- Ryu Controller (supported by NTT Labs)
- Cherry
- Faucet (Python based on Ryu for production networks)

2.4.1.5SDN Data Path

The SDN Data path is a logical network device that exposes visibility and uncontested control over its advertised forwarding and data processing capabilities [13].

2.4.1.6SDN Control to Data-Plane Interface (CDPI)

The SDN CDPI is the interface defined between an SDN Controller and an SDN data path, which provides at least (i) programmatic control of all forwarding operations, (ii) capabilities advertisement, (iii) statistics reporting, and (iv) event notification. One value of SDN lies in the expectation that the CDPI is implemented in an open, vendor-neutral and interoperable way [13].

2.4.1.7SDN Northbound Interfaces (NBI)

SDN NBIs are interfaces between SDN Applications and SDN Controllers and typically provide abstract network views and enable direct expression of network behavior and requirements [13].

2.4.1.8SDN Southbound Interfaces (SBI)

In the architecture of software-defined network, Southbound APIs (application program interface) that is used to communicate between the controller and the SDN network switches and routers [13].

2.5Traditional Networking to SDN

The research paper "Opportunities and Research Challenges of Hybrid SoftwareDefined Networks" (Vissicchio et al., 2014) proposes four different models toimplement hybrid SDN each with its own strengths and use cases [18].

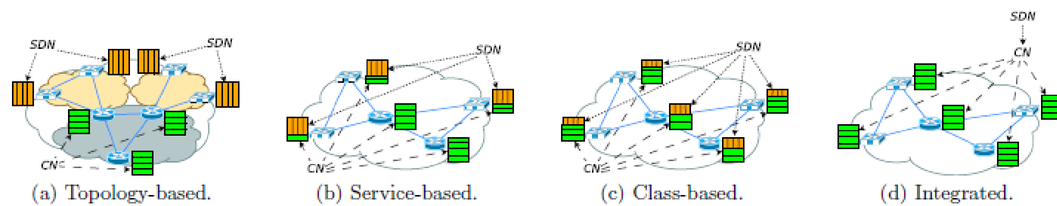


Figure 8: Transitional Models from Traditional Networking to SDN [18].

(a) Topology-based.

Traditional and SDN exist as physically and logically isolated zones within the network and converse with each other as they would with any remote network [18].

This model would fit any network that has already been divided into smaller, also the parts can be independently switched to SDN while the other parts keep operating normally [18].

(b) Service-based.

Traditional and SDN overlap at least partially physically. Network services

provided originally by the logical traditional network are gradually moved on to the SDN side so that both networks can still access them. This method

allows forfirst implementing SDN nodes into the key points of the network to for exampleenable SDN's ability to utilize a looped topology [18].

(c) Class-based.

Traditional and SDN overlap completely physically. Network traffic is divide into classes and then class-by-class moved from the logical traditional to the SDN sideof the network. Retaining the traditional network would allow the traffic to bemoved back if for some reason some kind of traffic wouldn't behave correctlywithin the SDN network [18].

(d) Integrated.

In the integrated model at first the SDN controller controls the traditional network nodes and then over time the nodes are changed to SDN nodes. This allowsimplementing SDN quickly to an existing network. However, this kind of interfacebetween the SDN controller and the traditional nodes does not exist yet [18].

2.6Related Works in SDN

In 2015, FarisKeti and ShavanAskar[3] publish the paper "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments" in this paper they describe the performance of Mininet tool for emulating SDN networks was evaluated. During this study many capabilities of Mininet emulator in the SDN paradigm evaluation was covered, from the creation of basic topologies with reference controller to the ability of connection with remote controllers (in this case POX controller). In addition, this paper took into consideration the following scenarios; changing the topologies, increasing the number of nodes,

controlling the behavior of forwarding hardware (switches). The effect of simulation environment limited resources was studied and a comparison between results for two different environments.

In 2015, Wenfeng Xia and Yonggang Wen, Senior Member, IEEE, ChuanHengFoh,[4] publish the paper "A Survey on Software-Defined Networking" this paper describe the concept of SDN and highlighted benefits of SDN in offering enhanced configuration, improved performance, and encouraged innovation. Moreover, we have provided a literature survey of recent SDN researches in the infrastructure layer, the control layer, and the application layer, as summarized in Table VI. Finally, we have introduced OpenFlow, the de facto SDN implementation.

In 2014, Foukas et al,[5] publish the paper "Software Defined Networking" it is a bout detailing the components of SDN and as such clarifies what a SDN system consists of. To understand what SDN does it is good to understand the components that do it. As is common SDN discussed in the paper is SDN implemented by using Open Flow. Some real-life scenarios of SDN are mentioned, for example how SDN might be used in data center and cellular networks where it is at its best.

In 2014, Jammal et al,[6] publish the paper "Software Defined Networking: State of the Art and Research Challenges" The applications and challenges of SDN are discussed in this paper the application detailed most is the data center network, how SDN is able to improve the performance and reliability over a traditional network. The relationship of SDN and NFV (Network Functions Virtualization) is discussed. The

challenges of SDN when implementing the concept to a real-life use case are made apparent and how some of them have been solved. The case they make is that SDN works really well in some scenarios but not in all of them. Caution should be exercised when trying to implement SDN in enterprise networks.

In 2014, De Oliveira et al,[7]publish the paper"Using Mininet for Emulation and Prototyping Software-Defined Networks." for testing SDN and most research has been using the Mininet SDN network simulator. go through basic use and test the scalability of it in the paper Mininet is a simple but powerful tool for simulating a SDN network. When used as a supporting document to the official documentation this paper helps getting used to using Mininet. Mininet's usability for simulation is evaluated and alternative simulation programs presented. In the paper, there is also a performance test of Mininet using a tree topology that supports the success of Mininet as the simulator of choice for SDN.

2.7difference between traditional and SDN types

In Table 1 we describe the difference between the traditional network and SDN networks according to the Previous Research.

Table 1: difference between traditional and software defined networking types [12].

NM	Traditional Networking	Software Defined Networking
1.	They are Static and inflexible networks. They are not useful for new business ventures. They possess little agility and flexibility	They are programmable networks during deployment time as well as at later stage based on change in the requirements. They help new business ventures through flexibility, agility and virtualization.
2.	They are Hardware appliances.	They are configured using open software.
3.	They have distributed control plane.	They have logically centralized control plane.
4.	They use custom ASICs and FPGAs.	They use merchant silicon.
5.	They work using protocols.	They use APIs to configure as per need.

Chapter Three:

Methodology

3. Methodology

3.1 Overview

In this chapter, we discuss about SDN controllers, software that we used in simulations and finally the implementations of SDN networks compare of traditional networks.

3.2SDN Evaluation

To test SDN in practice is not straight forward as the technology is still veryyoung. There are actual physical devices available but not widely and nor cheaply.

The main component of a SDN network, the controller (software), on the otherhand has many alternatives readily available for download for free.

3.3The SDN Controllers Considered for the Experiment

We use two types of controller:

3.3.1 Open Daylight Helium

Open Daylight is a Linux foundation project supported by many of the big names in networking such as Cisco, HP, Juniper and VMWare. It is expected to be one of the most popular controller platforms, Heliumopen Flow 1.3 natively. The applications for Open Daylight are written in Java [19].

3.3.1.1 Operation ODL

Helium is run as a Karaf distribution and any additional parts can be installed within the running distribution.

3.3.1.2 Available Applications

Basic SDN and switching functionality is included. Of the more advanced applications included Defense4All, a DDoS (Distributed Denial-of-Service (attack)) detection and protection app, and SNMP4SDN, SNMP (Simple Network Management Protocol) monitoring, can be mentioned.

3.3.2 Pox SDN Controller

POX is a platform for the rapid development and prototyping of network control software using Python. It’s one of a growing number of frameworks (including NOX, Floodlight, Trema, etc.,) for helping to write OpenFlow.

POX as well as being a framework for interacting with OpenFlow switches, it can be used as the basis for some of our ongoing work to help build the emerging discipline of Software Defined Networking. It can be used to explore and prototype distribution, SDN debugging, network virtualization, controller design, and programming models [8].

3.3.2.1 General Information about POX

POX provides a framework for communicating with SDN switches using either the OpenFlow or OVSDB protocol. Developers can use POX to create an SDN controller using the Python programming language. It is a

popular tool for teaching about and researching software defined networks and network applications programming [8].

3.3.2.2 POX Components

POX components are additional Python programs that can be invoked when POX is started from the command line. These components implement the network functionality in the software defined network. POX comes with some stock components already available [8].

3.4 The Simulation Software used for the Experiment

We use Mininet simulation as following

3.4.1 Mininet Basic Operation

Mininet is a network emulator, or perhaps more precisely a network emulation orchestration system. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. A Mininet host behaves just like a real machine; you can ssh into it (if you start up sshd and bridge the network to your host) and run arbitrary programs (including anything that is installed on the underlying Linux system) [20].

In short, Mininet's virtual hosts, switches, links, and controllers are the real thing they are just created using software rather than hardware and for the most part their behaviour is similar to discrete hardware

elements[21]. Mininet can be used to define a SDN enabled topology using a relatively simple python script.

3.4.1.1Build SDN Networks

There are two methods for building the topology of SDN network

3.4.1.2Start MiniEdit

We will use MiniEdit, the Mininet graphical user interface, to set up an emulated network made up of OpenFlow switches and Linux hosts [21].

To start Mininet, run the following command on a terminal window connected to the Mininet VM:

```
mininet@mininet-vm: ~$ sudo ~/mininet/topology/miniedit.py
```

Now the Mininet window will appear on your computer's desktop [21].
Then we Build the network consisting of a tree to switches with a central core switch connected to two other switches that are connected to two hosts, each. Connect a controller to all the switches as in Figure7.

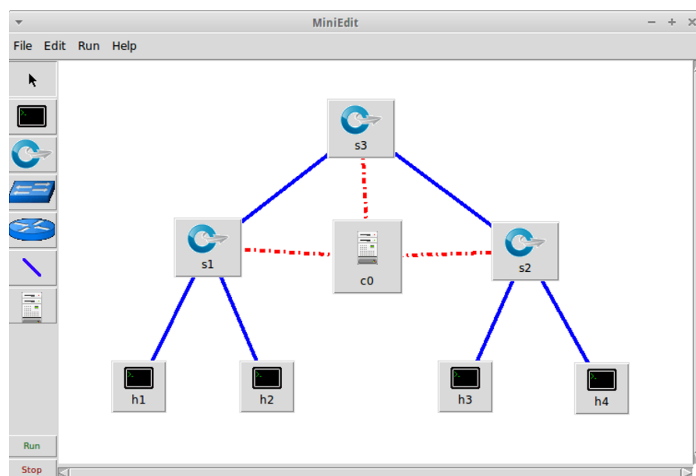


Figure 9: Simple tree with three switches

Ensure that the MiniEdit preferences are set so that we can use the MiniEdit command line after starting the simulation as presented in Figure8 [21].

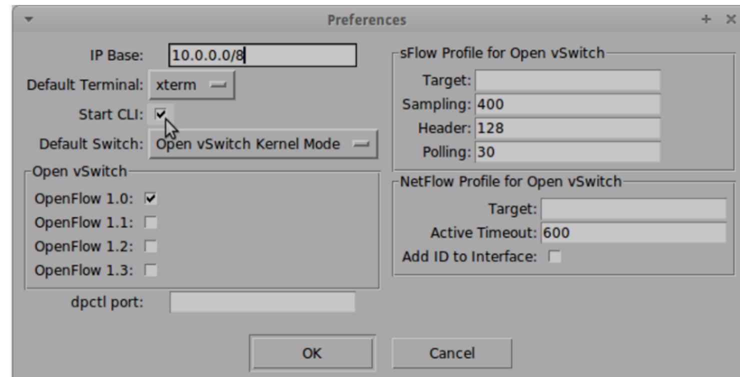


Figure 10: enable CLI in miniedit

Set up the controller as a remote controller. Then select Remote Controller in the controller properties window as in Figure 9 [21].

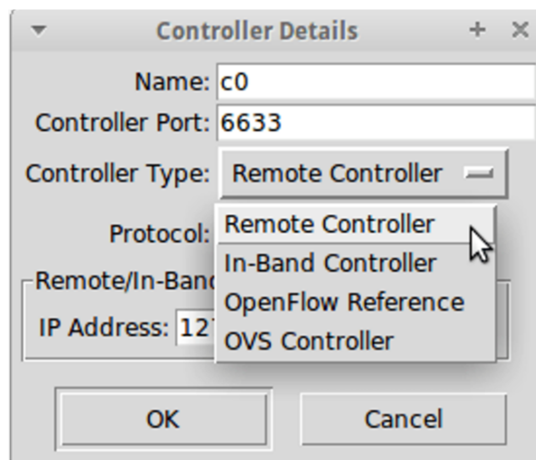


Figure 11: configure the controller as a remote controller

When default settings are used, MiniEdit configures OpenFlow switches to try to communicate with a remote controller using the host system's loopback IP address and the default OpenFlow port number [21].
Then we Start the MiniEdit simulation and we should:

a) save the MiniEdit topology for future use.
b) start the simulation by clicking on the Run icon in the MiniEdit tool bar.
c) The MiniEdit console window will show information about the simulation starting and then will display the Mininet CLI prompt.

3.4.1.3Alternative Method: Mininet Command Line

As an alternative to using MiniEdit, the same network can be set up using the Mininet topology commands [21].

mininet@mininet-vm: ~$ sudo mn --topo Name --controller remote

3.4.1.4Mininet Features

- It's fast - starting up a simple network takes just a few seconds. This means that your run-edit-debug loop can be very quick [23].
- You can create custom topologies: a single switch, larger Internet-like topologies, the Stanford backbone, a data centre, or anything else [23].
- You can run real programs: anything that runs on Linux is available for you to run, from web servers to TCP window monitoring tools to Wireshark [23].
- You can customize packet forwarding: Mininet's switches are programmable using the open Flow protocol. Custom Software-Defined Network designs that run in Mininet can easily be transferred to hardware open Flow switches for line-rate packet forwarding [23].

- You can run Mininet on your laptop, on a server, in a VM, on a native Linux box (Mininet is included with Ubuntu 12.10+!), or in the cloud (e.g. Amazon EC2.) [23].
- You can share and replicate results: anyone with a computer can run your code once you've packaged it up [23].
- You can use it easily: you can create and run Mininet experiments by writing simple (or complex if necessary) Python scripts [23].
- Mininet is an open source project, so you are encouraged to examine its source code on https://github.com/mininet, modify it, fix bugs, file issues/feature requests, and submit patches/pull requests. You may also edit this documentation to fix any errors or add clarifications or additional information [23].

3.4.1.5There isSome Limitations in Mininet

Mininet based networks cannot (currently) exceed the CPU or bandwidth available on a single server.Mininet cannot (currently) run non-Linux-compatible OpenFlow switches or applications; this has not been a major issue in practice [22].

3.4.2Packet Sniffer (Wireshark)

Wireshark is a free and open-source packet analyser. It is used for network troubleshooting, analysis, software and communications protocol development, originally named Ethereal. It lets you capture and interactively browse the traffic running on a computer network. It is the de

facto (and often de jure) standard across many industries and educational institutions. [9]

3.4.2.1 Features

Wireshark has a rich feature set which includes the following:

- Live capture and offline analysis.
- Data display can be refined using a display filter.
- Multi-platform: Runs on Windows, Linux, OS X, Solaris, FreeBSD, NetBSD, and many others.
- Captured network data can be browsed via a GUI, or via the TTY-mode TShark utility.
- The most powerful display filters in the industry.

3.4.2.2 Live Capture and Offline Analysis

Capturing live network data is one of the major features of Wireshark. The Wireshark capture engine provides the following features:

- Capture from different kinds of network hardware (SIP, Ethernet, Token Ring, ATM).
- Stop the capture on different triggers like: amount of captured data, captured time, captured number of packets.
- Simultaneously show decoded packets while Wireshark keeps on capturing.
- Filter packets, reducing the amount of data to be captured.

3.5 Description of the Experiment

First all the controllers were tested with Mininet to see how they are installed and how their basic operation has been handled.

Keeping in mind the requirements for the operation of the current network, basic switching and routing, that the controller should be able to handle.

Then when a controller was chosen its abilitieswere tested with more complex topologies. In order to get more familiar with thesimulation software testing was begun with a very simple topology that was thengradually extended to a bigger network, to see if the basic functions of the current network could be met.

As in Figure 12,describe the flow chare from beginning at the experiment until we reach our goal that SDN can meet the Functional of traditional network.

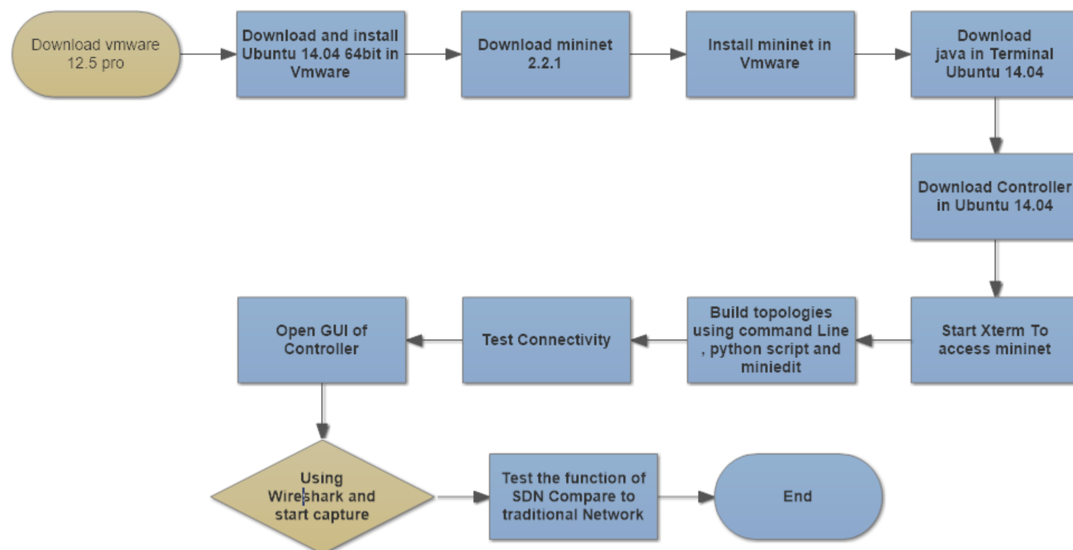


Figure 12: flow chart

3.6The Setup of the Experiment

Mininet emulation software comes as a pre-built virtual machine (VM) image, The VM was allocated two 1.7 GHz Intel Core 7 processors and 4 gigabytes of RAM (Random Access Memory). The different controllers were then installed on the Ubuntu 14.0.

IPv4 was used, however in the simulations conducted difference between IPv4 and IPv6 wouldnot have made a difference; Mininet uses hostnames, as is almost mandatory withIPv6, so the under-laying IP version does not matter, however only controllersrunning Open Flow 1.2 or greater have IPv6 support.

3.6.1Setting upODL Helium

Setting up Open Daylight Helium is straightforward. Download the package, extract the package, run it. Open Daylight Helium requires the installation ofsome additional components to function.

3.7The Experiment

To begin with the Mininet 2.2.0 VM was downloaded and using Virtual Machine to install the image of Mininet as in Figure 13.

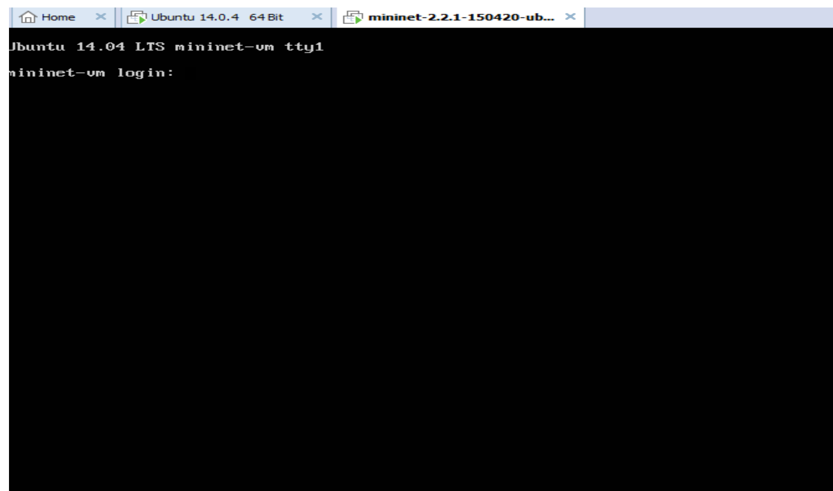


Figure 13: install the image of Mininet

After we install mininet in VMware we install Ubuntu 14.04 64-bitoperating system to install the controller on it as in Figure 14.

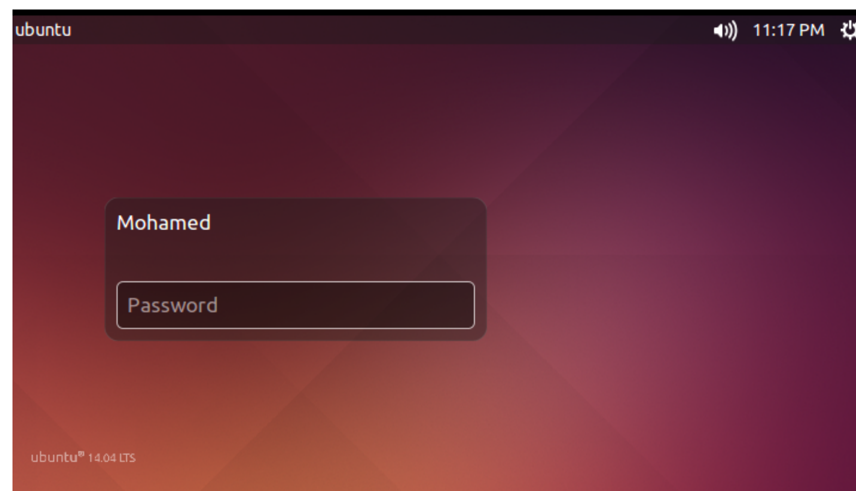


Figure 14: After installing Ubuntu 14.04 64-bit in VMware

After that we use xterm to access to mininet machine and we can use it to simulate an SDN network as in Figure 15.

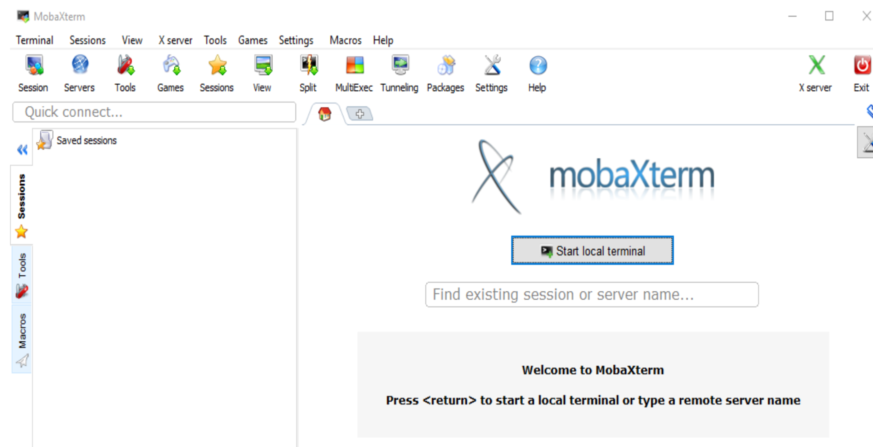


Figure 15: Xterm to access to mininet

3.8The Topologies Used in the Experiment

Topologies of different sizes and complexities were constructed to simulate basic functions found in the current network and to make use of some SDN specificfunctions.

3.8.1First Scenario Connectivity Test

To begin the testing a very simple Linear topology consisting of two SDN switches and two hosts were used just to see how Mininet works in Connecting the nodes together as seen in Figure 16. The functionality simulated hereis basic switching capability in traditional networks. Mininet allows for topology definition via command line parameters.

We use command topo=Linear,2spawns two switches connected to each other with a link and has one host on each switch as presented in Figure 16.

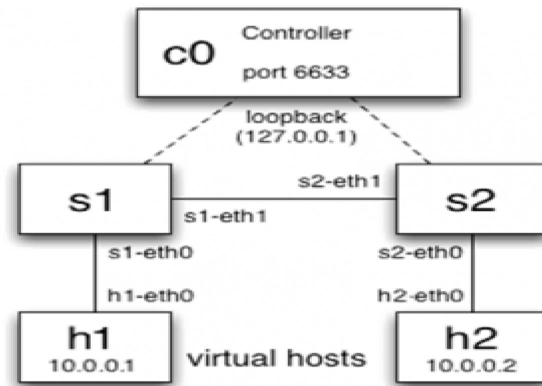


Figure 16: one controller and two switches

$ sudo mn --topo linear --switch ovsk --controller remote

In the above command, there are some important keywords worth paying attention to:

- mac: Auto set MAC addresses
- arp: Populate static ARP entries of each host in each other
- switch: ovsk refers to kernel mode OVS
- controller: remote controller can take IP address and port number as options.

We use command line as in Figure 17 to start linear topology that consist of two switches, two hosts and one controller as in Figure 16.

```
*** Cleanup complete.
mininet@mininet-vm:~$ sudo mn --controller=remote,ip=192.168.45.142 --topo=linea
r,2
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s2) (s2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet>
```

g to the professional edition here: http://mobaxterm.mobatek.net

Figure 17: start the Linear topology

This what the topology looks like in Open daylight controller, after sending some traffic in network as presented in Figure 18.

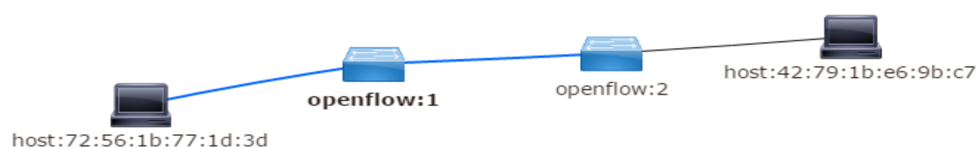


Figure 18: Linear topology looks like in Open daylight controller

3.8.2Second Scenario using Looped Topology

One of the most important features in SDN is the possibility of using a partially (or fully) meshed network without having any loops; because the controller can utilize all links automatically. The topology displayed in

Figure 19,consists of four switches with links to all adjacent switches and a host behindeach of the switches for testing connectivity. This topology simulates notonly switching but also a new feature that could be implemented to make thecurrent network better; more links between nodes makes for a faster, more reliable network. To make the topology work the controller was also added and then usedto test the basic functionality of the network.

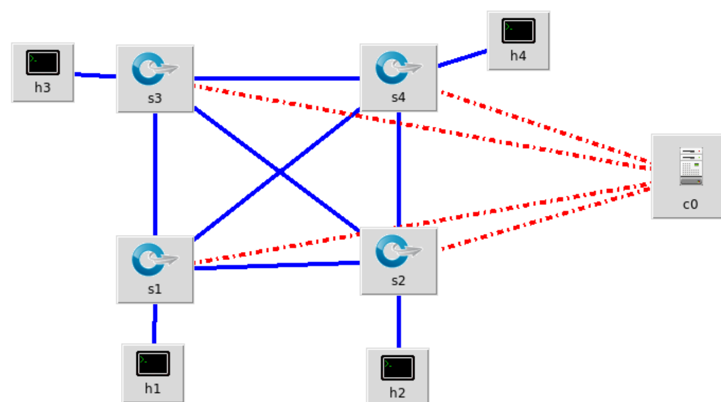


Figure 19: Looped Topology

3.8.3 Third Scenario using A Larger Number of Nodes

More switches and hosts added and connected to gather to increase complexity to see how the SDN controller is able to sort out the loops in its favour and if thereis any effect on the performance of the network. The purpose of this simulation isto see that the controller can handle the topology when it is a bitmore complex as in Figure 20.

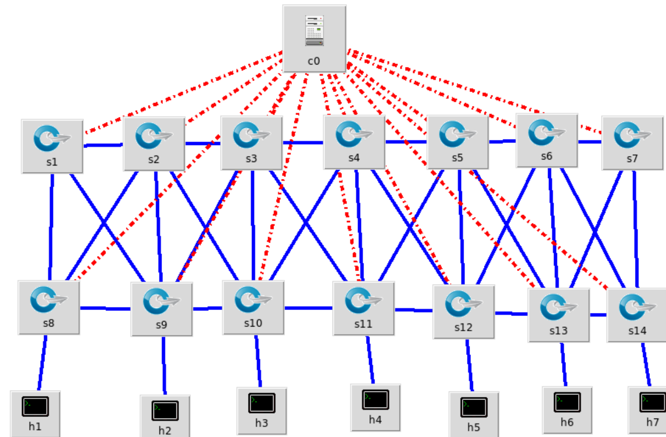


Figure 20: Larger Number of Nodes

If a larger network needs to be simulated this is better doneusing a Python script that automatically generates more nodes.

3.8.4 Fourth Scenario using Utilizing Flows [Appendix I]

SDN controller in this scenario using POX controller with [Appendix I], the idea that we can also be used as a router when defined by proper flows like in the topology in Figure 21, This simulates what routing does in a traditional network.

Defining a whole routing table this way would be extremely laborious but for the scope of this thesis this is enough to see that the functionality is there.

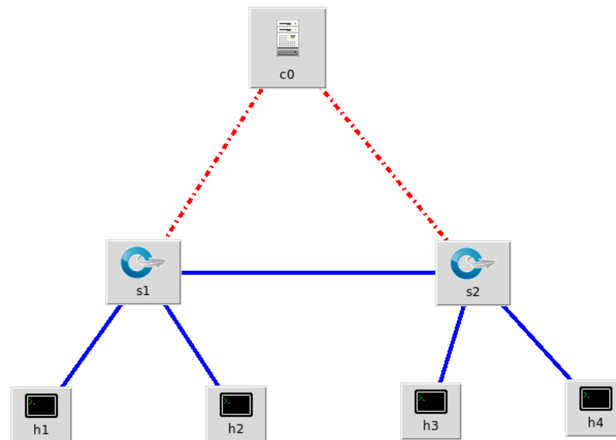


Figure 21: SDN controller works as router

3.8.4.1Running the POX

Start POX by running the pox.py program, and specifying the POX components to use. For example, to run POX so it makes the switches it controls emulate the behaviour of Ethernet learning switches, run the command:

mininet@mininet-vm: ~$ sudo ~/pox/pox.py forwarding.l2_learning

3.8.4.2Script Explanation (ImportantParts)

c1 = net.addController('c1', controller=RemoteController, ip="192.168.45.142", port=6633)

A remote controller c1 is defined to be found at IP address 192.168.45.142 port 6633 that is the VM NIC's IP address.

```
s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
```

Switch 1 named s1 is defined as an OVSKernelSwitch, Open VSwitch type ofSDN switch.

```
h1 = net.addHost('h1', cls=Host, ip='10.0.0.1',
mac='10:00:00:01:00:00', defaultRoute=None)
```

Host 1 named h1 is added and given the IP 10.0.0.1 and the MAC 10:00:00:01:00:00 to make easier to manage.

```
s1.linkTo( h1 )
```

A link between h1 and s1 is created. If no other parameters are defined it will be a “perfect” link with no delay or loss and with bandwidth only limited by thehardware. the simulation is running on

```
net.build()
c1.start()
```

The switch s1 is set to be controlled by the controller c1. Note that in Mininet the switches are connected to the controller this way and not via “physical” links.

3.8.4.3First StepInstall POX Controller

By these two commands as in Figure 22.

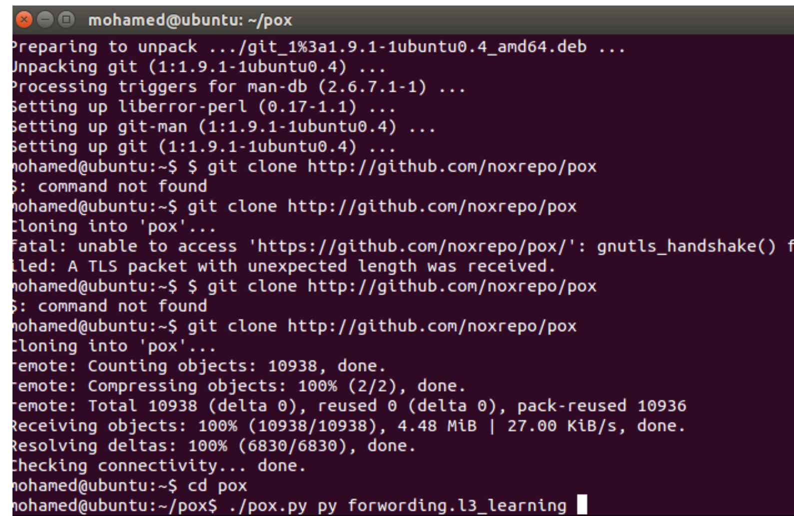


Figure 22: install POX controller in ubuntu

git clone http://github.com/noxrepo/pox

cd pox

3.8.4.4SecondStepRun the Script [Appendix I]

As in Figure 23, we run the python code that describe the topology

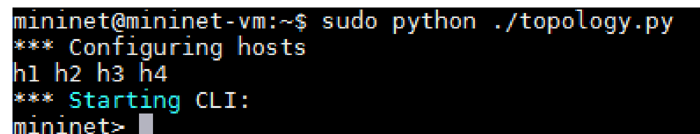


Figure 23: run the python script

The subnets 10.0.0.0/8 (h1,h2 and h3) and 11.0.0.0/8 (h4) are not able to ping each other because these IP addresses in different LANs as in Figure 24.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 X
h2 -> h1 h3 X
h3 -> h1 h2 X
h4 -> X X X
*** Results: 50% dropped (6/12 received)
```

Figure 24: testing the reachability

3.8.4.5ThirdStep add Feature to Learn Layer 3 Routing

By using command (Forwording.l3_learning)as in Figure 25, we can add feature to makes hosts in different subnets reach each other's and to prepare the controller to receive layer 3 in another word make the controller works as a router.

```
mohamed@ubuntu:~$ cd pox
mohamed@ubuntu:~/pox$ ./pox.py py forwarding.l3_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
Ready.
POX>
```

Figure 25: add feature to POX controller

3.8.4.6FourthStep add Default Routing in Mininet Software

We need to add the command (h1 route add -net default h1-eth0) as default routing to each host as in Figure 26.

```
mininet> h1 route add -net default h1-eth0
mininet> h2 route add -net default h2-eth0
mininet> h3 route add -net default h3-eth0
mininet> h4 route add -net default h4-eth0
```

Figure 26: add default routing

3.8.4.7FifthStepPing all Hosts

After that we ping to h4 and we can reach it as in Figure 27.

```
mininet@mininet-vm:~$ sudo python ./topology.py
*** Configuring hosts
h1 h2 h3 h4
*** Starting CLI:
mininet> h1 ping -c3 h4
connect: Network is unreachable
mininet> h1 route add -net default h1-eth0
mininet> h2 route add -net default h2-eth0
mininet>  h3 route add -net default h3-eth0
mininet>  h4 route add -net default h4-eth0
mininet>
mininet> h1 ping -c3 h4
PING 11.0.0.4 (11.0.0.4) 56(84) bytes of data.
64 bytes from 11.0.0.4: icmp_seq=1 ttl=64 time=20.6 ms
64 bytes from 11.0.0.4: icmp_seq=2 ttl=64 time=1.78 ms
64 bytes from 11.0.0.4: icmp_seq=3 ttl=64 time=0.080 ms

--- 11.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.080/7.510/20.669/9.330 ms
mininet>
```

Figure 27: reach the h4 host

We use pingall command to be sure we can reach all hosts as presented in Figure 28.

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

Figure 28: testing the network

Chapter Four:

Results and Discussions

`4. Results and Discussions

4.1 Overview

In this chapter, we testing the topologies that we mention it in chapter 3 and analysis it to see if we reach the goal of our thesis that we bring and test the SDN Basic Function compare to the Traditional networksand the way to transition to SDN network.

4.2 Simulation OneLinear network

Mininet is run from aMobaXterm_Personal_9.0 window and it needs to be run as the root user.

ssh mininet@192.168.45.144
mininet@mininet-vm:~/mininet/topo$ sudo mn --
controller=remote,ip=192.168.45.142 --topo=linear,2

This command run the topology and the controller has not been started yet,as we see there is no connectivity between thehosts.

mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

OpenDayLight controller is run from Ubuntu terminal as in Figure 29.

mohamed@ubuntu:~$ cd odl/bin

mohamed@ubuntu:~/odl/bin$ ./karaf -of13

opendaylight-user@root>feature:install

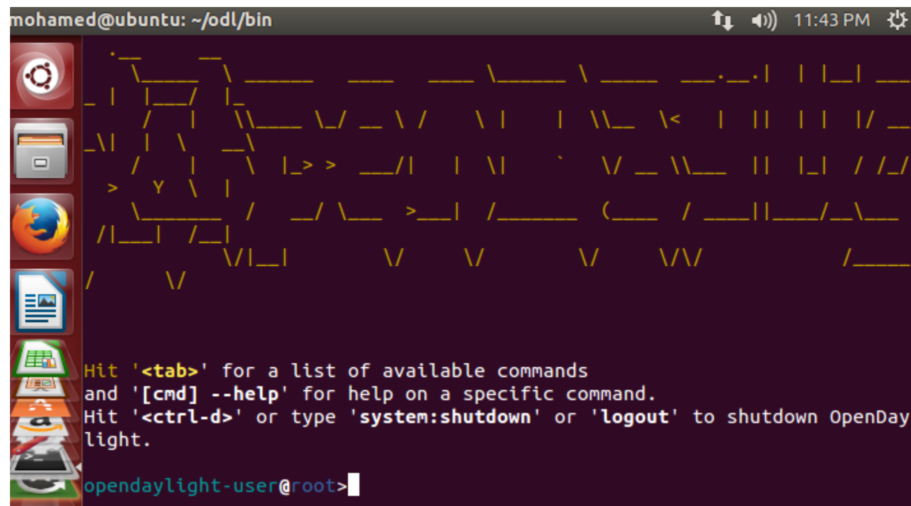


Figure 29: start Open daylight controller

From the Open DaylightGUI, the topology does not yet include anything elseexcept the switches as in Figure 30.

Figure 30: two switches in ODL controller

4.2.1ConnectivityBetween Hosts

We use pingall command to ping to all host, or ping from host 1 to host 2 as presented in Figure 31.

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.158 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.391 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.551 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.442 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.391 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.390 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.358 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.388 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.578 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.394 ms
```

Figure 31: Test connectivity

After the hosts have send some traffic into the network the controller is able to seethem as in Figure 32.

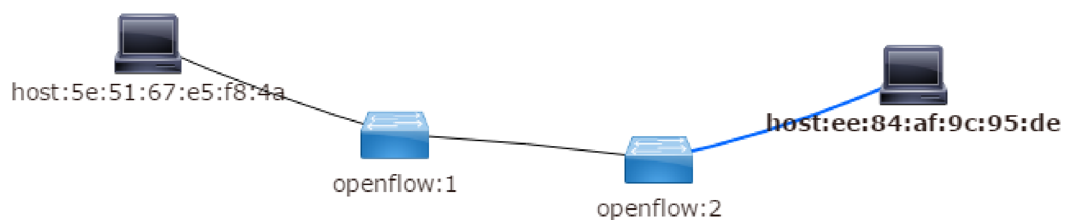


Figure 32: topology in ODL controller after ping

By running the Wireshark by using command (sudo wireshark &) in mininet software to see and capture the traffic details as in Figure 33.

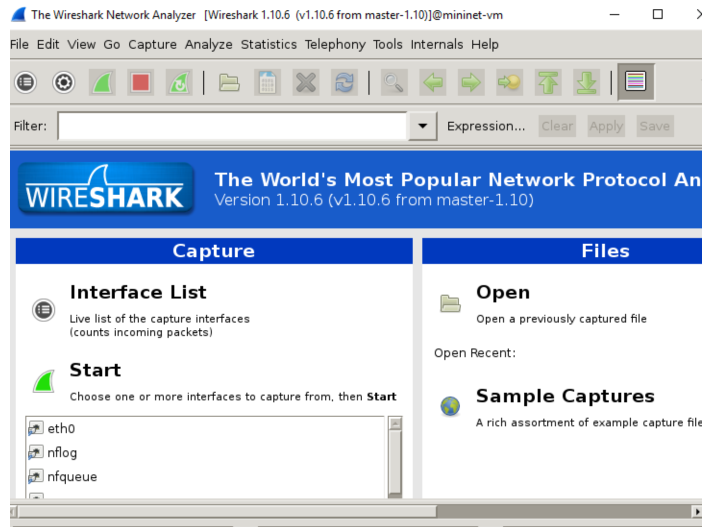


Figure 33: start Wireshark

Capture the traffic using Wireshark reveals how the controller first connectswith the switches and inquiries about their capabilities as seen in the capture inFigure34.

Filter: of   Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
309	1.484818000	ee:e7:e4:4d:46:e7	CayeeCom_00:00:01	LLDP + OF	284	Chassis Id = 00:00:00:00:00:02
310	1.485628000	c2:0f:da:97:bf:15	CayeeCom_00:00:01	OF 1.0	169	of_packet_in
311	1.485866000	c6:57:6d:e4:88:dd	CayeeCom_00:00:01	LLDP + OF	284	Chassis Id = 00:00:00:00:00:01
312	1.486498000	c6:57:6d:e4:88:dd	CayeeCom_00:00:01	OF 1.0	169	of_packet_in
446	2.875432000	192.168.45.142	192.168.45.144	OF 1.0	122	of_flow_stats_request
448	2.875992000	192.168.45.144	192.168.45.142	OF 1.0	462	of_flow_stats_reply
450	2.889913000	192.168.45.142	192.168.45.144	OF 1.0	86	of_port_stats_request
451	2.890533000	192.168.45.144	192.168.45.142	OF 1.0	390	of_port_stats_reply
452	2.899036000	192.168.45.142	192.168.45.144	OF 1.0	86	of_queue_stats_request
453	2.899473000	192.168.45.144	192.168.45.142	OF 1.0	78	of_queue_stats_reply
454	2.906613000	192.168.45.142	192.168.45.144	OF 1.0	78	of_table_stats_request
462	2.908955000	192.168.45.144	192.168.45.142	OF 1.0	406	of_table_stats_reply
464	2.923146000	192.168.45.142	192.168.45.144	OF 1.0	122	of_flow_stats_request
465	2.923350000	192.168.45.144	192.168.45.142	OF 1.0	462	of_flow_stats_reply
467	2.930576000	192.168.45.142	192.168.45.144	OF 1.0	86	of_port_stats_request
468	2.930813000	192.168.45.144	192.168.45.142	OF 1.0	390	of_port_stats_reply
469	2.933903000	192.168.45.142	192.168.45.144	OF 1.0	86	of_queue_stats_request
470	2.934067000	192.168.45.144	192.168.45.142	OF 1.0	78	of_queue_stats_reply
471	2.938064000	192.168.45.142	192.168.45.144	OF 1.0	78	of_table_stats_request

Frame 309: 284 bytes on wire (2272 bits), 284 bytes captured (2272 bits) on interface 0
Ethernet II, Src: Vmware_d6:c7:ea (00:0c:29:d6:c7:ea), Dst: Vmware_0d:20:10 (00:0c:29:0d:20:10)
Internet Protocol Version 4, Src: 192.168.45.142 (192.168.45.142), Dst: 192.168.45.144 (192.168.45.144)
Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 43265 (43265), Seq: 1, Ack: 1, Len: 218
OpenFlow

Figure 34: start Wireshark capture

When the hosts start communicating at first the controller transmits the packetswithin Open Flow packets and ARP packets as seen in the Wireshark capture in Figure 35 and Figure 36.

1749	8.949574000	Vmware_0d:20:10	Vmware_f5:d0:87	ARP	42 Who has 192.168.45.2? Tell 192.168.45.144
1750	8.949852000	Vmware_f5:d0:87	Vmware_0d:20:10	ARP	60 192.168.45.2 is at 00:50:56:f5:d0:87

Figure 35: ARP within Open flow

241	6.763986000	10.0.0.1	10.0.0.2	OFP+ICMP	184 Packet In (AM) (116B) => Echo (ping) request
245	6.764120000	10.0.0.2	10.0.0.1	OFP+ICMP	184 Packet In (AM) (116B) => Echo (ping) reply

Figure 36: ICMP within open flow

Later on, the hosts can ping each other without the need for the controller to Interfere by using OpenFlow protocol as seen in the Wireshark capture in Figure 37

Filter: icmp Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
2407	21.54387400(	10.0.0.2	10.0.0.1	OF 1.3	206	of_packet_in
2408	21.54397100(	10.0.0.1	10.0.0.2	OF 1.3	206	of_packet_in

Figure 37: open flow 1.3

From the GUI,we notice that how much traffic has gone through a node as in Figure 38, also as we saw the node traffic statistics sending packets, receiving packets and Drops packs.

Node Connector Statistics for Node Id - openflow:2

Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
openflow:2:1	14	106	1260	9080	0	0	0	0	0	0	0	0
openflow:2:2	106	106	9080	9080	0	0	0	0	0	0	0	0
openflow:2:LOCAL	0	0	0	0	0	0	0	0	0	0	0	0

Figure 38: Node Traffic statistics

4.2.2 Performance and Bandwidth

Performance of the network between two hosts can be measured using iperf.

Connectivity between the hosts has been established using the switches controlled by Open Daylight andthe bandwidth between h1 and h2 is 234Mbits/sec as in Figure 39.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['234 Mbits/sec', '236 Mbits/sec']
```

Figure 39: Bandwidth with host 1 and host 2

4.3Simulation Two Looped Topology

The switches send LLDP (Link LayerDiscovery Protocol) packets over Open Flow to sort the topology out as in Figure 40.

```
8525 120.2481440( 3a:1b:57:7b:bf:dc CayeeCom_00:00:01 OF 1.3 193 of_packet_in
8526 120.2482050( 5e:2e:00:59:3f:fb CayeeCom_00:00:01 OF 1.3 193 of_packet_in
8527 120.2482380( ea:75:1d:50:b4:2f CayeeCom_00:00:01 OF 1.3 193 of_packet_in
8528 120.2484570( 5a:03:00:ad:73:d5 CayeeCom_00:00:01 LLDP + OF 566 Chassis Id = 00:00:00:00:00:03 Port Id = 3 TTL = 4919 System Name = openflow:3 + of_packet_out
8529 120.2484900( 192.168.45.142 192.168.45.144 TCP 66 6633 > 43275 [ACK] Seq=2913 Ack=32495 Win=122368 Len=0 TSval=393752 TSecr=399480
8530 120.2486560( ea:9f:e9:56:22:21 CayeeCom_00:00:01 OF 1.3 193 of_packet_in
8531 120.2487090( 192.168.45.142 192.168.45.144 TCP 66 6633 > 43274 [ACK] Seq=3565 Ack=33015 Win=126976 Len=0 TSval=393752 TSecr=399480
8532 120.2487140( 192.168.45.142 192.168.45.144 TCP 66 6633 > 43276 [ACK] Seq=2865 Ack=25651 Win=104448 Len=0 TSval=393752 TSecr=399480
8533 120.2487360( 5a:03:00:ad:73:d5 CayeeCom_00:00:01 OF 1.3 193 of_packet_in
8534 120.2487950( 9a:37:bf:4e:cf:a0 CayeeCom_00:00:01 OF 1.3 193 of_packet_in
8535 120.2490860( 42:5e:e0:45:28:fc CayeeCom_00:00:01 LLDP + OF 566 Chassis Id = 00:00:00:00:00:04 Port Id = 1 TTL = 4919 System Name = openflow:4 + of_packet_out
8536 120.2490950( 192.168.45.142 192.168.45.144 TCP 66 6633 > 43276 [ACK] Seq=2865 Ack=25778 Win=104448 Len=0 TSval=393752 TSecr=399480
8537 120.2493550( 9a:03:6c:ae:e7:d8 CayeeCom_00:00:01 OF 1.3 193 of_packet_in
8538 120.2494360( b6:a6:34:f3:c9:96 CayeeCom_00:00:01 OF 1.3 193 of_packet_in
8539 120.2494990( 96:81:dd:e4:5e:71 CayeeCom_00:00:01 OF 1.3 193 of_packet_in
8540 120.2498400( 192.168.45.142 192.168.45.144 TCP 66 6633 > 43274 [ACK] Seq=3565 Ack=33142 Win=126976 Len=0 TSval=393752 TSecr=399481
8541 120.2498490( 192.168.45.142 192.168.45.144 TCP 66 6633 > 43276 [ACK] Seq=2865 Ack=25905 Win=104448 Len=0 TSval=393752 TSecr=399481
8542 120.2498520( fe:fb:78:4b:31:2e CayeeCom_00:00:01 LLDP + OF 566 Chassis Id = 00:00:00:00:00:02 Port Id = 3 TTL = 4919 System Name = openflow:2 + of_packet_out
8543 120.2500790( 1a:eb:77:55:53:f2 CayeeCom_00:00:01 OF 1.3 193 of_packet_in
```

Figure 40: wireshark with looped topology

After the topology is sorted there is connectivity between all hosts before starting sending any traffic, The GUItopology viewer shows the connections between the nodes as in Figure 41.

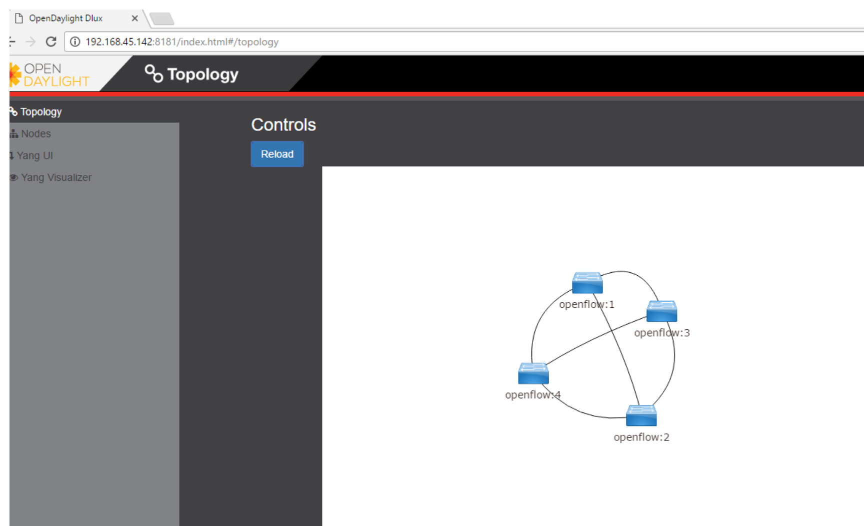


Figure 41: Looped topology in ODL Controller

After sending some traffic the topology is looks like Figure 42.

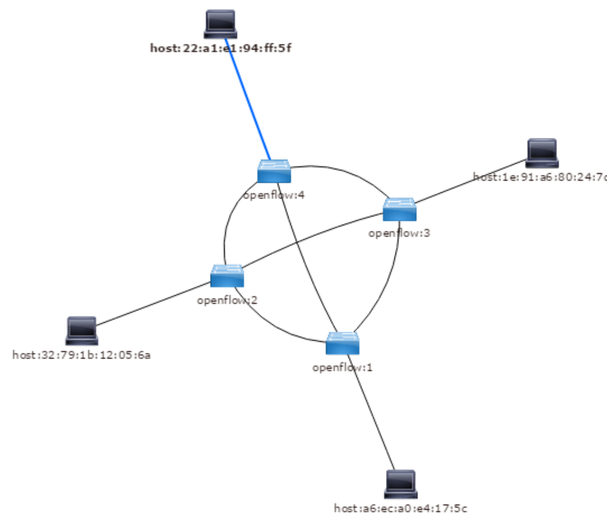


Figure 42: Looped topology in controller after send some traffic

4.3.1 Connectivity Between Hosts

Testing the connectivity between h1 and h4 as showing in Figure 43.

```
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.407 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.419 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.525 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.502 ms
```

Figure 43: connectivity testing for looped topology

4.3.2 Performance and Bandwidth

The bandwidth between h1 and h2 is 193Mbits/sec as in Figure 44.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['193 Mbits/sec', '198 Mbits/sec']
```

Figure 44: Testing bandwidth between h1 and h2 for looped topology

And Between h1 and h4 is 185Mbits/sec as in Figure 45.

```
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['185 Mbits/sec', '189 Mbits/sec']
```

Figure 45: Testing bandwidth between h1 and h4 for looped topology

4.4Simulation Three A Larger Number of Nodes

Starting the version 3 topology with OpenDayLight Helium resulted in the systemusing 100% CPU with large number of switches and hosts.

4.4.1 Connectivity Between Hosts

By using pingall command to test the connectivity between all hosts as in Figure 46.

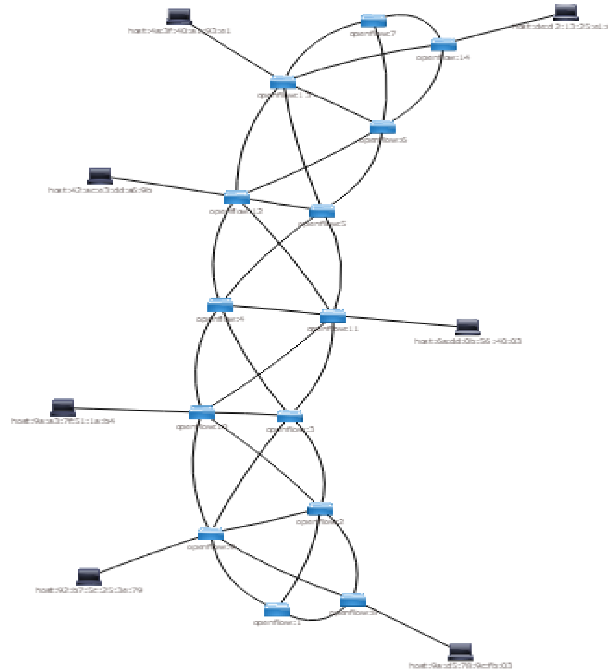


Figure 46: large number of nodes topology in ODL controller

4.4.2 Performance and Bandwidth

By Testing the bandwidth and performance using iperf between h1 h5 we presented results as 117Mbits/sec as in Figure 47.

```
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['117 Mbits/sec', '122 Mbits/sec']
```

Figure 47: bandwidth between h1 and h5 for large number of nodes

And between h1 and h7 as in Figure 48.

```
mininet> iperf h1 h7
*** Iperf: testing TCP bandwidth between h1 and h7
*** Results: ['114 Mbits/sec', '119 Mbits/sec']
```

Figure 48: bandwidth between h1 and h7 for large number of nodes

The node connector statistics for node id in open daylight as in Figure 49.

Node Connector Statistics for Node Id - openflow:12

Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
openflow:12:5	66	67	5610	5829	1	0	0	0	0	0	0	0
openflow:12:6	26012	279598	1735160	322062863	0	0	0	0	0	0	0	0
openflow:12:3	66	67	5610	5829	1	0	0	0	0	0	0	0
openflow:12:4	67	67	5695	5829	0	0	0	0	0	0	0	0
openflow:12:1	24609	281067	1640580	322163185	1	0	0	0	0	0	0	0
openflow:12:2	255054	50622	320427938	3375827	1	0	0	0	0	0	0	0
openflow:12:LOCAL	0	0	0	0	0	0	0	0	0	0	0	0

Figure 49: The node connector statistics in controller in large node topology

The open flow captureWiresharkas in Figure 50,

```
164 0.201978000 192.168.45.142    192.168.45.144    OF 1.3    82 of_group_features_stats_request
165 0.202345000 192.168.45.144    192.168.45.142    OF 1.3    94 of_bad_request_error_msg
166 0.203544000 192.168.45.142    192.168.45.144    OF 1.3    90 of_meter_config_stats_request
167 0.203948000 192.168.45.144    192.168.45.142    OF 1.3    82 of_meter_config_stats_reply
168 0.205040000 192.168.45.142    192.168.45.144    OF 1.3    90 of_meter_stats_request
169 0.205480000 192.168.45.144    192.168.45.142    OF 1.3    82 of_meter_stats_reply
170 0.206543000 192.168.45.142    192.168.45.144    OF 1.3   122 of_flow_stats_request
171 0.206891000 192.168.45.144    192.168.45.142    OF 1.3   722 of_flow_stats_reply
```

Figure 50: Open flow in Wireshark for large node topology

As we using version 1.3 this is the latest version of open flow as showing the details in figure 51.

```
▷ Frame 505: 320 bytes on wire (2560 bits), 320 bytes captured (2560 bits) on interface 0
▽ Ethernet II, Src: Vmware_d6:c7:ea (00:0c:29:d6:c7:ea), Dst: Vmware_0d:20:10 (00:0c:29:0d:20:10)
  ▷ Destination: Vmware_0d:20:10 (00:0c:29:0d:20:10)
  ▷ Source: Vmware_d6:c7:ea (00:0c:29:d6:c7:ea)
    Type: IP (0x0800)
▷ Internet Protocol Version 4, Src: 192.168.45.142 (192.168.45.142), Dst: 192.168.45.144 (192.168.45.144)
▷ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 43296 (43296), Seq: 185, Ack: 7645, Len: 254
▽ OpenFlow
    version: 4
    type: OFPT_PACKET_OUT (13)
    length: 127
    xid: 217
    buffer_id: 4294967295
    in_port: 4294967293
    actions_len: 16
  ▽ of_action list
    ▽ of_action_output
        type: OFPAT_OUTPUT (0)
        len: 16
        port: 2
        max_len: 65535
  ▽ Ethernet packet
    ▽ Ethernet II, Src: 56:96:47:92:24:31 (56:96:47:92:24:31), Dst: CayeeCom_00:00:01 (01:23:00:00:00:01)
      ▷ Destination: CayeeCom_00:00:01 (01:23:00:00:00:01)
      ▷ Source: 56:96:47:92:24:31 (56:96:47:92:24:31)
        Type: 802.1 Link Layer Discovery Protocol (LLDP) (0x88cc)
    ▽ Link Layer Discovery Protocol
      ▷ Chassis Subtype = MAC address, Id: 00:00:00:00:00:0b
      ▷ Port Subtype = Locally assigned, Id: 2
      ▷ Time To Live = 4919 sec
      ▶ System Name = openflow:11
      ▷ Stanford - Unknown (0)
      ▷ Stanford - Unknown (1)
      ▷ End of LLDPDU
```

Figure 51: The details of open flow for large node topology

Switching back to Open DaylightHelium allowed the use of this topology. Asthe features required of the controller for these simulations are the same in bothHydrogen, Helium and POX controllers.

4.5Simulation Four Utilizing Flows [Appendix I]

The subnets 10.0.0.0/8 and 11.0.0.0/8 are not able to ping each other.
To make the switch s1 route traffic between h1 and h4 the controller will installflows on it. These flows will do the following:
Flood ARP packets in the network to allow the hosts to find the router IP.
Using manually defined flows on the controller routing between hosts in differentsubnets were achieved. Doing it by using POX controller this way on any greater scale would be very labourintensive.

However, this shows some of the capabilities of the SDN controller byutilizing flows to do something that is done by more intelligent devices intraditional networks.

4.5.1 Connectivity Between Hosts

First, we run the code by using scriptas in Figure 52 and then we add the default route as mention in chapter 3, after that we test the connectivity by ping to the host 4 (in different subnet hot) as showing in figure 54, so this mean that the controller is work as router in traditional network.

```
mininet@mininet-vm:~$ sudo python ./topology.py
*** Configuring hosts
h1 h2 h3 h4
*** Starting CLI:
mininet>
```

Figure 52: starting the code in mininet software

As in Figure 53 the controller is work as a router in traditional networks.

```
mininet> h1 ping h4
PING 11.0.0.4 (11.0.0.4) 56(84) bytes of data.
64 bytes from 11.0.0.4: icmp_seq=1 ttl=64 time=0.029 ms
64 bytes from 11.0.0.4: icmp_seq=2 ttl=64 time=0.076 ms
64 bytes from 11.0.0.4: icmp_seq=3 ttl=64 time=0.063 ms
64 bytes from 11.0.0.4: icmp_seq=4 ttl=64 time=0.074 ms
64 bytes from 11.0.0.4: icmp_seq=5 ttl=64 time=0.080 ms
```

Figure 53: Testing connectivity between h1 and h4

4.5.2 Performance and Bandwidth

After testing the connectivity and be sure that the functionality is working fine, then we testing the performance and bandwidth between hosts

by using iperf command as in Figure 54,in our testing we make unlimited bandwidth to test the maximum bandwidth.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['27.3 Gbits/sec', '27.3 Gbits/sec']
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['23.6 Gbits/sec', '23.7 Gbits/sec']
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['24.0 Gbits/sec', '24.0 Gbits/sec']
mininet> iperf h2 h4
*** Iperf: testing TCP bandwidth between h2 and h4
*** Results: ['23.8 Gbits/sec', '23.9 Gbits/sec']
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['23.9 Gbits/sec', '23.9 Gbits/sec']
mininet> iperf h3 h4
*** Iperf: testing TCP bandwidth between h3 and h4
*** Results: ['23.7 Gbits/sec', '23.8 Gbits/sec']
```

Figure 54: bandwidth between hosts in utilization topology

4.6Analysis The Results

The requirements for the functionalities of the current network are not complex,only basic switching and routing are required. These were simulated with thedifferent topologies.

The Open daylight controller was able to perform switching with the included L2 switch module with good performance in the first three simulations.

However, the controller, Pox, was implemented routing with the use of flows in the fourth simulation.Compared to traditional networking this required more configuration the way it isnow implemented on the controller, almost like manual packet handling.

Of course, this cannot be the way to do it in real life applications but it shows thatinstead of routing defining flows can be used to manipulate the traffic.

4.7Transitioning to SDN

Software-Defined Networking (SDN) has become one of the hottest topics in the industry, and for good reason, given the transformative changes that it can bring to many segments across IT, datacentre, and carrier markets.

First of all, it should be determined whether there is any particular reason to Transform the current network to a SDN network in any timespan. The value of SDN resides in its powerful abstractions.

The main problem of the current network is the complicated management spreadin many places and done in many ways. For this SDN can, at the moment, onlyhelp by centralizing the control of the core network. For the access network SDNis not a viable alternative yet, the focus of the technology has not been inprovisioning customer lines.

Chapter Five:

Conclusion and Recommendations

5. Conclusion and Recommendations

5.1 Conclusion

SDN begin implementing is not straight forward as the preparation for the simulations proved. The huge amount of available SDN controller software and the little amount available SDN switch hardware makes it difficult to do testing.

The simulations run showed that the basic functionality needed is there and the SDN concept works, but real-life performance testing could not beconducted in the scope of this thesis. The control of the whole network iscentralized to the controller but learning to configure the controller is another challenge for the users.

The practical side of doing an actual transition to SDN should be documented. The idea of the transition is fairly simple but in practice how does one go aboutdoing it, what needs to be taken in consideration so the network remains stableand available through the process.

5.2 Recommendations

In future work, next points explain briefly what are planned:

The first suggestion is to introduce reality into the scenarios. In this research, all the simulations have been run in static scenarios with one, two, three and four designs. It would be interesting to study the performance of the same parameters varying the reality of dynamic networks.

The second suggestion is to really get SDN into the networking community the interoperability of all SDN components must be assured by figuring out the inconsistencies between different developers; vendor locked SDN is not

true SDN asthe requirements of basic switching and routing were fulfilled with the additional functionality of resolving loops. The current network could be carried out using Open Daylight controller, but a more developed way to define flows would allow for a more user-friendly way to manage a larger number of nodes in a network.

References

[1] Steve Goeringer ,“Software-Defined Networking: The New Norm for Networks,” Open Networking Foundation White paper, retrieved on 2015 Polar Star Consulting, LLC.

[2] Chang-Gyu LIM, Soo-Myung PAHK, Young-Hwa KIM “Model of Transport SDN and MPLS-TP for T-SDN Controller” ETRI (Electronics and Telecommunications Research Institute), Daejeon, Korea Jan. 31 ~ Feb. 3, 2016 ICACT2016

[3] FarisKeti andShavanAskar"Emulation of Software Defined Networks Using Mininet in Different Simulation Environments "2015 6th International Conference on Intelligent Systems, Modelling and Simulation Electrical and Computer Engineering Department Faculty of Engineering Duhok-Kurdistan Region,Iraq.

[4] Wenfeng Xia, Yonggang Wen, ChuanHengFoh, DusitNiyato, and HaiyongXie,“A Survey on Software-Defined Networking” IEEE COMMUNICATION SURVEYS & TUTORIALS, VOL. 17, NO. 1, FIRST QUARTER 2015.

[5] Foukas, X., Marina, M.K. &Kontovasilis, K. 2014. Software Defined Networks Concepts. Liyanage, M., Gurtov, A. &Ylianttila, M. Software Defined Mobile Networks (SDMN): Beyond LTE Network Architecture. Wiley.

[6] Jammal, M. et al. Software defined networking: State of the art and research challenges. Computer Networks2014.

[7] de Oliveira, R.L.S., Schweitzer, C.M., Shinoda, A.A. &Prete, L.R. 2014. Using Mininet for Emulation and Prototyping Software-Defined

Networks. 2014 IEEE Colombian Conference on Communications and Computing (COLCOM).

[8] Brianlink, 2015. [Online].Refrencehttp://www.brianlinkletter.com/using-the-pox-sdn-controller/https://iperf.fr/

[9] In May 2006https://en.wikipedia.org/wiki/Wireshark

[10]Opennetwork,2015.[Online].https://www.opennetworking.org/?p=1492&option=com_wordpress&Itemid=316

[11]Foster, N., Guha, A., Reitblatt, M., Story, A., Freedman, M., Katta, N., Monsanto, C., Reich, J,Rexford, J., Schlesinger, C., Walker, D., Harrison, R.: Languages for software-defined networks. Communications Magazine, IEEE 51(2), 128–134 (2013)

[12]Rfwireless,2013.[Online]http://www.rfwirelessworld.com/Terminology/traditional-networking-vs-software-defined-networking.html

[13] Foundation, O.N.: Openflow switch specification version 1.3.1. Tech. rep., Open Networking Foundation (September 2012)

[14]OpenSDN,2015.[Online]https://www.opennetworking.org/sdn resources/sdn-definition

[15] Aryaka, 2014. [Online]http://www.aryaka.com/blog/why-sdn-concepts-need-to-extend-into-the-wan/

[16]Marist,C.: What is Avior? (2012), http://openflow.marist.edu/avior.html

[17]Muntaner, G.: Evaluation of OpenFlow Controllers. Master's thesis (October 2012)

[18]Stefano Vissicchio_ Laurent Vanbever Olivier Bonaventure Universitecatholique de Louvain Princeton University Universitecatholique de Louvain stefano.April 2014. Present paper about "Opportunities and Research Challenges of Hybrid Software Defined Networks"

[19]OpenDaylight,2015.[Online]https://www.opendaylight.org/software/downloads/helium

[20]D. Kreutz, F. M. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodol-

molky, and S. Uhlig, \Software-de_ned networking: A comprehensive survey," proceedings of the IEEE, vol. 103, no. 1, pp. 14{76, 2015.

[21]Brian,2014. [Online]http://www.brianlinkletter.com/how-to-use-miniedit-mininets graphical-user-interface/

[22]Mininet, 2015. [Online]http://mininet.org/overview/

[23]Github,2014.[Online]https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

Appendix I

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, OVSKernelSwitch, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info

def emptyNet():

   net = Mininet(controller=RemoteController, switch=OVSKernelSwitch)

   c1 = net.addController('c1', controller=RemoteController, ip="127.0.0.1",
port=6633)

   h1 = net.addHost( 'h1', ip='10.0.0.1' )
   h2 = net.addHost( 'h2', ip='10.0.0.2' )
   h3 = net.addHost( 'h3', ip='10.0.0.3' )
   h4 = net.addHost( 'h4', ip='11.0.0.4' )

   s1 = net.addSwitch( 's1' )
   s2 = net.addSwitch( 's2' )

   s1.linkTo( h1 )
   s1.linkTo( h2 )
   s2.linkTo( h3 )
   s2.linkTo( h4 )
```

```
    s1.linkTo( s2 )

    net.build()
    c1.start()
    s1.start([c1])
    s2.start([c1])

    CLI( net )
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    emptyNet()
```