Republic of Yemen

University of Science & Technology-Yemen

Faculty of Computing and IT

Computer Sciences Department

Computer Networks Program

Migrating UST Network To SDN and NFV

Done by:

Alfudhail Hassan Alsharafi

Ibraheem Abdullah Morghim

Mohammed Abdulraqeb Asham

Supervised by:

Dr. Yasser Aldhamary

This project has been built as a part of requirements for the award of Bachelor's degree of Computer Sciences - Computer Networks Program for the year 2017/2018

Acknowledgement

First ,we would thank our God for his help and blessing during our work and lives . Also, We would thank our parents for their help and encouragement . We would thank our supervisor Dr. Yasser Aldamary and head our department Dr.Belal Alfuhaidi . Also we want to thank all people who encouraged or helped us directly or undirectly during our work:

- Dr.Ammar Alzahary
- Dr.Waleed Shaher
- Eng. Mohammed Alsultan
- Eng.Ameen Alfaqeeh
- Eng.Mohammed Alhakeem
- Eng.Hassan Assaad
- Eng.Mobarak Alamery
- Eng.Mohammed Gayash
- Eng. Majed Altwayti
- Eng. Mohammed Aldafaee

Abstract

To enable programmability ,increase agility ,reduce hardware and reduce costs , SDN and NFV are the most trend of networking world today . Migrating UST network to SDN and NFV and integrate between them at the same network were the main aims of this project . A design is proposed to implement migration to SDN and NFV for the UST network .We provided a recommended plan for migration to SDN then we implemented it in a an emulator program , Then we integrated NFV with SDN in UST Network . We compare between both , the emulated exiting the network and the emulated SDN network . Finally , we got that SDN is better in RTT and Throughput . Also , SDN with NFV reduce cost .

Table of Contents

List of figure

Chapter one
Introduction

1.1 Introduction

Generally, the beginning of enterprises and companies is limited , then they grow step by step and day by day . This growth includes increasing of employees , offices, buildings, branches and the network that connect all these objects . So, the network of enterprises and companies grows dramatically , when they grow . Network scaling means that traffic of data is increasing while the number of employees and services are added to the company , this traffic is going to be huge more than the network can handle .Actually , this really well affect the efficiency of the network and limit the bandwidth , so new network devices , new network administrators and engineers must be brought to the company which means more costs and very high complexity.

In addition , Routing and switching functions are going to be complex , and more planning , management , services ,and protocols should be implemented. The type of service and protocol well force to bay a network device with a specific version from a specific vender. So , network companies well monopolize some services and compatibility between devices.

To solve these complexities and more , very new technologies has come up to networks world. Software Defined Network (SDN) and Network Function Virtualization (NFV) .

The main idea of SDN is to split the data plan from control plan , data plan well be located in a switch hardware (openflow supported switch) , and control plan is located in a controller device .[1] while NFV is to make network devices virtualized .

The controller well control the switches by implementing scripts (instead of configrations) programed by a network programmer (instead of network admin) . So , SDN well let the network to be programmable and enabling to invent protocols and technologies instead of being monopolized by venders. In this project , we are going to migrate the network of University of Sciences and Technology (UST) to SDN and NFV network .

1.2 Problem Statement

Nowadays , a lot of problems and difficulties face traditional networks . In our visit to the data center (Network department.) of UST , we observed that some problems face this network as any other traditional network. These problems include :

1.2.1 Venders dependence:

Some services and protocols force to buy from a specific venders [2][3] .

1.2.2 Network congealment:

Services and protocols are embedded into network devices , so to use a service or a protocol that network devices does not support , a new network device that support this service must be bought[4].

1.2.3 Administration complexity:

it is so hard to configure every single network device independently .

1.2.4 Troubleshooting complexity:

Digging where is the problem is also a real difficulty .

1.3 Goals

1 – Implementing migration to SDN and NFV

2 - Understanding these very new technologies and sharpening our skills on them

3 - Proving how network can be controlled and virtualized by SDN and NFV

4 - Extract a general plan and steps to migrate any network to SDN

5- Applying our experiences and skills (in networks and programing) that we have learned at our studying years to use them in a new field

1.4 Scope

The geographical and functional scope are going to be defined as follows:

1.4.1 Geographical Scope:

The network of UST at the main branch

1.4.2 Functional scope:

1- Migrating the whole network of UST at main branch to SDN

2- Migrating to NFV

3 – Integrate SDN with NFV at the same topology

4- Implement some scenarios that are required from the network

1.5 Tools

1 – OpenDayLight (ODL) controller:

Is the controller that well control the network.

2 – Python 3.0

To write programs and scripts that well control the network.

3- Mininet:

Emulation platform to emulate SDN inside.

4- OpenStack

To implement NFV.

5 – MikroTik RouterOS

To implement NFV.

6-VMware

To run virtual machines.

7 – Ubunto Server (Linux)

To run mininit and Vmware inside.

8 – computers (Laptops)

1.6 Project Methodology

To complete this project perfectly , we chose a set of steps as a project methodology :

1 – Learn SDN and NFV technologies and thier importance.

2 – Learn the tools used in this project including emulation and virtualization tools .

3 – collecting information about the project.

4 – Going to the data center of the UST network and meeting its engineers.

5 - Analysis the starting UST network.

6 – studying the need of SDN and NFV for UST network and analysis project requirements .

7 - provide the supposed design for SDN and NFV network .

8 – implement the SDN and NFV

9 - Test reachability on the new network.

9 – Test the networks compare performance between the old and new networks.

1.7 Project organization

We are going to start with a general introduction and objectives of this project at chapter1 . At chapter 2, the literature review and recommended guidelines for migration to SDN . At chapter 3 and 4 , Analysis and design of both starting and target network. At 5 ,the implementation of migration . At 6 , conclusion and Future work .

1.8 Project Plan

Table : 1 Project Plan

N0	Task Name	Duration	Start	Finish	Predecessors
1	**SDN Migration**	**139 days**	**20/10/17**	**01/05/18**	
2	**Conceptual and initialize project**	**31 days**	**20/10/17**	**30/11/17**	
3	Define the problem	1 day	20/10/17	20/10/17	
4	identify project objectives	5 days	23/10/17	27/10/17	3
5	identify golas	3 days	30/10/17	01/11/17	4
6	Define Feasibility	1 day	02/11/17	02/11/17	5
7	**Plan**	**20 days**	**06/11/17**	**30/11/17**	
8	identify project scop	10 days	06/11/17	17/11/17	
9	identify Tools	5 days	25/11/17	30/11/17	8
10	**literature review**	**30 days**	**05/12/17**	**15/01/18**	
11	Definite and study SDN Technology	10 days	05/12/17	18/12/17	
12	Definite and study Network Virtualization	10 days	20/12/17	02/01/18	

13	Definite and study Migration to SDN	10 days	04/12/17	15/12/17	
14	**execute Project**	**76 days**	**10/01/18**	**25/04/18**	
15	**Analysis**	**21 days**	**15/01/18**	**10/02/18**	**10**
16	Feasibility study	2 days	15/01/18	16/01/18	
17	Definition of methodology analysis	3 days	17/01/18	19/01/18	
18	Specification of requirements	3 days	22/01/18	24/01/18	
16	Describe the current network of UST	2 days	24/01/18	25/01/18	
20	Learn simulation programs	4 days	26/01/18	31/01/18	
21	Comparison between SDN migration use cases	3 days	01/02/18	05/02/18	
22	Determine the steps to migrate to SDN	5 days	05/02/18	09/02/18	
23	**Design**	**10 days**	**15/02/18**	**28/02/18**	**15**
24	Determine design programs	2 days	15/02/18	16/02/18	
25	Design of the network diagram of UST	3 days	16/02/18	20/02/18	
26	Design of the network diagram of UST of SDN technology	7 days	20/02/18	28/02/18	
27	**Implementation**	**14 days**	**04/03/18**	**21/03/18**	**23**
28	Simulation of the current UST network	6 days?	05/03/18	12/03/18	

29	Simulation of the UST SDN network	7 days	13/03/18	21/03/18	28
30	**test**	**12 days**	**26/03/18**	**10/04/18**	**27**
31	Determine the objectives of the test	2 days	26/03/18	27/03/18	
32	Determine the testing Approach	2 days	28/03/18	29/03/18	
33	Determine the test programs	1 day	30/03/18	30/03/18	
34	Test the current UST network	3 days	02/04/18	04/04/18	
35	Test the UST SDN network	4 days	05/04/18	10/04/18	
36	**Conclusions and Recommendations**	**2 days**	**13/04/18**	**16/04/18**	**30**
37	Conclusions	1 day	13/04/18	13/04/18	
38	Recommendations	1 day	16/04/18	16/04/18	
39	Evaluate Project	6 days	20/04/18	27/04/18	

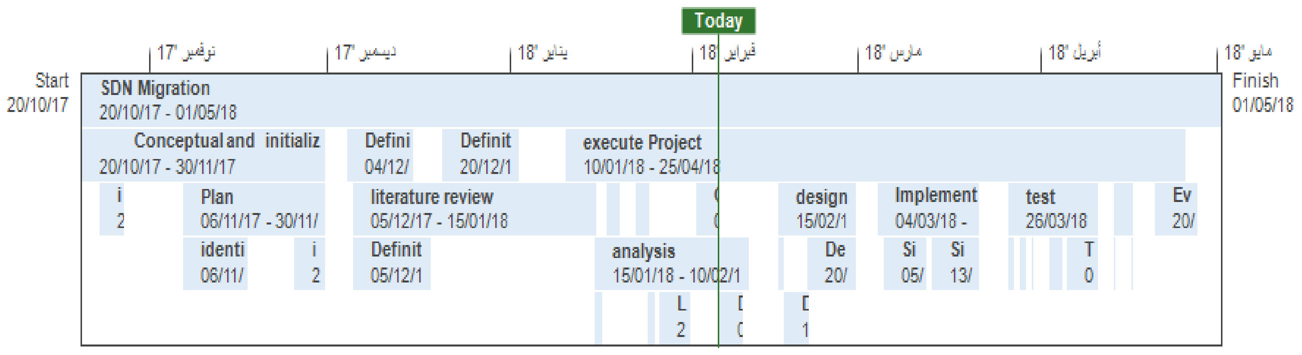


Figure : 1 time line Project Plan

Chapter two

Literature Review

2.1 Introduction :

In this chapter we are going to offer a background about SDN and NFV and about concepts and technologies that are related to SDN like languages , controllers and virtualization . we also well define SDN migration and topics related to SDN migration . We are going to offer a general plan that contains general steps to migrate any organization network to SDN . Finally , we well mention previous projects in SDN migration .

2.2 Definition of Term in Project :

Table 2.1: Definition of Term in Project

Term	Definition
SDN	Software-defined networking (SDN) refers to a new way of organization computer network functionality
OpenFlow	is a protocol for communication between Control layer and Infrastructure Layer
VXLAN	Virtual Extensible LAN base case is to connect two or more layer three network domains and make them look like a common layer two domain
ACL	Access list Routing Filter to make more role and block some output packets
QoS	Quality of Serves make priority for packets
NFV	Network Function Virtualization is technique that uses a virtualization concept to separate network functions from physical infrastructure
OpFlex	OpFlex is created by Cisco and is a southbound protocol similar to OpenFlow
ACI	Cisco Controller Platform
APIC	Cisco Controller
IETF	Internet Engineering Task Force (IETF) develops and promotes voluntary Internet standards, in particular the standards that comprise the Internet protocol suite (TCP/IP)
ONF	Open Networking Foundation (ONF) is a nonprofit trade organization, funded by companies such as Deutsche Telekom, Facebook, Google, Microsoft
Flog	Logic Programming for Software-Defined Networks
Nettle	Taking the Sting Out of Programming Network Routers in SDN
FatTire	Fault Tolerating Regular Expressions
FML	Program language in Folg Programing
Frenetic	Program language in Folg Programing
FRP	language for expressing electrical circuits
VLAN	Virtual Lan
Switchlets	Virtualization of the switch
VINI	Virtual Network Infrastructure
Cabo	Anther of Virtual Network
ATM	Asynchronies Transfer Mode
API	Application Programing Interface
NAS	Network-attached storage (NAS) is a file-level computer data storage server connected to a computer network providing data access to a heterogeneous group of clients
SAN	storage area network (SAN) is a network which provides access to consolidated, block level data storage

2.3 Software-Defined Network (SDN)

A detailed description for the current UST network:

2.3.1 Definition :

separation of control plane function from forwarding plane, it means that networks managed and controlled by software application and SDN controller rather than traditional networks management consoles and commands that requires a lot of administrative overhead .[5],[9.].

2.3.2 disadvantage for traditional networks:

1- **Complexity**: Large number of network devices (like routers , switches, firewall, and so on) and Many complex protocols that embedded on this device.[5],[6],[7],[8].

2- **Inconsistent policies** : like security and Quality of Serves (QoS) policies in traditional network need to be manually configurated for hundreds or thousands devices this makes policies change very complex because manual configurations are prone to error that causes many hours of troubleshooting . [5],[6],[8].

3- **Difficulty of scaling** : with increasing applications and demand on network bandwidth so networks growth needs to redesign the network to meet the requirements. [5],[6],[8].

4- **Vendor dependence:** companies and enterprises seek to deploy new capabilities and services in rapid response to changing business needs or user demands but their ability to respond is hindered by vendors' equipment product cycles which can range to three years or more. [6],[8].

2.4 Benefits of SDN :

1- **An integrated network (Centralized control of multi-vendor environments) :** in SDN we migrate to a centralized model, unlike traditional networks , where the SDN controller becomes the single point of configuration of the network, it means SDN controller software can control any OpenFlow-enabled network device from any vendor.[1],[6].

2- **Consistency** : controller applies policies on group of applications and facilitates policies changing , the framework offers a consistent and concise way to handle policy changes. [5].

3- **Scalable layer 2 across layer 3** : With SDN, we can create layer 2 networks across multiple switches or layer 3 domains using Virtual Extensible LAN (VXLAN), The VXLAN base case is to connect two or more layer three network domains and make them look like a common layer two domain. This allows virtual machines on different networks to communicate as if they were in the same layer 2 subnet.[1],[10].

4- **Flexible application based network** : in SDN everything is an application and all these applications are running on SDN controller .[1].

5- **Unified wired and wireless:** some controllers support both wired and wireless network.[1]

6- **Extensible policy model :** Because the policy model is open and can be extended to other vendors and other device types .[5]

2.5 SDN architecture:

ONF organization defined SDN to three layers : [6]

1- **Application Layer** : this is the first layer that consists of services and applications that are provided by the network for users like Routing Filter ACL and QoS this layer connected to control layer by Representational state transfer (REST) APIs (northbound interface).
2- **Control layer** : it Represents the central control point that gives instructions to network devices by controller applications. There are many controllers we discuss them later ,this layer connected to infrastructure layer by southbound protocols we are going to explain it later.
3- **Infrastructure Layer** : consists of devices that forwarded and routing for data Whether physically or virtual such as in Network Functions Virtualization(NFV).

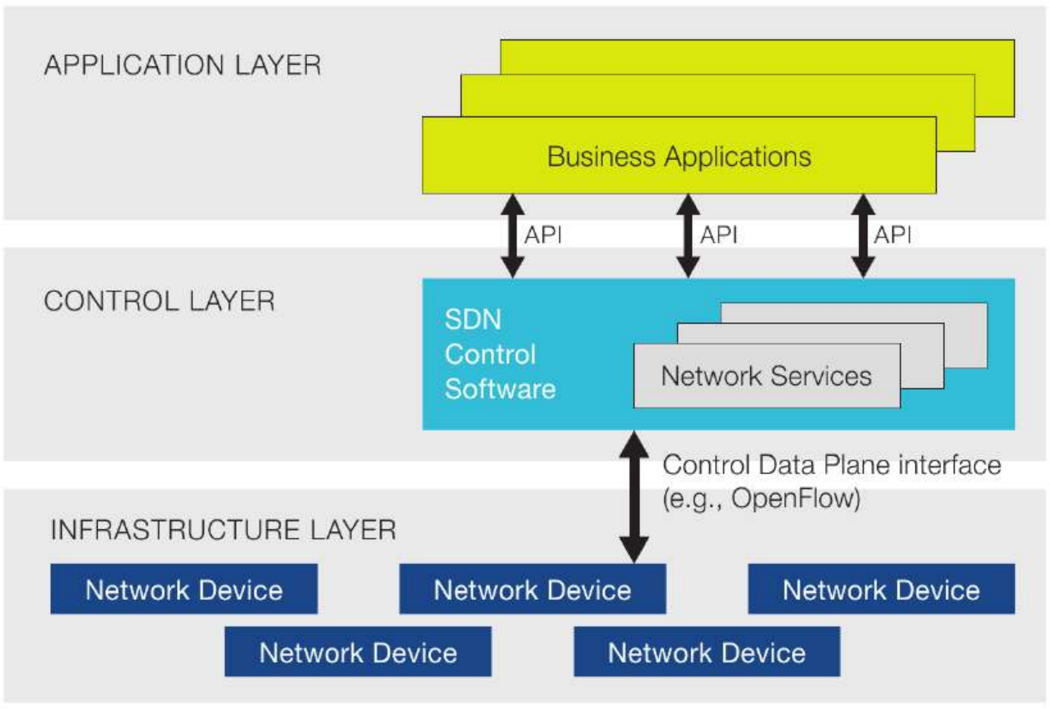


Figure 2.1 : SDN architecture

2.6 NFV :

is technique that uses a virtualization concept to separate network functions from physical infrastructure ,that means that to create virtual network functions like firewall ,switches ,routers and load balancers using standard x86 servers . SDN is decoupling the control from the data plane that means to technologies is dependent on the other.

" The decoupling of network functions from the underlying hardware is closely related to the decoupling of the control from the data plane advocated by SDN and therefore the distinction of the two technologies can be a bit vague. It is important to understand that even though closely related, SDN and NFV refer to different domains.

NFV is complementary to SDN but does not depend on it and vice-versa. For instance, the control functions of SDN could be implemented as virtual functions on the NFV technology. On the other hand, an NFV orchestration system could control the forwarding behavior of physical switches through SDN. However, neither technology is a requirement for the operation of other, but both could benefit from the advantages each can offer." [7].

2.7 The southbound protocols :

There are a number of protocols that are used by SDN controllers to provide programmatic interface for how communication between Control layer and Infrastructure Layer like OpenFlow and Cisco's OpFlex .[1],[2].

2.7.1 OpFlex:

OpFlex is created by Cisco and is a southbound protocol similar to OpenFlow. However, this protocol is open source . Cisco has already submitted it to IETF. Cisco sets OpFlex in their Nexus 9,000 switches as well as on their SDN controller platform (ACI) and controller (APIC).[1].

2.7.2 OpenFlow

Definition : is a protocol for communication between Control layer and Infrastructure Layer . OpenFlow provides external program with access to forwarding plan of network . The ONF manages the OpenFlow standards . OpenFlow allows network engineers to define how traffic flow through network devices based on parameters such as usage patterns, applications, and cloud resources,[1],[2],[4].

OpenFlow switch: is a physical device or virtual software in infrastructure layer that supports OpenFlow protocol to forward packets in SDN environment. General, OpenFlow pushes policy rules out from the controller to the switches.[1],[2],[4].

2.7.3 The OpenFlow switch consists of three components [4]:

1- Group table is responsible with flow tables for performing packet lookups and packet forwarding.

2- Flow tables

3- OpenFlow channel is used for communication with controller founded in control layer

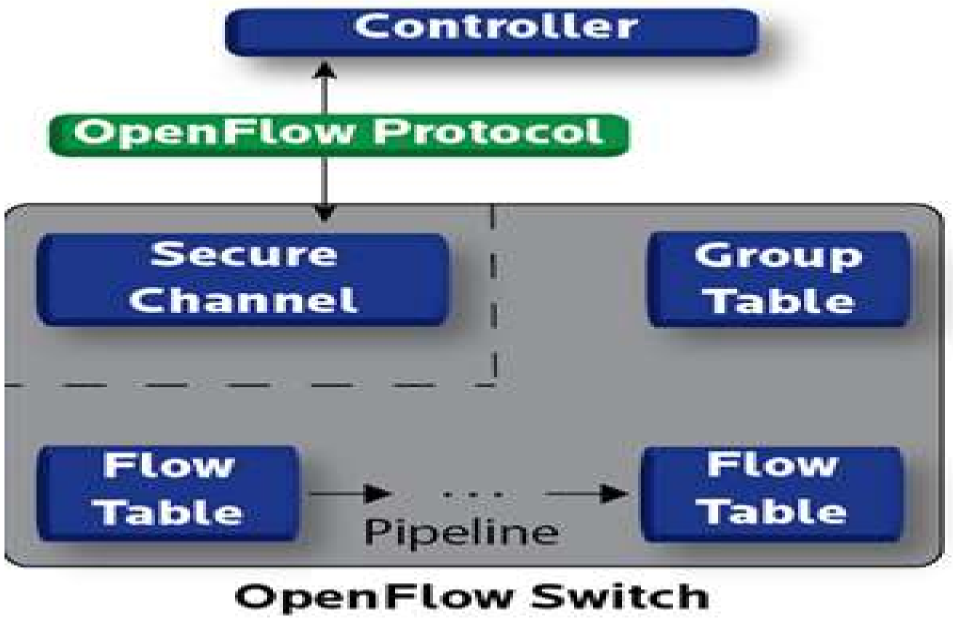


Figure 2.2: OpenFlow switch

Table 2.2: Some devices that support OpenFlow protocol

Vendor	Product
HP	1- HP 5920 & 5900 Switch Series. 2- HP 5130 EI Switch Series. 3- HP Switch 2920 series. 4- HP Switch 3500 series. 5- HP Switch 3800 series. 6- HP Switch 5400 series, v1 and v2 modules 7- HP Switch 5406R series. 8- HP Switch 5412A series. 9- HP Switch 8200 series, v1 and v2 modules
Juniper Networks	1- MX Series 2- EX9200 3- QFX5100 4- EX4600
Alcatel-Lucent	1- OmniSwitch 10K. 2- OmniSwitch 9900. 3- OmniSwitch 6900 4- OmniSwitch 6860 5- OmniSwitch 6865
IBM	IBM System Networking RackSwitch G8264
Brocade	Brocade VDX 2741, Brocade VDX 6740, Brocade VDX 6940 and Brocade VDX 8770

2.8 SDN controllers:

2.8.1 Definition :

is a control point program located in control layer that supports OpenFlow protocol or any southbound protocols .It provides programmatic interface to switches (southbound interface) and communicate with applications in application layer (Northbound interface) , the controller is the network operating system.

2.8.2 Example :

OpenFlow controller each time a decision must be made on the OpenFlow switches such as when a new packet flow reaches an OpenFlow switch and its receive from network applications additional policies and information for implement its on OpenFlow switches .

2.8.3 Controller types :

There are a lot of controller and each of them consist of advantages and disadvantage We will discuss some of them :

1- **NOX and POX**

NOX was first OpenFlow controller written in C++ and provides API for Python too, It was developed to new NOX support only C++ its much faster and much cleaner codebase , the NOX processing of OpenFlow messages incoming individually the NOX simple to implement, but not efficient and robust and couldn't scale. Then it developed to only python version of NOX which it called POX, POX is open source. [1],[2],[3].

2- **Trema**

Trema OpenFlow Controller is an extensible set of Ruby scripts and C. [2],[3] 3- **Beacon :**

Beacon is a fast, cross-platform, modular, Java-based controller that supports both event-based and threaded operation but it was only supporting star topologies, which was one of the limitations of this controller.[1],[3]

4- **Floodlight**

Floodlight is a Java-based OpenFlow controller, based on the Beacon implementation it supports both physical and virtual OpenFlow switches ,
And it is open source and under Apache license the Floodlight a set of Java module applications, which are loaded in the Floodlight properties file for example, learning switch, hub, firewall, and static flow entry pusher . [1], [2],[3].

5- **Application Policy Infrastructure Controller (APIC)**

Its support OpFlex protocol that is southbound protocol between Cisco APIC controller and Cisco Nexus switches.[1]

6- **OpenDaylight (ODL)**

OpenDaylight is a Linux Foundation Collaborative project it is Built on Java language and support Multithreading and its support OpenFlow protocol and OpFlex protocol the ODL is opensource The OpenDaylight Project is currently supported by 31 networking industry companies. These supporters include Cisco, IBM, Juniper Microsoft, Redhat, VMWare Ciena, Intel, Dell, HP, and many more. [1], [2],[3]

7- **Ryu**

Ryu is a component-based, open source ,integrates with OpenStack and supports OpenFlow ,it is built on python language. [1], [3].

Table 2.3: SDN Controller types

Controller	Language	PD	OF	MT	TLS	RA	%	Other Features
OpenDayLight	Java/Python	Y	1.3	Y	Y	Y	61%	GUI
ONOS	Java	Y	1.3	Y	_	_	23%	Built for Service Providers. Also, supports OVSDB, BGP, Netconf, TL1.
OPNFV	Java/Python	_	1.3	Y	Y	Y	26%	Supports ODL, ONOS, or OpenContrail, NV/SDN.
DIFANE	C	Y	1.0	Y	_	_	_	Built on NOX
HyperFlow	C++	Y	1.0	Y	_	_	_	Built on NOX
NOX	C++	N	1.0	N	_	_	_	Deprecated
SNAC	C++	N	1.0	N	_	_	_	Built on NOX, GUI, closed source
POX	Python	N	1.0	N	_	_	_	Development stagnated, GUI.
Pyretic	Python	N	1.0	N	_	_	_	Built on POX. [Deprecated.]
Ryu	Python	N	1.5	N	Y	_	15%	Frequent switch certifications.
Ryuretic	Python	N	1.5	N	Y	_	_	Built on Ryu.
Beacon	Java	N	1.0	Y	_	_	_	Limited to STAR topo, GUI.
Floodlight	Java	N	1.4	Y	Y	Y	11%	Forked from Beacon, GUI.
Trema	Ruby/C	N	1.3	_	_	_	_	
Kandoo	Go	Y	1.0	Y	_	_	_	
OpenContrail	Java/Python	Y	_	Y	Y	Y	14%	Developed for OpenStack. Southbound API: XMPP, BGP, and NETCONF.
OpenMUL	C/Python	Y	1.4	Y	Y	Y	8%	Supports NETCONF & OSVDB. Created for stability and performance.

PD (Physically Distributed), MT (Multi-Threaded), TLS (Transport Layer Security) , RA (Rest API), OF (OpenFlow Version),

% (Deployed Percentage [3])–Note some deployments may use more than one controller type.

2.9 Network Programming Languages

In this section we are going to discuss languages used in SDN:

• Flog : Logic Programming for Software-Defined Networks.

• Nettle : Taking the Sting Out of Programming Network Routers.

• FatTire : Fault Tolerating Regular Expressions.

2.9.1 Networks Management

2.9.1.1 In the past:

- Networks managed through a set of complex low-level, and heterogeneous interfaces
- Firewalls + network address translators + load balancers + routers + switches == Configured separately
- Thousands of lines low-level code in different domain-specific languages
- Complex routing mechanisms (error-prone tasks)

2.9.1.2 In Software Defined Networks

• Logically centralized Controller:

– Managing distributed switches

– General purpose machines

– Working on routing decisions

– Instruct the switches to install the necessary packet-forwarding rules.

2.9.2 Flog :

SDN Logic Programming Language

• SDN packet-forwarding rule :

➔ Predicate + Action + Priority

• Example:

➔Predicate: match packets based on the IP header (MAC, IP, etc.)

➔ Action: Drop, forward or flood the packet to ports

➔ Priority: rules are executed according with priorities

Combines two programing languages:

2.9.2.1 FML:

• set of high-level built-in policy operators (SDN abstractions)

• allow/deny certain flows

• provide quality of service

• Programing model Not flexible

2.9.2.2 Frenetic:

• declarative query language - SQL-like syntax

• functional stream-processing language

• language for describing packet forwarding

• From FML:

– Programming for controlling software-defined networks.

• **From Frenetic:**

Controller programs split into :

- Mechanism for querying network state
- Mechanism for processing data extracted from queries
- Component for generating packet-forwarding policies

  (automatically push to the switches)

Event driven => execution of logic programs

1. Generates a packet-forwarding policy compiled and deployed on switches

2. Generates states : drive the logic program when the next network event is processed

Network Events:

- Switches online / offline
- Ports on switches active / inactive
- Statistics gathered by switches
- Packets arrive at the controller and require handling

2.9.3 Nettle :

Taking the Sting Out of Programming Network Routers

- Do not configure network ; program it

- Networks of OpenFlow switches controlled using a high-level, declarative and expressive language
- Based on the principles of functional reactive programming (FRP)
- Embedded in Haskell => general-purpose purely functional programming language.
- Domain Specific Language.

2.9.3.1 Layered architecture:

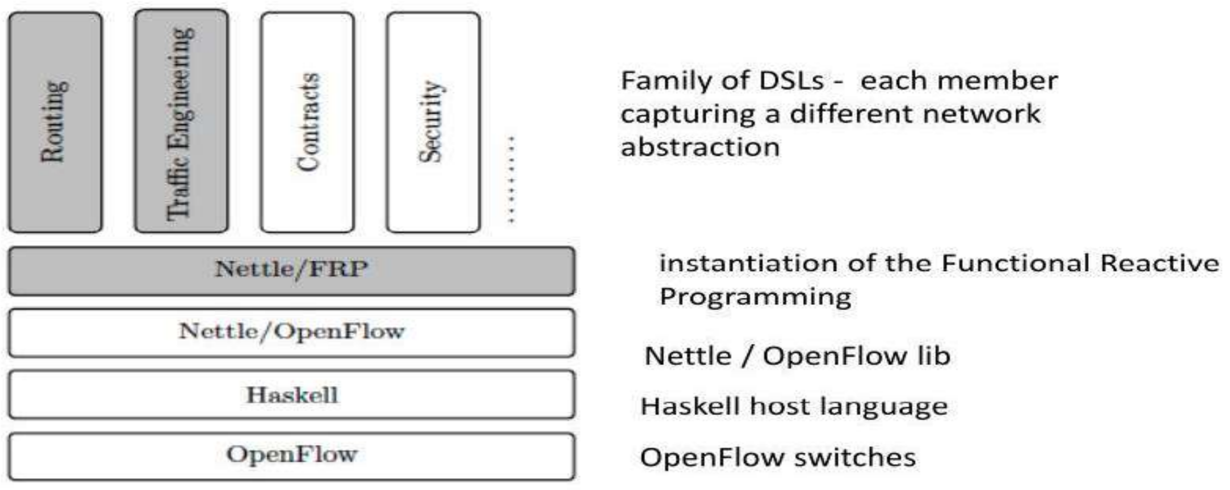


Figure 2.3: Nettle Layered architecture

2.9.3.1.1 Nettle/FRP :

as a language for expressing electrical circuits

• Focus on the stream of control messages among OpenFlow switches

• Nettle => powerful collection of

– Signal functions

– Event operators

2.9.3.1.2 OpenFlow switches :

maintains flow table with flow entries:

❖ Match conditions IPs , header Fields

❖ Forword action to specific ports, flooding , dropping packets

❖ Statistics are updated

❖ Expirations settings expires a flow entry after prescribed time

2.9.3.1.3 Nettle Controller transforms:

stream of messages from switches

⬇

stream of commands for switches

2.9.4 FatTire : Fault Tolerating Regular Expressions

• Programs for fault tolerant Networks

• Based on regular expressions

• Main features:

- ➢ Expressive: easy to describe forwarding and fault tolerant policies
- ➢ Efficient: based on fast failover from OpenFlow
- ➢ Correct: reasoning about the behavior of the system during failure recovery

Central feature: Regular expressions for sets of legal paths through the network.

• FatTire programs are translated to OpenFlow switch configurations.

• Automatic response to link failures with no controller intervention.

Table 2.4: Network Programming Languages

Program Language	Main Characteristic	Advantages	Disadvantages	Implemnted in
Flog	- Network Event driven - Focused on packets Flow	- Simple - Combines Frenetic and FML	Too simple and limited to flow control	C++,Python
FatTire	Targets fast failover mechanisms provided by OpenFlow standard	- High level -Regular expression powered - Turns failover scenarios easier to understand	Only focused over solving link failures configuration	OCmal
Nettle	- allow fine-grained control over switch behavior - event-based programming model	- Strong typed - Extensible		Haskell

2.10 Emulators

For Emulator network using SDN we brefere using Mininet Emulator.

Mininet is :

- a network emulator that runs in a Virtual Machine.
- A virtual network environment that can run on a single PC
- Runs real kernel, switch, and application code on a single machine
    - Command-line, UI, Python interfaces.
- Many OpenFlow features are built-in
    - Useful: developing, deploying , and sharing

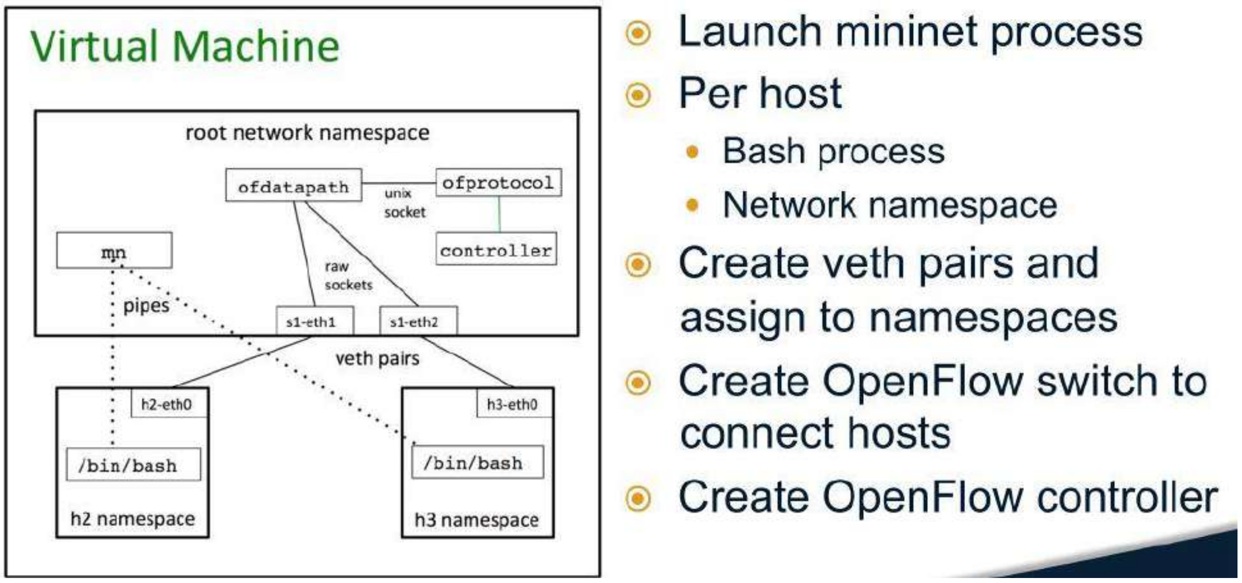
Why Mininet ?

- Fast
- Possible to create custom topologies
- Can run real programs (anything that can run on Linux can run on a Mininet host)
- Programmable OpenFlow switches
- Easy to use
- Open source

Alternatives

- Real system: Pain to configure
- Networked VMs: Scalability
- Simulator: No path to hardware deployment

The Mininet VM in a Nutshell



- Launch mininet process
- Per host
  - Bash process
  - Network namespace
- Create veth pairs and assign to namespaces
- Create OpenFlow switch to connect hosts
- Create OpenFlow controller

Figure 2.4: The Mininet VM in a Nutshell

2.11 Network Virtualization

The meaning :-Representation of one or more logical network topologies on the same infrastructure.

2.11.1 Many different instantiations:-

- Virtual Lan (VLAN)
- Various technologies and network testbeds
- Today: VMWare, Nicira, etc.

2.11.2 Benefits of Network Virtualization :-

Sharing

- Multiple logical routers on a single platform
- Resource isolation in CPU , Memory , Bandwidth , Forwarding Table.

Customizability

- Customization routing and forwarding software.
- General-purpose CPUs for the control plan
- Network processors and FPGA for Data plan

2.11.3 Three Examples of Virtual Networks

1- Tempest: Switchlets (1998)

- Separation of control framework from switches
- Virtualization of the switch

2- VINI: A Virtual Network Infrastructure (2006)

- Virtualization of the network infrastructure

3- Cabo: Separates infrastructure, services (2007)

2.11.3.1 Tempest : swichlits:

- Multiple control architectures over ATM.
- Separation of switch.
- controller and fabric via open signaling .
- Partitioning of switch resources across controllers.
- Partitions port space, bandwidth, buffers.
- Different controllers control each switchlet..

2.11.3.2 VINI : virtual network infrastructure

- Enable deployment studies in real networks
- Share the nodes, links using virtualization

The first requirement of VINI is that it be a fixed infrastructure.

The reason is for control: experiments don't want the network topology changing from under them. The idea is that your experiment will provide layer 3 Building a VINI requires a big investment, so it has to be shared. At the same time, we expect some VINI experiments to be long-running. So there can be a red experiment deployed on VINI, and make experiment on the same nodes. A VINI isolates experiments by giving each the illusion of dedicated hardware and resources. And every experiments take its own topologies A VINI exposes network events and can also inject them into an experiment.

Experiments can carry traffic for real end-users

Experiments can participate in Internet routing

- How does an experiment find out about external destinations?

  It should be able to integrate with the current Internet routing infrastructure, e.g., peer with BGP routers.

  Ultimately experiments should be able to advertise address blocks into the Internet, become a virtual ISP.

2.11.3.2.1 VINI Status - Virtual Hosts

- VINI based on PlanetLab software
  - Simultaneous experiments in separate VMs
  - Each has "root" in its own VM, can customize
  - Reserve CPU and bandwidth per experiment

2.11.3.2.2 VINI Status - Virtual Networks

- ➢ Configure a virtual topology for a slice
  - Point-to-point virtual Ethernet links
  - Slice controls routing table, virtual devices on the virtual hosts

Purpose: allow experimentation with routing software (e.g., XORP, Quagga) that already runs on Linux.

2.11.3.2.3 VINI Trellis v0.1

- ➢ Virtual host
  - Linux kernel IPv4 routing table
  - Point-to-point virtual Ethernet
  - Applications add/change routes
- ➢ Substrate
  - Ethernet software bridge
  - Traffic shaper
  - Ethernet-over-GRE tunnels (to span multiple IP hops)

2.12 OpenStack

Definition : OpenStack is a group of open source cloud computing platforms . It began in 2010 as a joint project of Rackspace Hosting and of NASA. It provides an interface to manage every aspect of virtual machines.

2.12.1 OpenStack Architecture

OpenStack architecture is described according to the services it provides

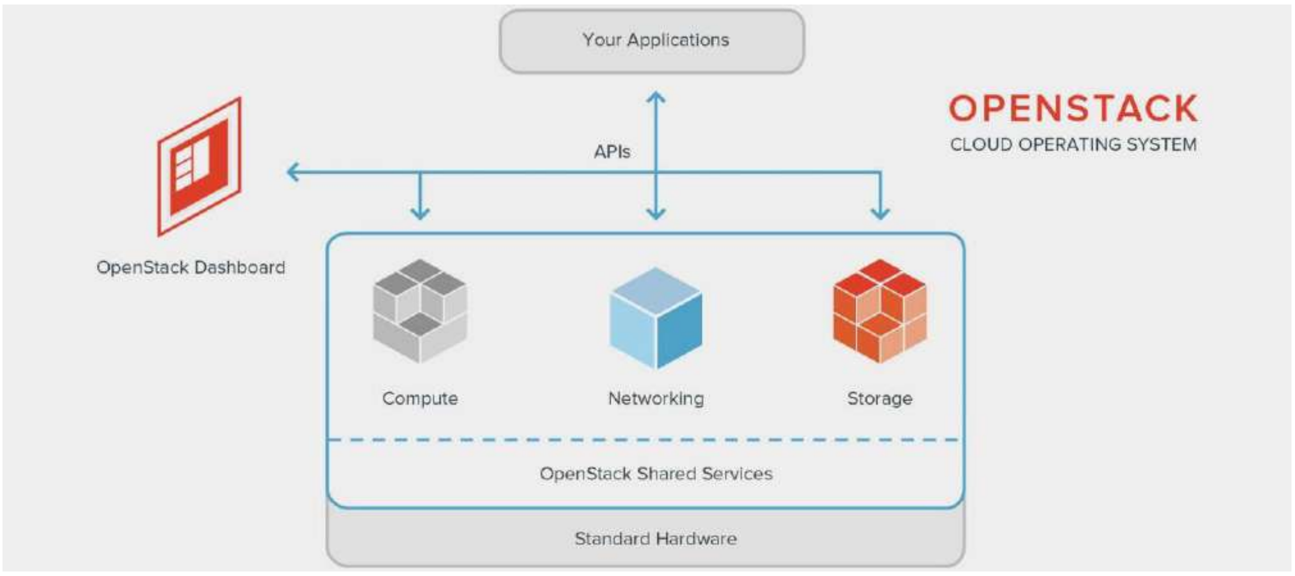


Figure 2.5: OpenStack Architecture

OPENSTACK generally consist of :

1- Compute services
2- Network services
3- Storage services
4- OpenStack controls

In each component, there are many services

Table 2.5: Primary OpenStack services

OPENSTACK SERVICE	OPENSTACK PROJECT	DESCRIPTION
Compute	Nova	A compute service responsible for creating virtual machine instances and managing their life cycle, as well as managing the hypervisor of choice. The hypervisors are pluggable to Nova, while the Nova API remains the same, regardless of the underlying hypervisor.
Identity service	Keystone	Provides an authentication and authorization service for OpenStack services. Provides a catalog of endpoints for all OpenStack services. . Keystone is capable of integrating with third-party directory services such as LDAP.
Networking	Neutron	Enables network connectivity as a service for other OpenStack services, such as OpenStack Compute. Has a pluggable architecture that supports many popular networking vendors and technologies. Provides an API for users to define networks and advanced services such as FWaaS
Block storage	Cinder	block storage service responsible for creating and managing external storage, including block devices and NFS. It is capable of connecting to vendor storage hardware through plug-ins. Cinder has several generic plug-ins, which can connect to NFS and iSCSI, for example. Vendors add value by creating dedicated plug-ins for their storage devices.
Object storage	Swift	Stores and retrieves arbitrary unstructured data objects via a RESTful, HTTP based API. It is highly fault tolerant with its data replication and scale-out architecture. Its implementation is not like a file server with mountable directories.
Image service	Glance	An image service responsible for managing images uploaded by users. Glance is not a storage service, but it is responsible for saving image attributes, making a virtual catalog of the images.
Dashboard	Horizon	dashboard that creates a GUI for users to control the OpenStack deployment. This is an extensible framework to which vendors can add features. Horizon uses the same APIs exposed to users.
Orchestration	Heat	orchestration service responsible for managing the life cycle of the OpenStack infrastructure (such as servers, floating IP addresses, volumes, security groups, and so on) and applications. Uses Heat Orchestration Templates (HOT) to describe the infrastructure for an

		application and provides an API for Amazon's AWS template format.
Telemetry	ceilometer	The Telemetry service uses an agent-based architecture. Several modules combine their responsibilities to collect, normalize, and redirect data to be used for use cases such as metering, monitoring, and alerting.

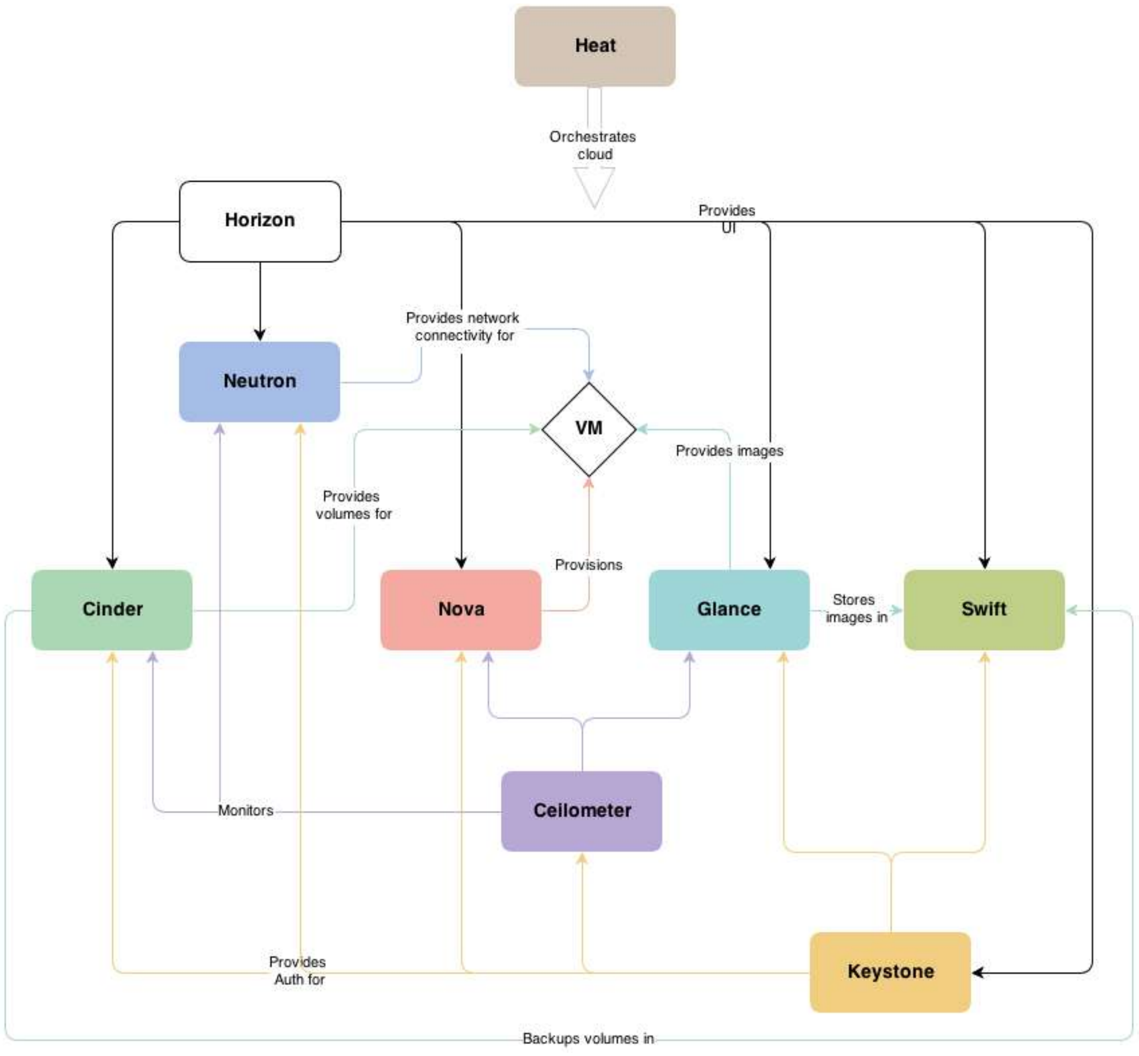


Figure 2.6: Primary OpenStack services

2.13 Integrate SDN with NFV

NFV and SDN are technologies capable of providing one network solution. SDN can provide connectivity between Virtual Network Functions (VNFs) in a flexible and automated way, whereas NFV can use SDN as part of a service function chain.

2.14 OpenFlow Manager (OFM)

OpenFlow Manager (OFM) is an application developed to run on top of ODL to visualize OpenFlow (OF) topologies, program OF paths and gather OF stats. Software Defined Networking (SDN) involves an application interacting with a network (composed of domain-specific devices) for the purpose of simplifying operations or enabling a service. A controller is positioned between the application and network and interacts with network elements (e.g. switches) in the southbound direction using a variety of different protocols. In the northbound direction it presents an abstraction of the network using in practice common REST APIs. The controller vehicle for this application is ODL. The OpenFlow Manager (OFM) is an application that leverages this innovation to manage OpenFlow network.

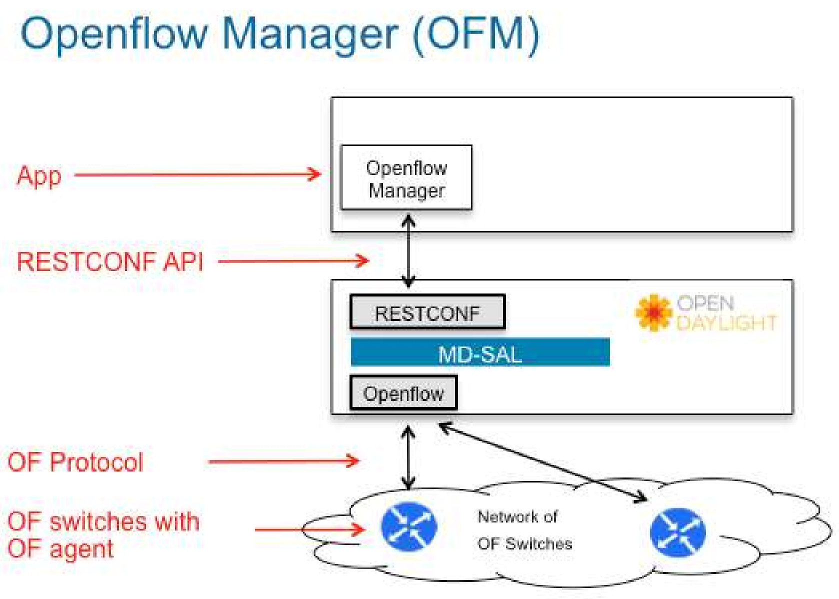


Figure 2.7: OpenFlow Manager (OFM)

2.15 Migration to SDN

In this section we are going to cover SDN Migration. We are going to cover the concept of migration briefly , a description about ONF , and methods and standards about migration to SDN according to ONF .We also well mention previous projects and previous studies about migration.

Migration:

In SDN, this term refers basically to the process of transporting a traditional network to SDN network.

2.16 Open Networking Foundation (ONF)

ONF (as they introduce their selves) is a consortium that owns standardization of OpenFlow and it is dedicated to the promotion and adoption of SDN through open standards development

2.17 Migration Guidelines

ONF recommends some guidelines and recommendations for migration:

1 - Network scenarios and use cases are to be identified , that well really simplify the migration .

2 - The target network and its core requirements must be fully identified.

3 - The objective is to simplify the network and reduce the cost of operation. A secondary goal is to improve utilization.

4 – During migration steps , there are risks may effect the migration and the network its self. So , for each step must be a way for rollback

2.18 Migration Requirements

Here are some OpenFlow standard based objectives and high-level requirements that must be met:

1 - The target network software must support programmability, through Application Programming Interfaces (API) , so the controller should be able to control the devices through APIs

2 - The target network must support software updates and rollback

3- The target network must support heterogeneity, so multiple devices from different vendors are supported.

4- Starting network needs to be transformed into an intermediate state then the process can be continued from a specific point safely .

2.19 SDN Devices

Devices in SDN can be classified to three classes:

1 – legacy Devices : traditional switches that does not support OpenFlow

2 – OpenFlow Devices: Switches with only OpenFlow forwarding plan

3- Hybrid Devices: Switches with both legacy control and data plan and OpenFlow capabilities

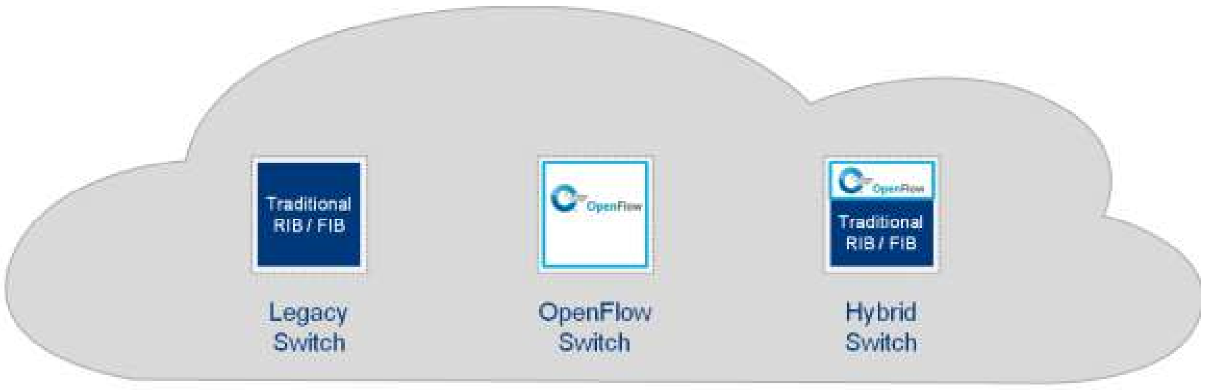


Figure 2.4: Types of Diveces

2.20 Migration Approaches

There are 2 main approaches that are used for migration listed as follows :

1 – Direct migration(GreenField):

In this approach , upgrading existing networking devices with OpenFlow Agents and decommissioning the control machine in favor of OpenFlow controllers and configurators.

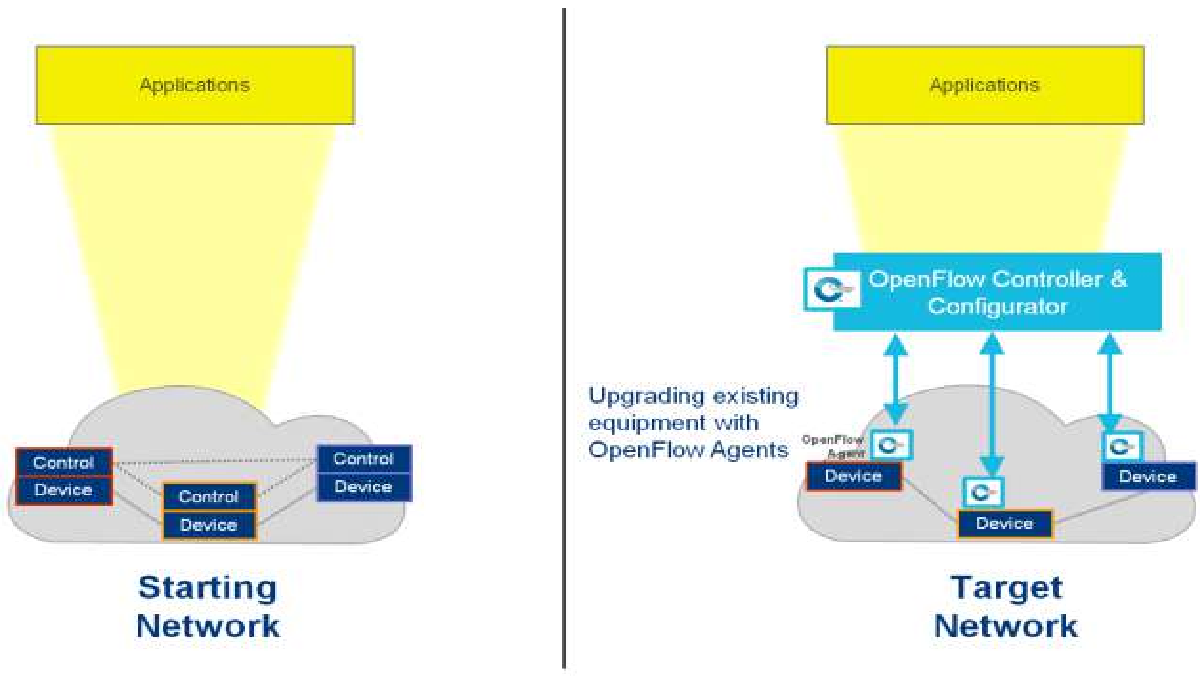


Figure 2.5: GreenField Migration

2 – Phased approach :

OpenFlow devices are deployed in conjunction with exiting devices. Network operations are maintained by both existing control machine and openflow controller and configurator. Once finishing migration , control machine is decommissioned

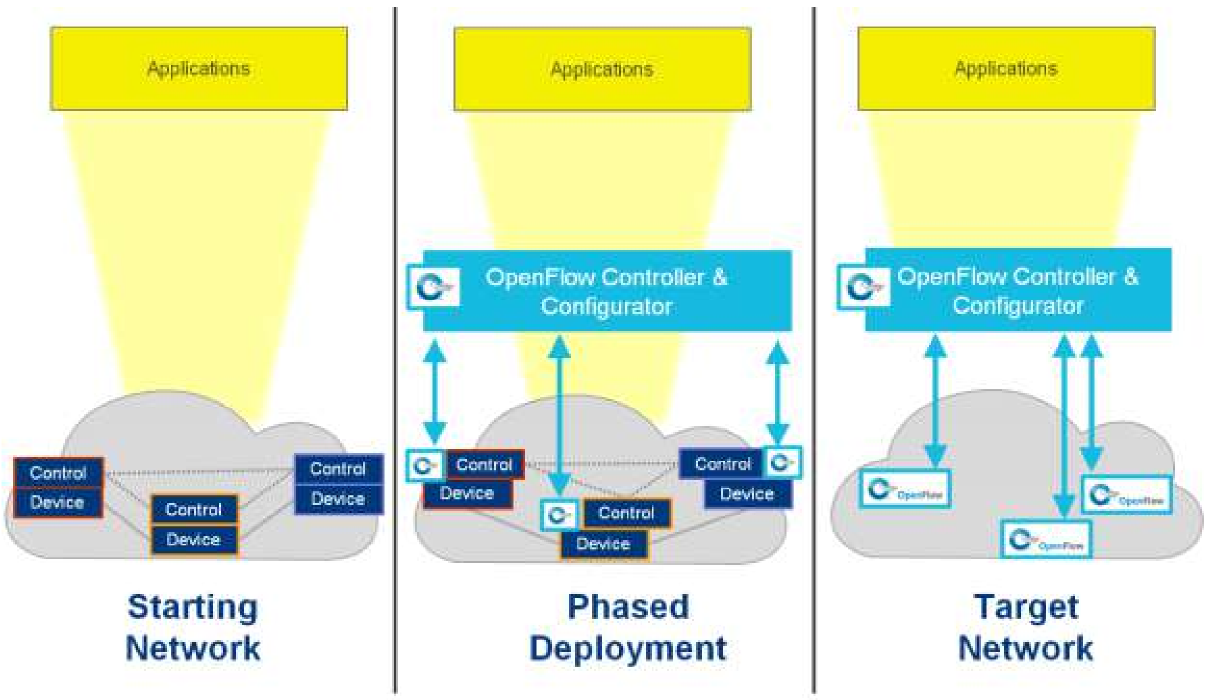


Figure 2.6: phased Migration

This approach can be classified to :

A – Mixed deployment: new openflow devices are used with legacy devices

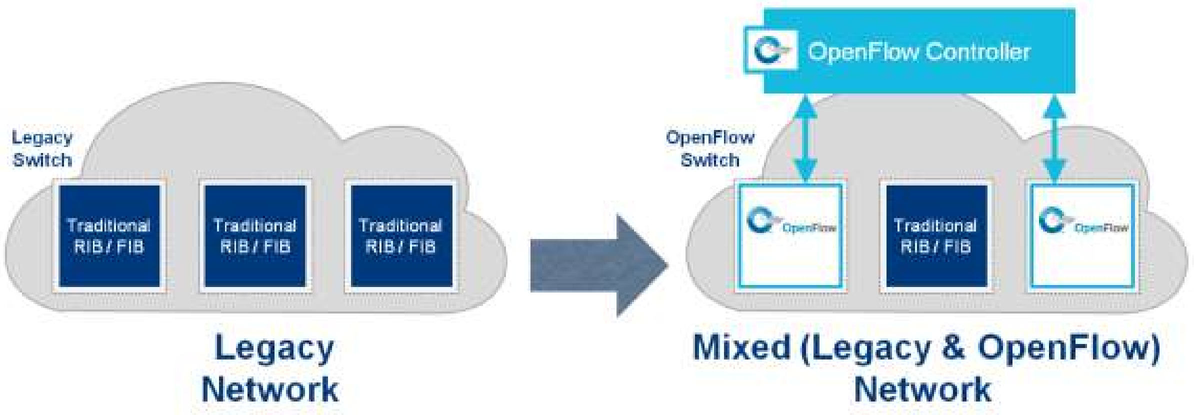


Figure 2.7: Mixed Deployment

B – Hybrid deployment : legacy devices , openflow devices and hybrid devices are used

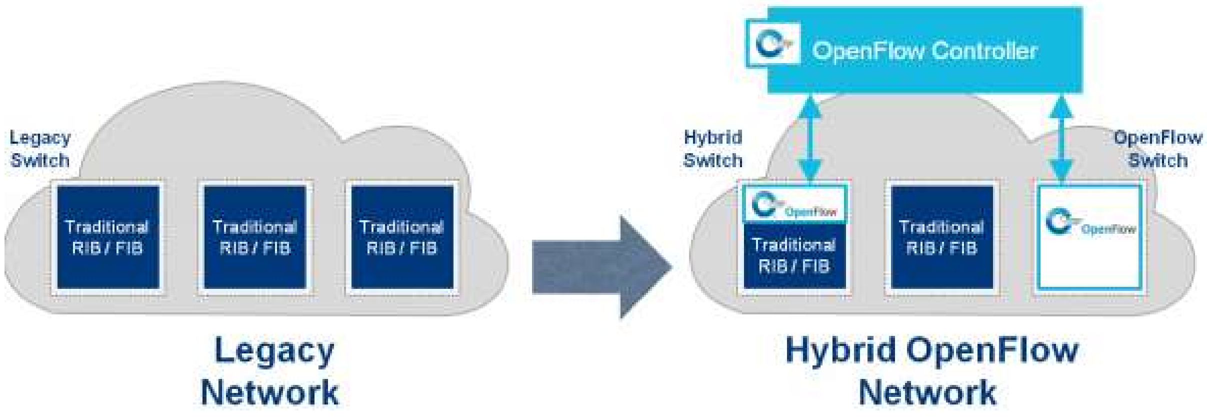


Figure 2.8: Hybrid Deployment

2.21 Network Types

Some types of network that must be considered for migration process:

1 – Campus Network :

multiple buildings, each building have a wiring closet. The buildings are interconnected with a central operations center. Components of the Campus network include a Campus wide backbone with an egress point to WAN that associated to data center

2 – Enterprise Network:

Composed of network resources (e.g NAS or SAN) used to interconnect subnetworks of servers(virtual or physical)

3 – Multi -tenant:

Sharing physical resources using virtualization

3 – WAN/SP Networks:

Significance very big projects in using SDN for to manage interchange resources

2.21 SDN Deployment Architecture

The deployment architecture depends on the network that will be migrated but typically it consists of forwarding devices controlled by logically centralized controllers and physically ,the controller can be centralized on servers or embedded with some or all openflow devices.

2.23 Traffic in SDN

A very important note in SDN is to consider the traffic flows between the controllers and agents, and to but in mind the cases that 100% of traffic deviated to controller and the cases that require heavy traffic like VM migration.

2.24 Pre-migration Planning

During migration phases, the main goal is to save service continuity and to keep reachability . In light of this goal , there are some practices offering best migration:

1 – Gap analysis: to ensure that there are alternatives to use in the case of meeting challenges during migration

2 - Back-out Procedures: clear procedures that offering a way to back to the starting network if something goes wrong during migration steps.

3 - Feature-Set analysis: defining what features are required and ensuring that openflow protocol or switches meet these features.

2.25 Plan for Migration Recommended

ONF offers a document to use to simplify migration to SDN. This document may differ from network to network as needed. Here we provide a quotation of ONF document.

2.25.1 Starting Network

Must be very understood and documented. The type of the network , hardware used in the network , software , applications , monitoring tools , protocols , and buildings .

1- General Description :

the purpose of the network and services delivered by the network.

2- Operational Mode:

Description for network layers and protocols , which one well be included or excluded from migration.

3- Deployed Equipment:.

a description of interfaces, traffic flows, etc in SDN equipment

4- Redundancy Model:

a specification of redundancy model functional requirements.

5- Management Tools:

A description of the management systems and tools that are used for monitoring and managing the network.

6- Network Capacity

Speeds, traffic , number of users and nodes .

7- Problems and Challenges

Problems in starting network that motivate to migrate to SDN

8- Pre-Migration Assessment

To compare SDN network performance with the starting network

2.25.2 B - Target Software-Defined Network

The motivation of SDN , Characteristics of target network and requirements of SDN

1- Objectives

Problem statement of issues that well be solved by SDN and how SDN will solve these issues

2- SDN Architecture

Detailed architecture of target network including determining the protocol that well be used in SDN (e.g openflow) , devices used in target network , interfaces , and physical architecture that offers locations of controller and devices and how they well be connected. Traffic flow also should be mentioned here.

3- Migration Approach

Determining the migration approach with clear satisfying and the process of the approach

4- Dependencies Target

Defining what the solution depends on like controllers , network management tools and services must be delivered

5- GAP Analysis

What is missing to get the migration completed? Are existing tools sufficient, or is there a need for new tool development? Where gaps exist, contingencies should be specified.

6- Migration Procedures

a step-by-step documentation for the migration . In other words the plan from starting network through SDN network

7- Post-Migration Acceptance and service acceptance

Testing the success of migration to ensure that SDN support expected services.

8- **Migration Timeline**

time planning should be undertaken with possible rollbacks.

9- Skill Sets Requirements

Skills needed to finish migration like understanding of Vlans or expertizing a programing language .

2.26 Previous projects:

2.26.1 Google Inter-Datacenter WAN Use Case

Google’s global user based services (Google Web Search, Google+, Gmail, YouTube, Google Maps, etc.) require significant amount of data to be moved from one region to another, making these applications and services very WAN-intensive. Google concluded that the delivery of such services would not be scalable with the current technologies due to their non-linear complexity in management and configuration. As a result, Google has decided to use SDN for managing WAN as a fabric as opposed to a collection of boxes.

2.26.2 NTT Provider Edge Use Case:

This use case documents NTT’s migration of BGP to OpenFlow.In the traditional BGP deployment models, the Provider Edge (PE) router maintains numerous BGP adjacencies as well as large number of BGP routes/paths for multiple address families such as IPv4, IPv6, VPNv4 and VPNv6 etc. In addition, to meet customer service level agreements, the PE may be configured with aggressive BGP session or Bidirectional Forwarding Detection (BFD) timers. Handling BGP state machine, processing BGP updates as per configured policies and calculating best paths for each address-family puts a heavy load on the router. Additionally, by definition, service changes are quite frequent on the PEs to provision new customers or update customer policies.

Because of the limited resources, including CPU and memory, as well as the proprietary nature of the operating system (OS), service acceleration and innovation is dependent on vendor implementation. BGP free edge defines a new paradigm of simplifying the eBGP routing (control plane load) on the PE routers.

In this deployment model, a PE router is converted into a forwarding/transport node to handle data plane traffic whereas BGP control plane function is offloaded to a separate external entity. The external control plane entity leverages OpenFlow/SDN to program the forwarding entries for the data plane traffic on the PE router.

2.26.3 Stanford Campus Network Use Case

A part of the Stanford campus network was successfully migrated to support OpenFlow in 2010.

Chapter Three

Analysis

3.1 Introduction :

In this chapter , we are going to mention SDN and NFV requirements . We also well implement the general SDN migration plan that we provided in UST migration . We are going to provide a comparison between migration plans of UST network with other use cases .

3.2 Requirements

Requirements include business requirements and functional requirements:

3.2.1 Business Requirements

- **Increase business agility**

  The flexibility of SDN makes it far easier and faster to roll out new innovative services, such as real-time HD video conferencing and cloud applications, while still delivering a consistently high quality end user experience

- **Eliminate vendor lock-in**

  Open platforms are key in eliminating vendor lock-in and driving growth in SDN.

  According to Transparency Market Research the SDN market is set to surge to US$3.52 billion by 2018. The OpenDaylight platform, which is leading the transformation to open SDN, now accounts for 95 percent of the entire SDN market. This enables enterprises to use multivendor solutions to benefit healthy price competition and faster innovation.

- **Reduce costs**

  SDN pools multiple compute, storage and processing functions onto low cost commodity servers to reduce capital expenditure.

  At the same time, NFV enables a lot of manual network configuration and management tasks to be automated, reducing operating costs.

3.2.2 Physical Requirements :

The UST SDN Architecture needs to address the following requirements :

- **Simplicity and Expressiveness**

  Generic semantic models that can represent multiple abstraction layers need to be identified.

  Such semantic models can include specific Information Models, but models that encompass behavior, e.g. more general domain models, also need to be considered.

- **Applicability**

  - **Definition of Scope and Value Proposition**

    The SDN Architecture needs to ensure compatibility with existing network paradigms, standards, and common practices. The various paradigms need to be able to coexist. Abstractions and standard protocols need to be defined to allow for rapid adoption of SDN solutions into existing operational environments motivated by the most business critical usage scenarios.

  - **Network Types and Paradigms**

    The SDN Architecture needs to make provision for supporting all types of networks, including wide-area transport networks, transport services, data center networks, intra-site service chaining (including VNF chains) and residential IP services.

  - **Evolution over Time**

  The SDN Architecture should focus on fully supporting specific identified usage scenarios instead of poorly supporting fractions of a larger set. Additional usage scenarios can be supported at a later date.

- **Interworking**

  Deployability of SDN is facilitated by remaining compatible with existing technologies where possible.

- **Scalability**
  - **Scalability of the SDN Controller plane:**

    Although logically centralized, a single SDN Controller instance can be implemented as a distributed system, spanning multiple physical platforms, in order to improve scalability and availability.

  - **Scalability within a Network Element:**

    A Network Element may be implemented as a distributed system, potentially spanning a heterogeneous set of processors ASICs, with internal load balancing , cascading
  - **Scalability specific to individual functions/capabilities:**

    Which functions are implemented in a fast vs. a slow path may differ for various elements, and the supported capacities may differ, therefore standards need to make provision for a wide range of capabilities/footprints and permit negotiation of feature sets and associated parameters at initial configuration time or even at arbitrary times.
- **Security**

  When security is discussed in the context of SDN, it can refer to the following
  - The use of SDN to implement network security
  - The security of SDN infrastructure and mechanisms itself

    Security concerns pertaining to the SDN infrastructure itself (e.g. SDN Controller plane traffic) can be grouped into the following categories:
    - ❖ Authentication of communicating entities (e.g. SDN Controllers and Network Elements).

- ❖ Access Control by enforcing configured policies that govern rights for each entity.
- ❖ Privacy by encrypting communication channels using protocols like (D)TLS/IPsec.
- ❖ Auditing by maintaining and permitting access to applicable records, for example records that capture the identity of the initiator and/or the details of an operation.
- ❖ Denial of Service Mitigation by policing (metering).

- **Resilience and Fault Tolerance**

- **Error Handling**

The SDN Architecture needs to define in general terms how error conditions need to be handled. Error conditions should be handled as gracefully as possible, with erroneous inputs on any Interface.

- **High Availability Support**

The SDN Architecture needs to make provision for implementing highly available networks.

This implies supporting redundancy of links and Network Elements (i.e. for the data plane) and of SDN Controllers (i.e. for the SDN Controller plane).

- **Management and Monitoring**

The SDN Architecture needs to take into account the management of the SDN system itself, for example by considering arrangements for pairing Network Elements and SDN Controllers as well as by making provision for performance monitoring of the SDN Controller/Network Element interfaces.

3.3 UST Migration Plan:

In this section we are going to list the steps required to migrate UST network to SDN:

3.3.1 Starting Network :

A detailed description for the current UST network:

3.3.1.1 General Description :

UST consists of three Buildings :

1- Medical building.
2- Engineering building.
3- Administration seances building.

UST Network is designed based on Cisco Three-tier Hierarchical Network Model

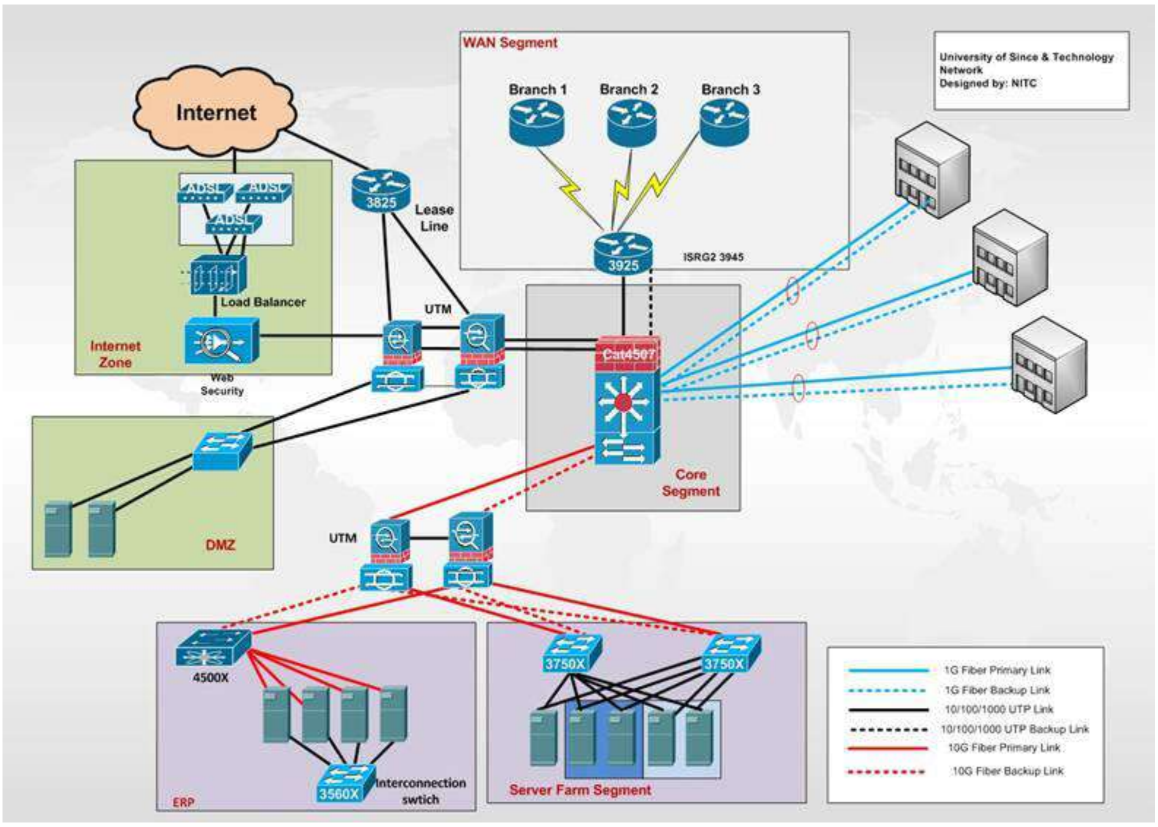


Figure 3.1: UST Network

1- Access   layer :

in each building there are several switches Which is called access switches , in each floor , there is one switch per Lab and one switch for staffs . these access switches connect End Devices like labs' Devices and staffs' Devices , Printers and Access points .

Access points are devices that connect wireless End Devices like cell phones and Laptop to access switches , there are about six Access points in each floor.

2- **Distributed layer :**

In each building there is one Distributed switche that connects access switches to the core switch trough fiber optic and there are multiple VLANs (Virtual LAN) listed as follows :

1- 1 VLAN dedicated to each floor of the building.

2- 1 VLAN dedicated to each access point.

3- 2 VLANs inside access point One for staffs and one for students

4- 1 VLAN dedicated for each Lab .

5- 1 VLAN for Management .

3- **Core Layer**

In the main computer center there is device Which is called Core switch that connects Distributed switches with each other, the Distributed switches are connected to Core switch through two

cables :

1- 1G Fiber primary link.

2- 1G Fiber Backup link.

Core switch are connected to DMZ (Demilitarized Zone) Servers through UTM (Unified Threat Management) Devices by 1000 UTP Link and DMZ , Core switch are connected to ERP (Enterprise resource planning) Servers and Server Farm Segment through UTM (Unified Threat Management) Devices by two Cables :

1- 10G Fiber primary link
2- 10G Fiber Backup link.

Internet Zone :

Internet Zone consists of load Balancer with ADSL Modems .

WAN segment :

Branches Routers are connected to the main Branch Router over MPLS -WAN Technology .We well exclude this segment from migration ,because it needs SD-WAN technology that is out of our scope.

3.3.1.2 Redundancy Model:

In UST Network there devices and cables Which acts as a backup or Redundant Which we make clear through:

1- There are 1G Fiber Backup links from Distributed Switches to Core Switch
2- There is UTM Backup for Duplicate connection to DMZ and Internet Zone.
3- There is UTM Backup for Duplicate connection to ERP and Server Farm Segment.
4- There are 10G Fiber Backup links from Core Switch to UTM and from UTM to ERP and Server Farm Segment Switches .
5- There is Switch Backup in Server Farm Segment.

3.3.1.3 Problems and Challenges

Problems and challenges that are facing UST network are listed in chapter one under problem statement section.

3.3.2 Target Software-Defined Network

The target UST Network is to create a programmable network that the controller can control the network through instructions go over openflow protocol

The target network well be three layers:

1- **Data plane layer**: by using new OpenFlow-enabled switches or adding OpenFlow support to their existing switches. And all device like (Switches , Routers, Firewalls, LoadBalancers and others …)

2- **Controller platform :** control all switches and decide how to forword packets between openflow switches .it acts as the mind of the whole network

2 - **Applications or Innovation**: making the network programmable by building or using APIs to serve a certain purpose .

3.3.2.1 Objectives

The aim of this project is to migrate UST network to SDN network for these reasons:

1 – To break venders monopoly and make the network work with multiple devices from different vendors

2 – Make the network programmable

3 – To reduce administration complexity . So configure the whole network from the controller

4- To facilitate troubshooting process by making all network configuration in one device

5– To improve Network performance .

3.3.2.2 Migration Approach

We are going to use the phased approach to migrate UST network to SDN .We well use this approach to make a way for rollback and to minimize the risks during migration. Stanford project also used this approach.

We well implement that at distributed layer by moving VLANs , servers and access switches through the following phases:

1 – Add OpenFlow support on switch : by updating the firmware to support openflow

2 - Verify OpenFlow support on switch : by connecting test hosts and test VLANs to the switch and make them managed by the controller.

3- Migrate VLANs , servers and access switches to a new network , this is done by :

1- creating a new subnet.

2 – move servers and access switches to the new subnet.

3 - Verify reachability within new subnet.

4 - Enable OpenFlow for new subnet: this is done by configuring the controller . Then verifying that the subnet with its servers and access switches are managed by the controller

3.3.2.3 SDN Architecture

We are going to reorganize the current UST network to be SDN and provide a supposed design for the new network architecture . We are going to replace or update the switches to be Openflow supported switches .The SDN architecture of the new network well consist of three layers:

1 – Application layer : well include the applications and APIs that well set the instructions to controller . It well also include programs that are innovated to serve network requirements

2 – Controller Platform : This layer well control the data plan layer according to the instructions from the application layer . It well consists of a primary controller located on an Ubuntu server , and an other secondary controller located on an other server .

3 – Data plan layer : This layer well forward the data flow to paths which the controller tell

We well explain the target UST SDN design in details in design chapter .

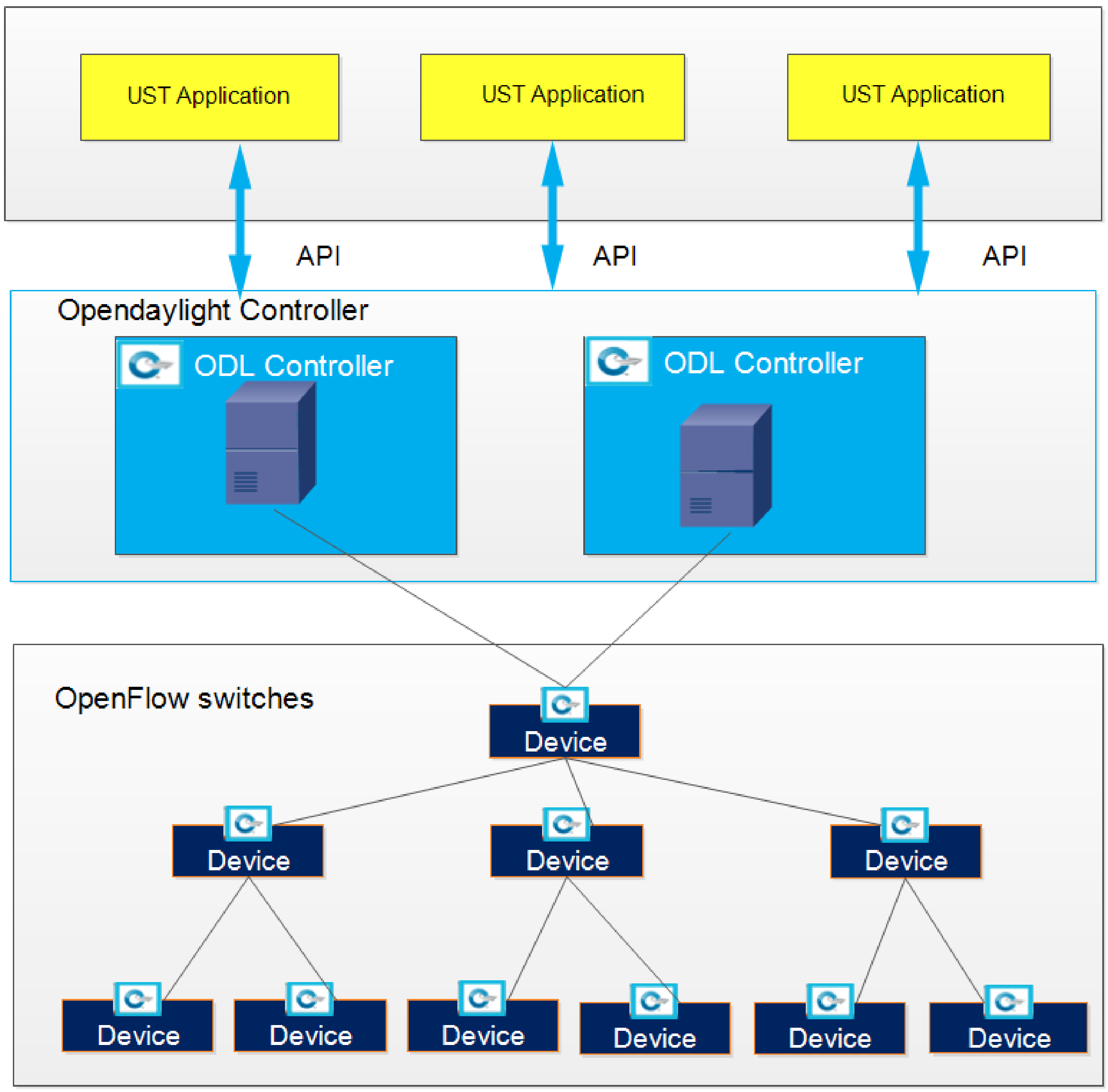


Figure 3.1: UST SDN Architecture

3.3.2.4 Dependencies Target

Here we are going to mention what controller and tools the target network depends on:

1 - The target network well be controlled by OpenDaylight controller .

2- The switches well be Openflow supported

3 - To test the network performance we are going to use some tools like WireShark

4 – For management , Solarwinds and OFM well be used

3.3.2.5 Migration Procedures

We are going to list the steps that are required to implement UST migration :

1- Install Java platform on an Ubuntu server

2 – Install ODL controller

3 – export java to ODL controller

3- Install features that provides topologies , APIs and openflow switches on ODL controller

4 – Install a GUI to ODL

5 – connecting switches to the controller as the topology provided in design chapter .

6 – configure the network from controller to be connecting correctly

7 – creating VLANs from the controller and adding hosts to their VLANs

8 – link Solarwinds the controller

9– Testing the network reachability

10 – Testing Network performance

These steps well be explained in details in implementation chapter later

3.3.2.6 Post-Migration Acceptance

After migrating the UST network , we are going to test the network under several cases for ensuring that the network provides the expected services and performance . That well be done implementation chapter later .

3.3.2.8 Skill Sets Requirements

To implement UST migration to SDN , The team must have some essential skills in the following fields :

1 – Linux Administration

1 – Python programing

2 – Routing and Switching

3 – Java programing

3.4 Migration plan Comparison:

Table 3.1: Starting Network Migration plan Comparison

Starting Network	**Google Inter-Datacenter WAN**	**NTT Provider Edge**	**Stanford Campus Network**	**UST Migration**
General Description	√	√	√	√
Operational Mode	√	√		
***Deployed Equipment*:**	√	√		
Redundancy Model		√		√
Management Tools	√	√		
Network Capacity	√	√		
Problems and Challenges		√		√
Pre-Migration Assessment	√	√		

Table 3.2: Target Software-Defined Network Migration plan

Target Software-Defined Network	Google Inter-Datacenter WAN	NTT Provider Edge	Stanford Campus Network	UST Migration
Objectives	√	√	√	√
Migration approach	√	√	√	√
SDN Architecture	√	√	√	√
Dependencies Target	√	√	√	√
GAP Analysis	√	√	√	
Migration Procedures	√	√		√
Post-Migration Acceptance	√	√	√	√
Migration Timeline			√	
Skill Sets Requirements			√	√

Chapter Four

Design

4.1 Design

In this chapter we are going to make a design that suppose UST network after migration to SDN Virtualization .

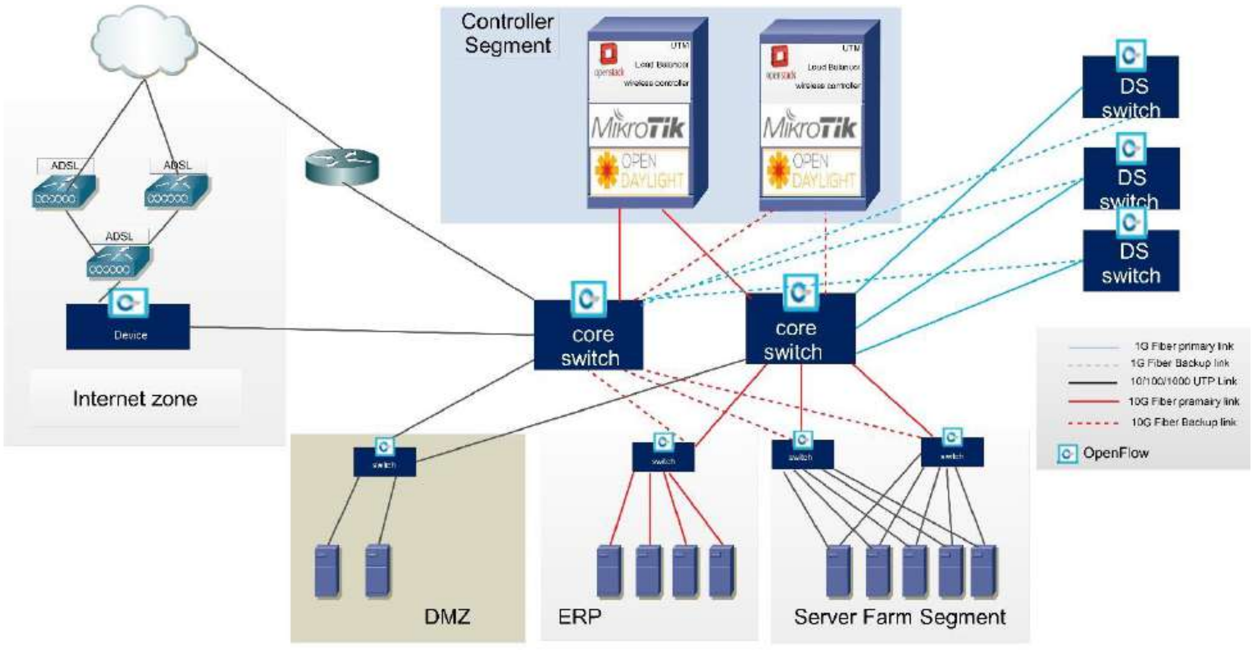


Figure 4: Design UST network with SDN and NFV

As we see in the figure above there are several changes that well be implemented in the existing network to make the migration to SDN and Virtualization :

First : Upgrading all distributed and core switches to openflow-supported switches.

Second : Due to the importance and critical functions of core switch , we propose to add a secondary openflow-supported core switch to support redundancy . The second core switch well be chosen from any vender with consider of the abilities of hardware .

Third : We propose to add two power servers With Ubuntu server Linux (or any servers distribution of Linux OSs) each one is redundant for the other, these servers have a hypervisor (Vmware) to hold the controller(OpenDayLight) and the framework of virtualization (OpenStack or MikroTik) . These servers well be located in SDN farm in the data center .

Fourth : Any Application or a network written program or script well be added on the ODL controller .

Fifth: We propose to replace the UTM objects of the existing network with a virtualized one located on the framework of virtualization (OpenStack or MikroTik) that's installed on the hypervisor (Vmware).

4.2 Costs of added Proposed objects :

1 – One core Switches

Cisco Catalyst switch = 4299$.

2 – Two Power servers

DELL , 128 GB RAM , **4 x Dell 146GB 10k 2.5" drives ,** US $4,487.6 * 2 = US $8975.2   .

3 – Vmware License =148.7 $

4 – Virtualized Firewall "free "

5 – MikroTik License = 50$

6 – OpenStack "free

7 – OpenDayLight "free "

8 – Ubuntu Server Linux "free "

9 – OpenflowManager application (OFM) "free "

Chapter five

Implementation

Implementation of migration to SDN and NFV :

In this chapter we are going to show the implementation of migration to SDN and NFV:

We make an emulation and virtualization for the SDN and NFV of UST topology

5.1 SDN implementation :

We used OpenDaylight for controller and mininet and mininedit for emulation infrastructure for OpenFlow switches and we used OpenFlowManager (OFM) as application on OpenDaylight controller to control the flow entry we'll describe how install them

5.1.1 Install and Configuring OpenDaylight controller :

To install OpenDaylight controller we need to VMware program for install ubuntu linux on virtual machine . after install ubuntu linux 14.4 on virtual machine We install OpenDaylight Through these steps:

1. Download last version of OpenDaylight zip file from https://www.opendaylight.org/downloads to directory that was created .
2. prerequisite: JVM 1.7+ (JAVA_HOME should be set to environment) .
3. Unzip the zip file and Navigate to the directory and run **./bin/karaf**.
4. L2Switch Features Installation **: karaf@root>feature:install odl-l2switch-switch-ui**
5. Installing DLUX Features : It is responsible for the Graphical User Interface (GUI) features in the OpenDaylight controller. It can be installed with the following commands: Execute Next, install "**feature:install odl-mdsal-apidocs**", Finally, execute "**feature:install odl-dlux-all**" command.
6. install "**feature:install odl-restconf**" in the OpenDaylight Console to enable STP for preventing loop
7. Navigate to **http://controller-ip:8181** to open the web interface, then use the following credentials to log in: User: **admin** ,Password: **admin.**

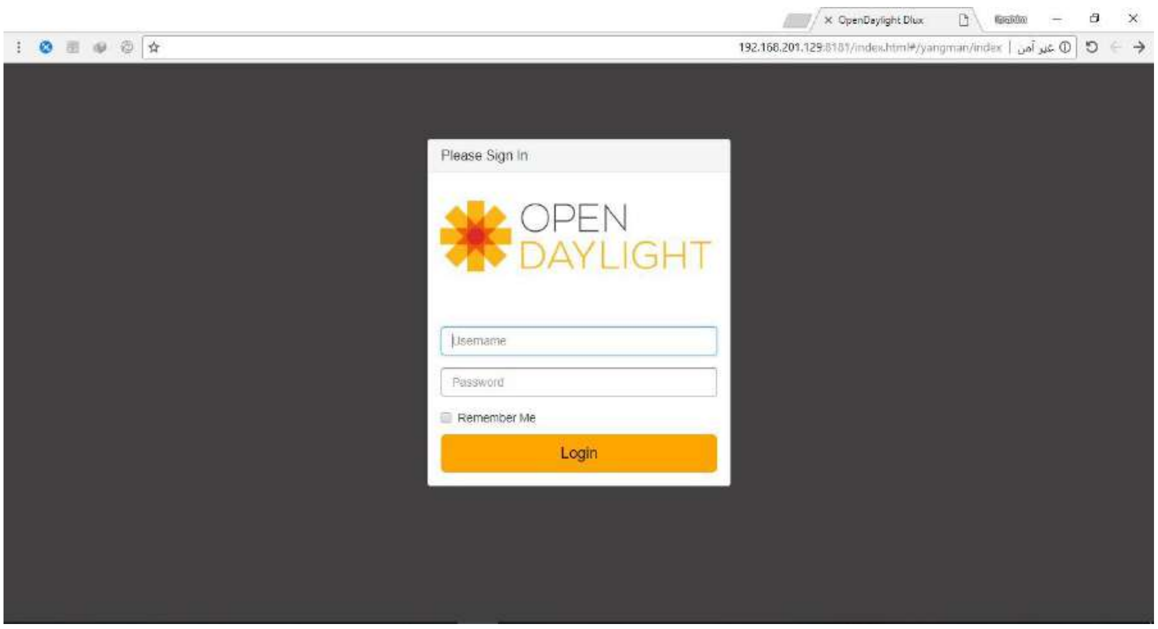


Figure 5.1: OpenDaylight GUI

5.1.2 Installing and Configuring Mininet :

the Mininet is a network emulator. It can create a network of virtual switches, controllers, servers, hosts, links and other network infrastructure devices to work with to install the Mininet , follow the following steps :

a. Clone the source code "**git clone git://github.com/mininet/Mininet**" to Mininet machine.

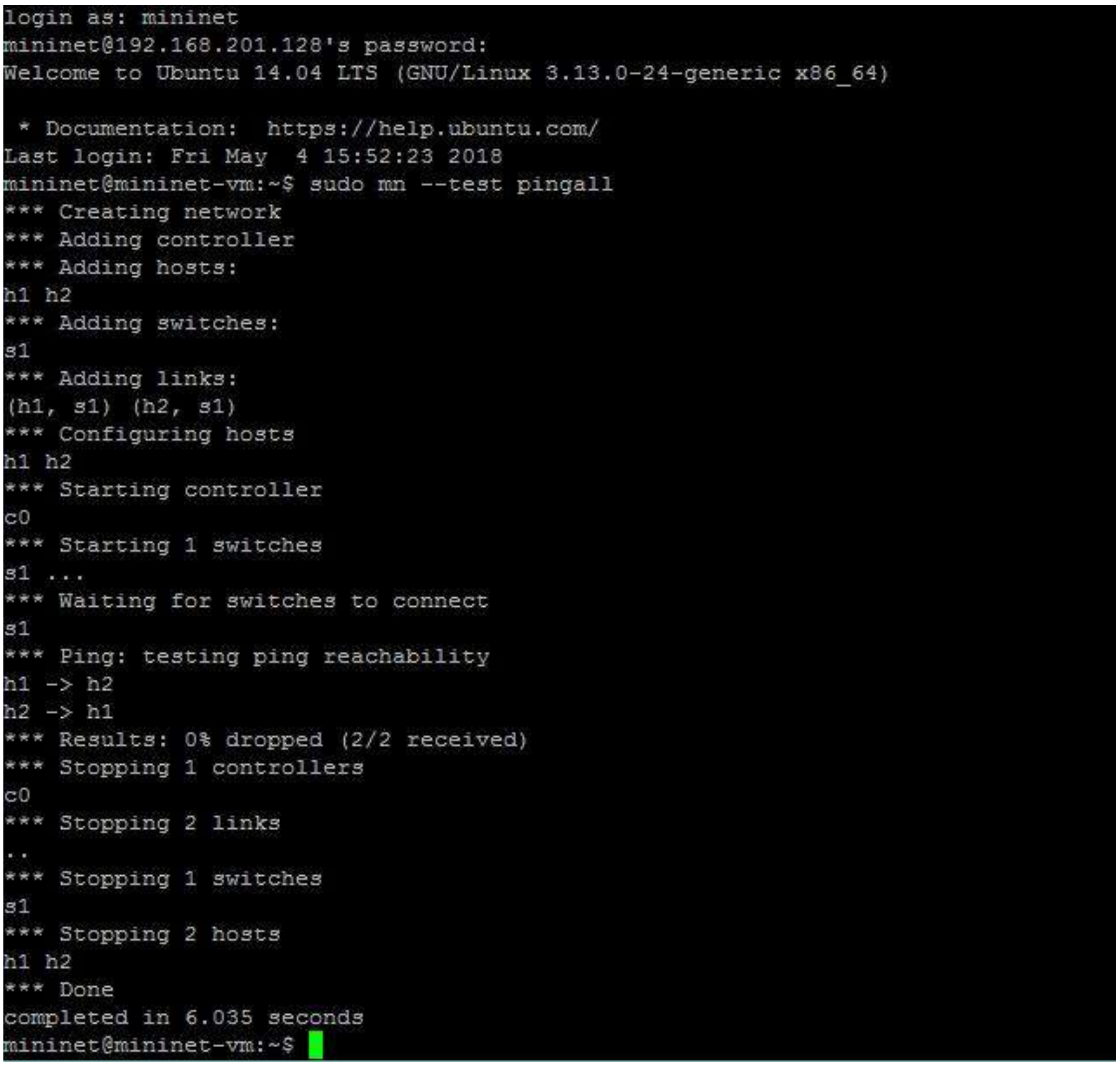
b. In the "Mininet' folder list available versions "**git tag**".

c. Choose the version to be installed "**git checkout –b [VERSION]**".

d. Once the code is fetched, install Mininet "**Mininet/util/install.sh -a**".

The "-a" option will install everything included in the Mininet: dependencies to the openvSwitch, OpenFlow wireshark, POX and other packages.

e. After the installation is completed, verify the basic functionality "**sudo mn --test pingall**". This command will test IP connectivity between the hosts in the network created in Mininet. The result should be the same as shown in figure 5.2 .



```
login as: mininet
mininet@192.168.201.128's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Fri May  4 15:52:23 2018
mininet@mininet-vm:~$ sudo mn --test pingall
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 6.035 seconds
mininet@mininet-vm:~$ 
```

Figure 5.2: Testing Mininet functionality.

After successful installation and testing, it is now possible to connect to the OpenDaylight controller installed in the Installing and Configuring OpenDaylight section. The Mininet network emulator includes MiniEdit, a simple GUI editor for Mininet. MiniEdit is an experimental tool created to demonstrate how Mininet can be extended.

5.1.2.1 Start MiniEdit

To Start MiniEdit first Installing Xming serverWe used Xming server to show the miniedit GUI in windows host

Then run MiniEdit script witch located in Mininet’s examples folder. To run MiniEdit, execute the command:

$ sudo ~/mininet/examples/miniedit.py

as shown in figure 5.3.

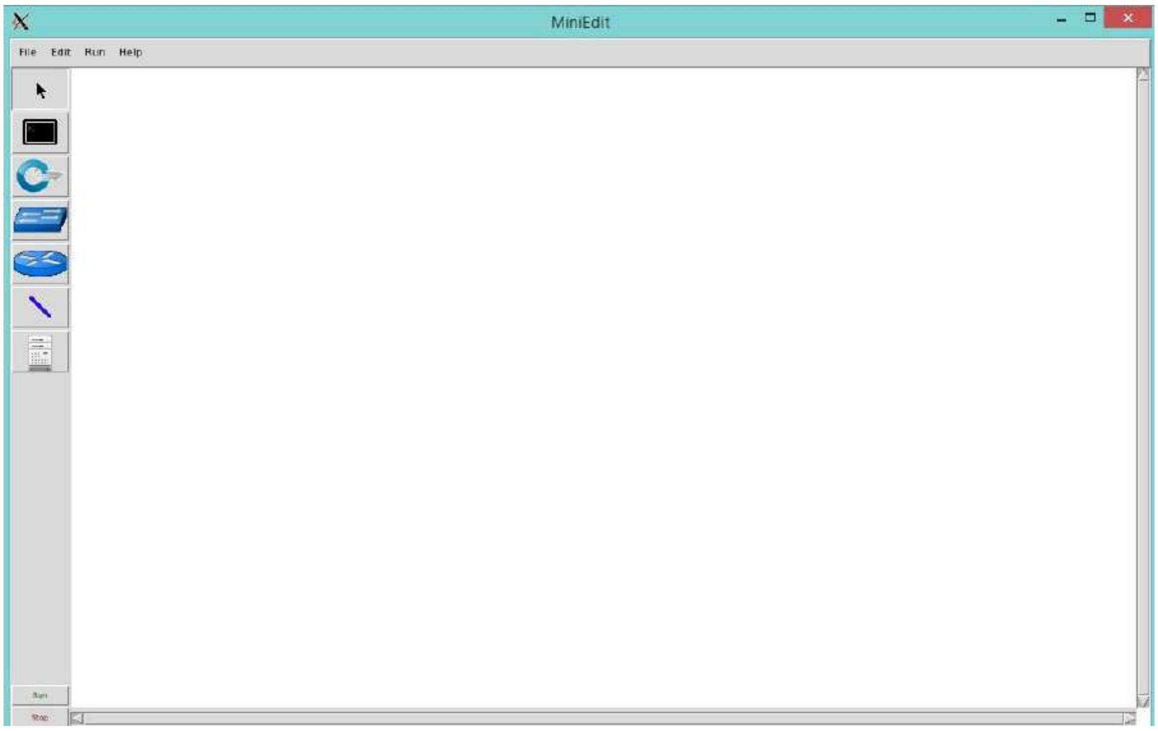


Figure 5.3: Xming server to show the miniedit GUI

5.1.3 Install and Configuring OpenFlowManager (OFM)

To install OpenFlowManager (OFM) on OpenDaylight Through these steps:

1. OFM requires Python to work, so our Ubuntu master instance needs a package called software-properties-common which includes Python libraries:

   ‘’**apt-get install -y bash-completion software-properties-common python-software-properties sudo curl ssh git**”.
2. We're going to configure SSH so that only our authorized_keys will work for SSH and while we're at it, we're going to enable root login:

   “**nano /etc/ssh/sshd_config** ###(change PermitRootLogin to yes)” and restart the ssh service so that they take effect: " **service ssh start**” and generating ssh keys : “**ssh-keygen -t rsa -P** “ .
3. Install Java : First, add Oracle's PPA, then update your package repository.

   “**sudo add-apt-repository ppa:webupd8team/java sudo apt-get update**”

   Then Oracle JDK 8 “**sudo apt-get install oracle-java8-installer**” then Setting the JAVA_HOME Environment Variable to determine the Java installation location “**nano ~/.bashrc export JAVA_HOME="/usr/lib/jvm/java-8-oracle" . ~/.bashrc**” .
4. Install Node.js , Node.js is available from the NodeSource Debian and Ubuntu binary distributions repository : “ **curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -apt-get install nodejs**”.
5. clone a repository openflowmanger from github “**git clone <u>https://github.com/CiscoDevNet/OpenDaylight-Openflow-App.git</u>**”
6. **sed -i 's/localhost/<server_ip>/g' ./OpenDaylight-Openflow-App/ofm/src/common/config/env.module.js** ### Use the IP we noted earlier, for "server_ip"
7. **npm install -g grunt-cli**

8. add this feature on OpenDaylight "**feature:install odl-restconf-all odl-openflowplugin-all odl-l2switch-all odl-mdsal-all odl-yangtools-common**".
9. **http://server ip :9000** to display OFM on browser .

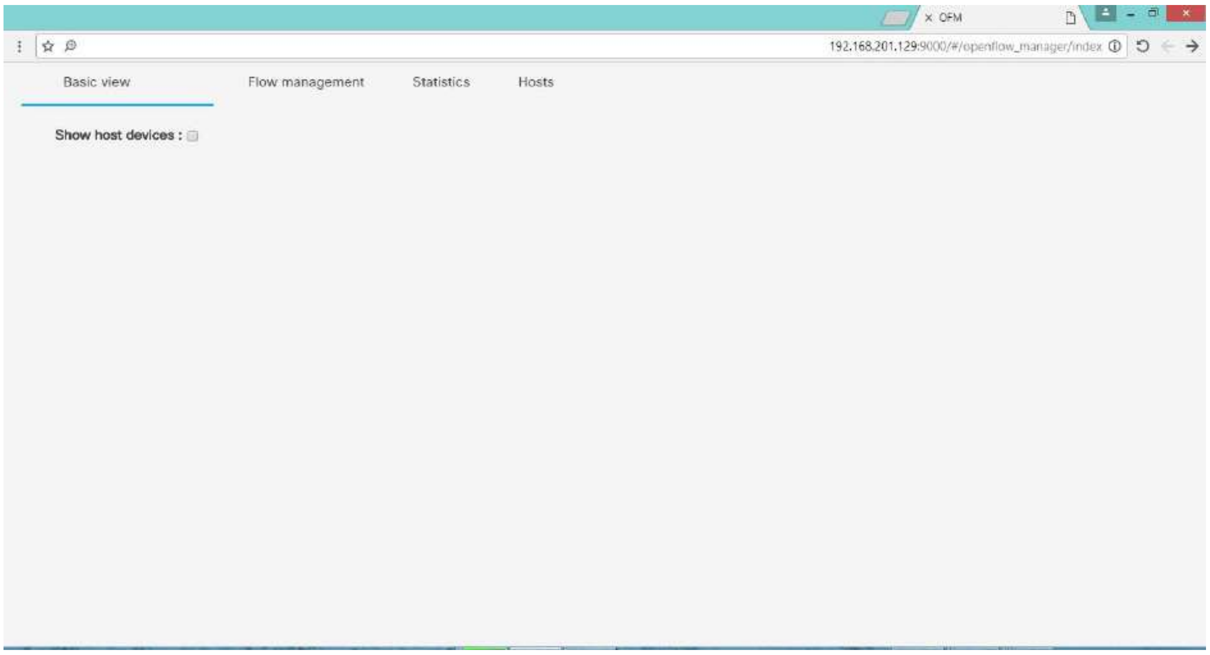
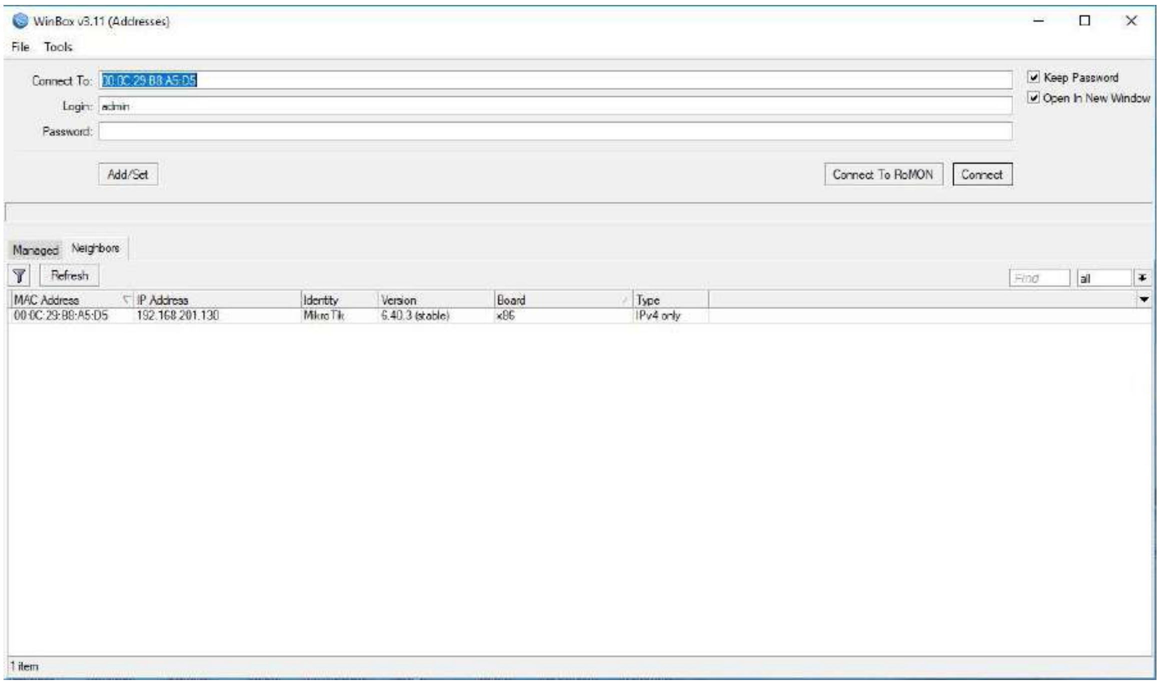


Figure 5.4: OpenFlowManager (OFM)

5.2 Implementation NFV:

Due to high resources that OpenStack requires we used Virtual Mikrotik RouterOS to achieve NFV   and bind it to mininet . and we used Winbox , Winbox is a small utility that allows administration of Mikrotik RouterOS using a fast and simple GUI

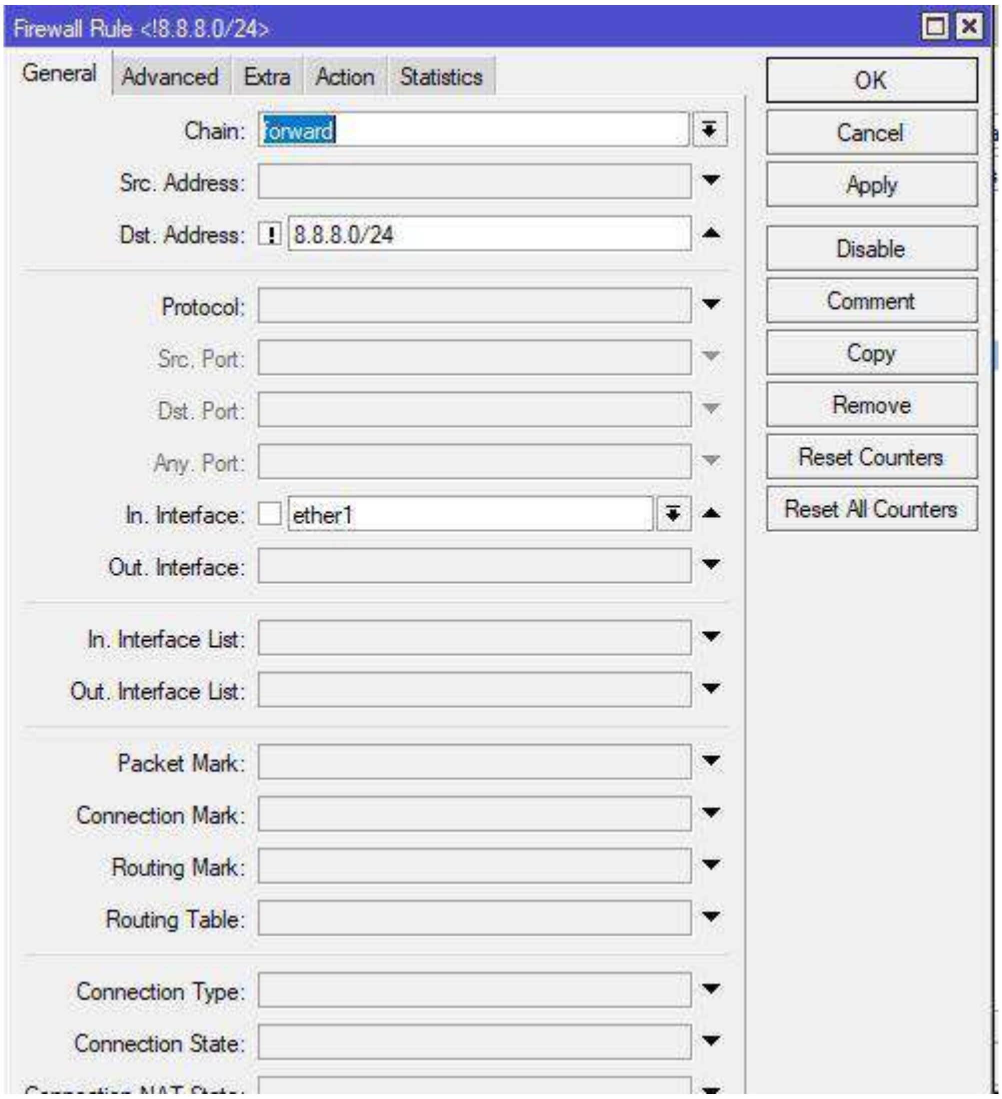


Connect To: 00:0C:29:B8:A5:D5
Login: admin
Password:
[✓] Keep Password
[✓] Open In New Window

MAC Address	IP Address	Identity	Version	Board	Type
00:0C:29:B8:A5:D5	192.168.201.130	MikroTik	6.40.3 (stable)	x86	IPv4 only

Figure 5.5: Open Virtual Mikrotik Router by Winbox

5.2.1 virtualized firewall on Virtual Mikrotik RouterOS



Firewall Rule <!8.8.8.0/24>

Chain:	forward
Src. Address:	
Dst. Address:	! 8.8.8.0/24
Protocol:	
Src. Port:	
Dst. Port:	
Any. Port:	
In. Interface:	ether1
Out. Interface:	
In. Interface List:	
Out. Interface List:	
Packet Mark:	
Connection Mark:	
Routing Mark:	
Routing Table:	
Connection Type:	
Connection State:	

Figure 5.6: firewall in Virtual Mikrotik Router

5.3 SDN and NFV in UST Network

as we mentioned earlier we used mininet for emulation physical openflow switches and we use OpenDaylight controller for control openflow switches . we used virtual mikrotik Router for routing packet on different subnets and put some polices by virtualized Firewall and DNS that are existing in MikroTik RouterOS and we use OFM application for manage flow entry ,We will explain this in detail :

1- Building infrastructure network by MiniEdit

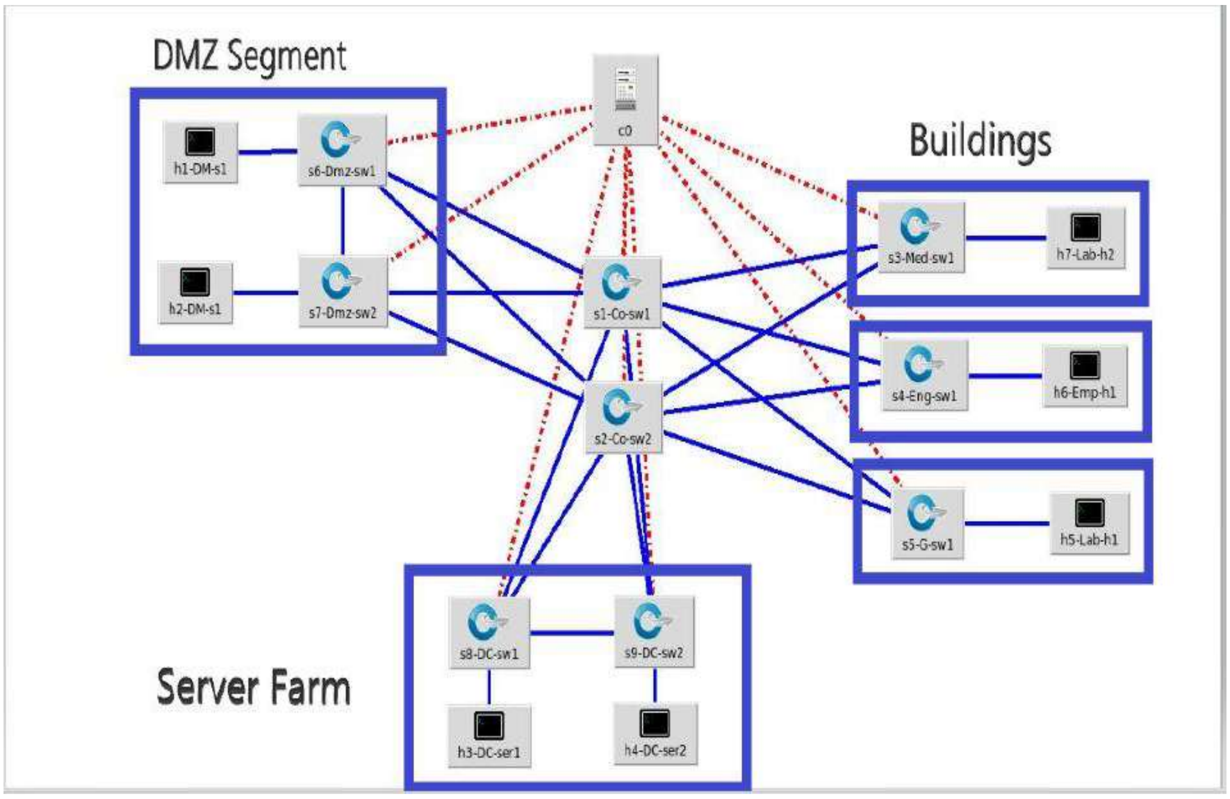


Figure 5.6: UST Network on MiniEdit

2- Bind the openflow switches to opendaylight controller as shown in figure 5.8

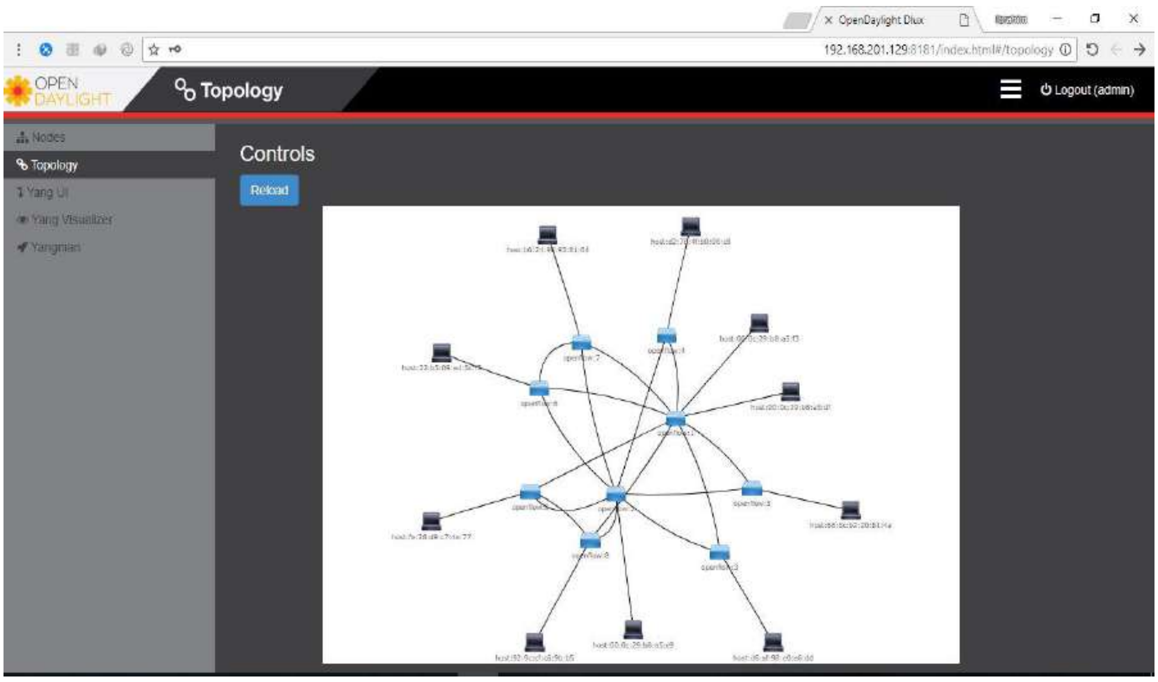


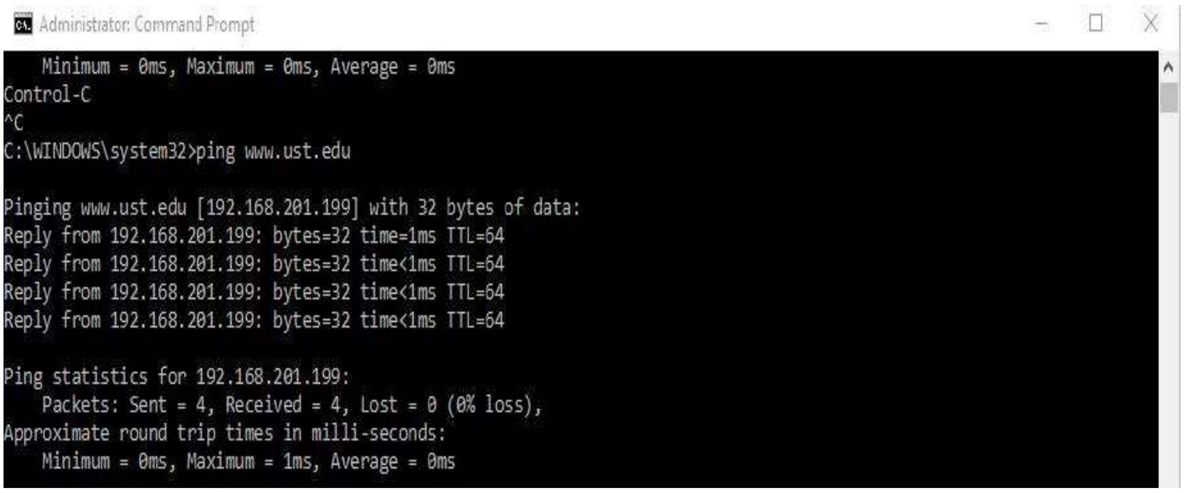
Figure 5.8: UST Network in ODL

3- Bind virtual router to core openflow switch on mininet this was done through :

We add three custom specific virtual network interfaces by VMware application on mininet and same thing on virtual Router and bind this interfaces from core Openflow switches to virtual router interfaces like a physical connection .

4- Policies that are implemented

1 – We have protected the servers farm from unauthorized access by making the DMZ servers the only servers that receives quires from outside the network by DNS resolution :



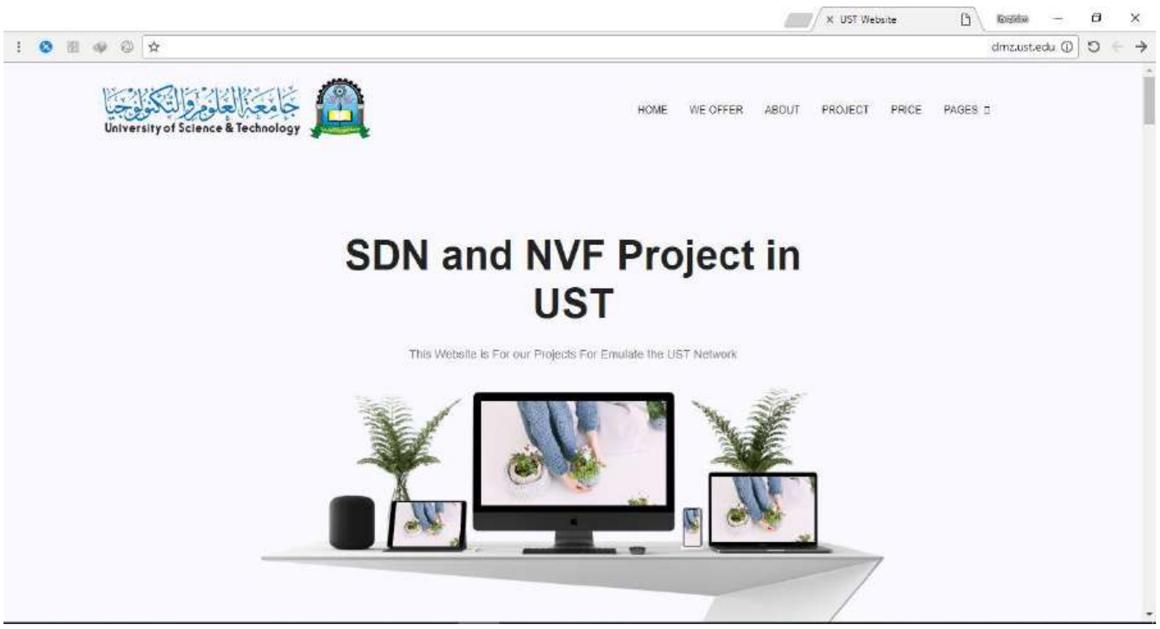
```
Administrator: Command Prompt
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
Control-C
^C
C:\WINDOWS\system32>ping www.ust.edu

Pinging www.ust.edu [192.168.201.199] with 32 bytes of data:
Reply from 192.168.201.199: bytes=32 time=1ms TTL=64
Reply from 192.168.201.199: bytes=32 time<1ms TTL=64
Reply from 192.168.201.199: bytes=32 time<1ms TTL=64
Reply from 192.168.201.199: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.201.199:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
```

Figure 5.11: PING to DMZ servers

2 – We have installed the web server only at the DMZ servers :



SDN and NVF Project in UST

This Website is For our Projects For Emulate the UST Network

Figure 5.12: web server on DMZ servers

3 – The DMZ servers are the only that can access the server farm.

5- We used OFM to manage flows through the network

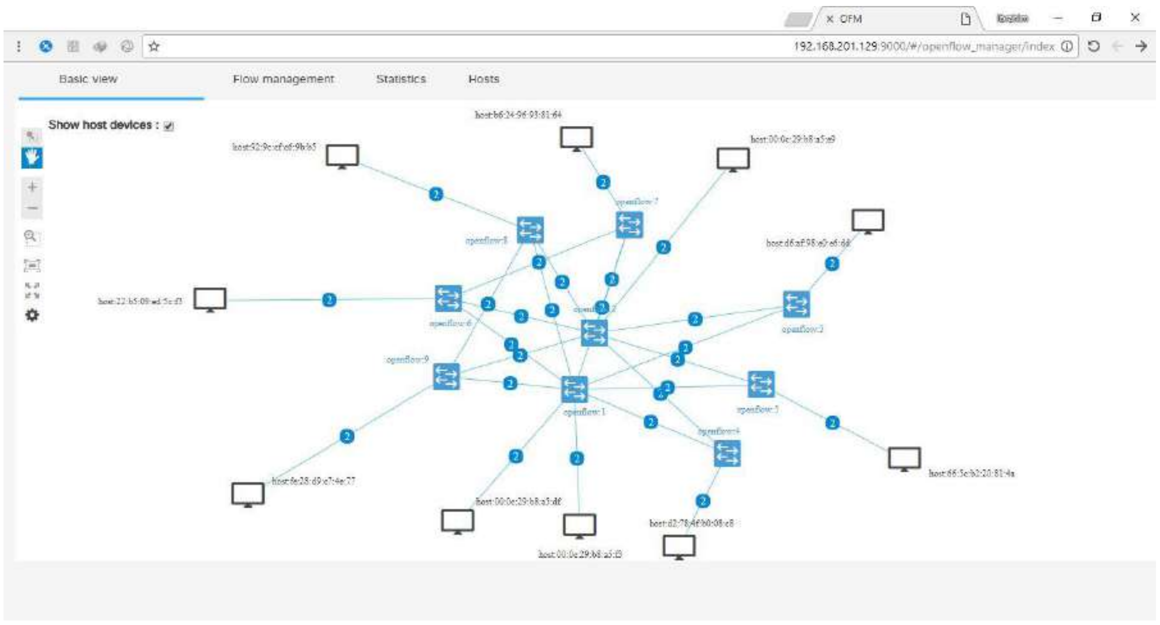
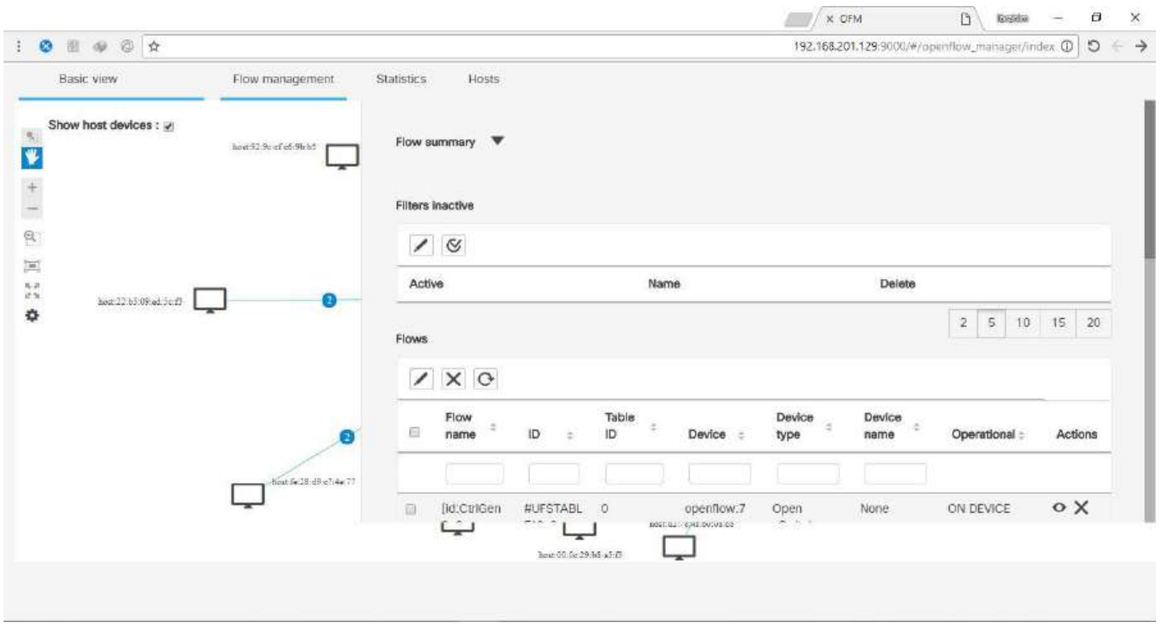


Figure 5.9: UST Network in OFM



Flow summary

Filters inactive

Active	Name	Delete

2 5 10 15 20

Flows

	Flow name	ID	Table ID	Device	Device type	Device name	Operational	Actions
	[id:CtrlGen	#UFSTABL	0	openflow:7	Open	None	ON DEVICE	

Figure 5.10: Management UST Network by OFM

5.4 Testing

We generate a traffic from a host in one building to a DMZ server, the size of each packet was 64000 bit .The capturing time was 60 seconds , 30 second with traffic and the other 30 without generated traffic . We compared both throughput and Round Trip Time (TTR) between the SDN topology and the legacy topology .We used Wireshark tool for Testing .

5.5 Results

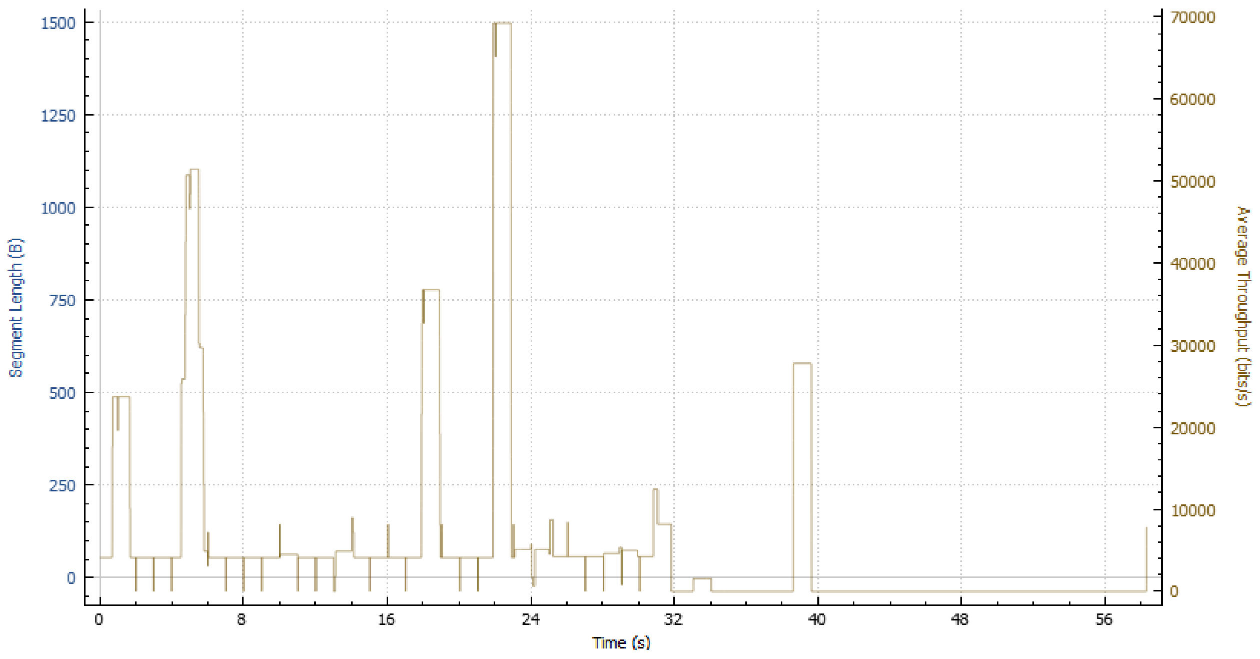


Figure 5.13: Throughput in SDN

The previous figure shows that the Throughput in the SDN network

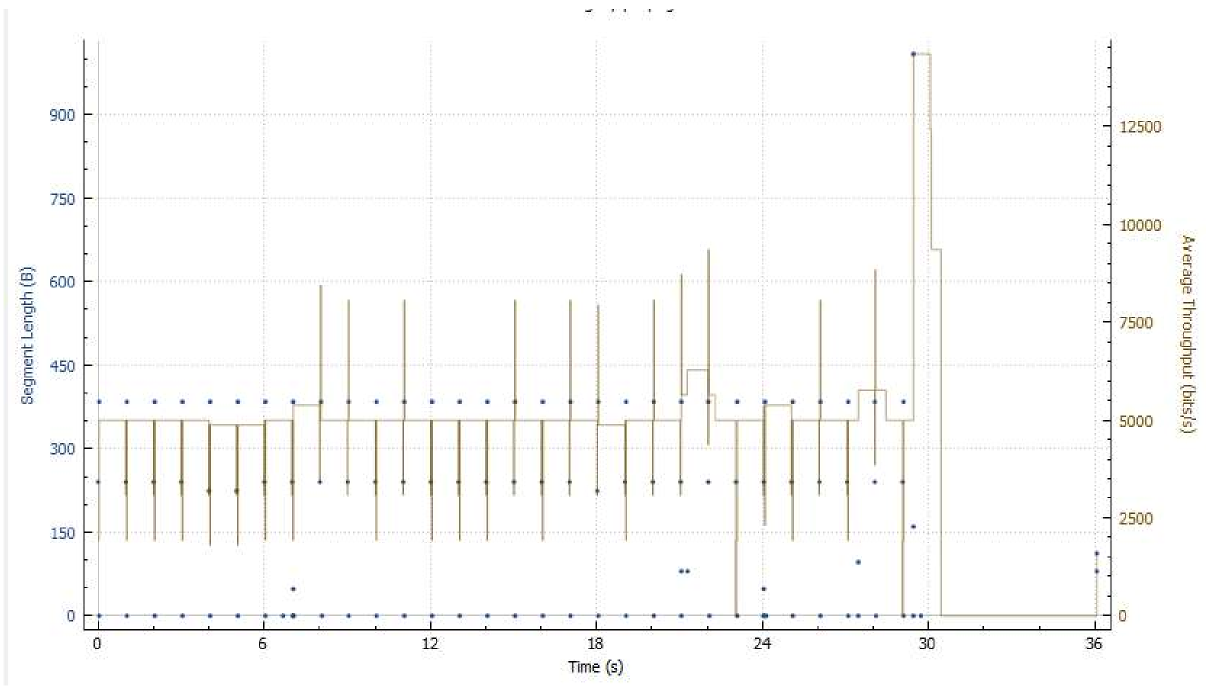


Figure 5.14: Throughput in legacy topology

The previous figure shows that the Throughput in the legacy network.

We can find that throughput in SDN network is quiet higher than throughput in legacy network

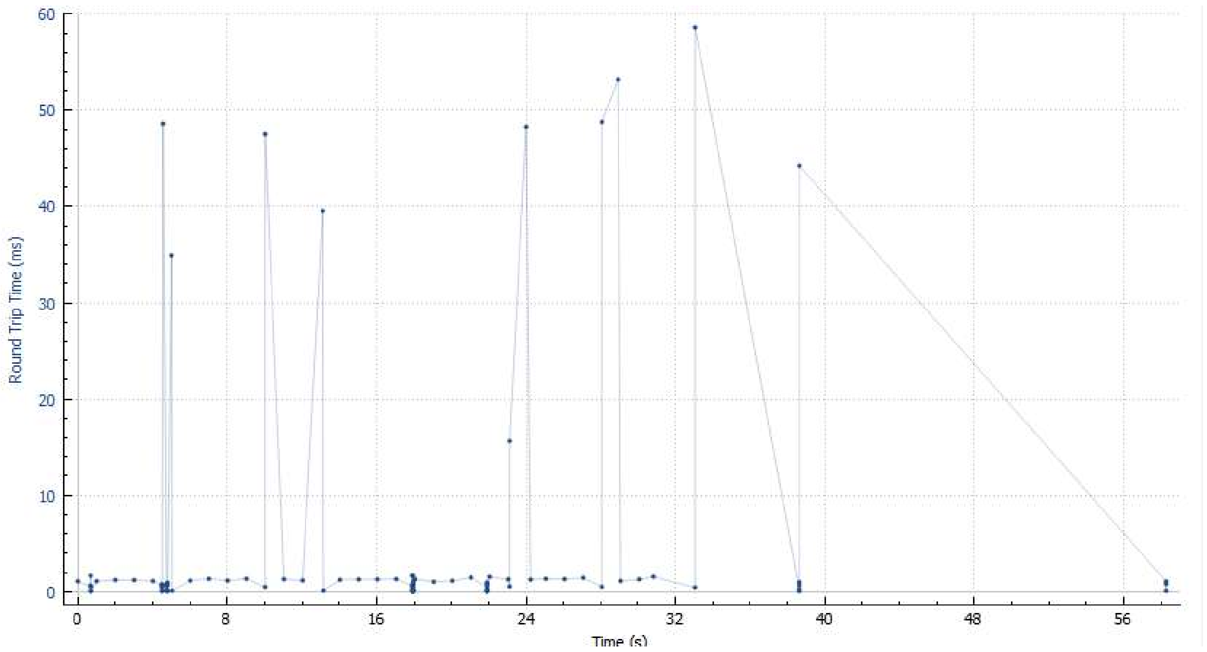


Figure 5.15: RTT in SDN

The previous figure shows that the RTT in the SDN network.

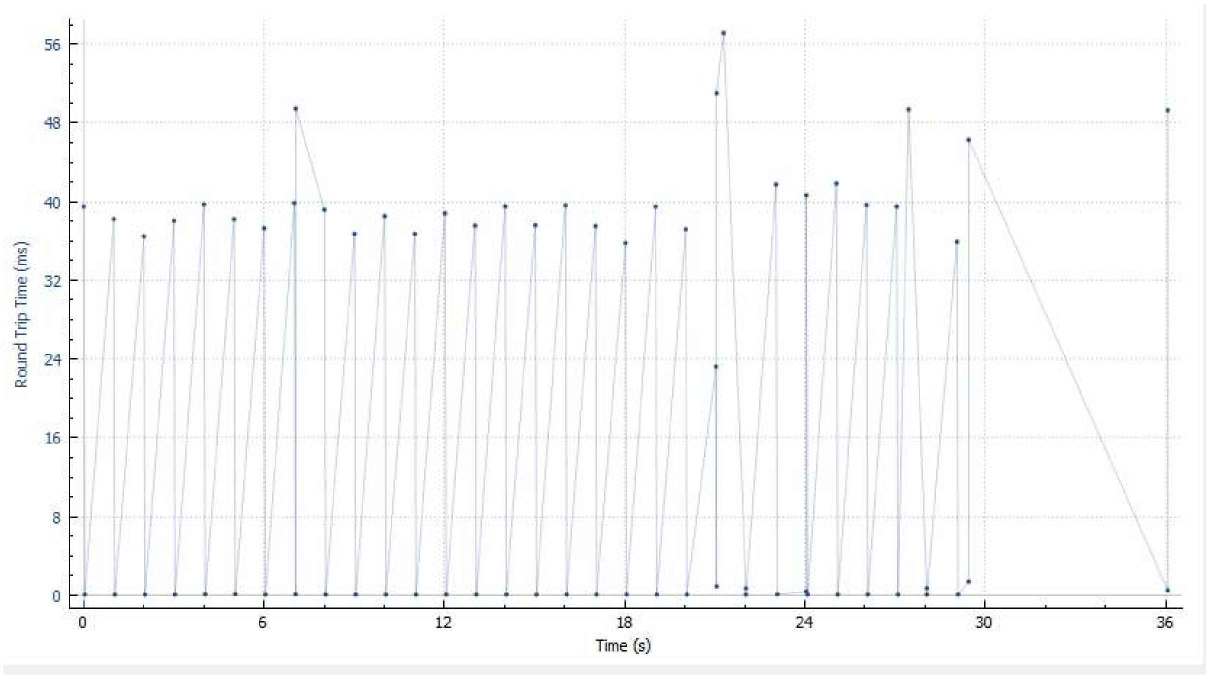


Figure 5.15: RTT in legacy Topology

The previous figure shows that the RTT in the legacy network.

We can find that RTT in SDN is clearly less than RTT in legacy network

Chapter Six

Conclusion & Future Work

6.1 Conclusion

To enhance UST network by enabling programmability , increasing agility , reducing physical network devices and reducing costs , we have migrated UST network to SDN and NFV in a virtualized emulated network .

We have migrated UST network according to ONF steps , and we used mininet to emulate the SDN network .

Then we have made some network devices virtualized (UTMs) to implement NFV . We have done NFV using MikroTik Router OS that includes a lot of network devices .

Also, we have proposed a design that can implement NFV using OpenStack .

Due to high resources that OpenStack requires , we preferred to implement NFV using MikroTik as a project .

We have proved that network can by programmable and controlled by an application by installing an application (OFM) on the controller (ODL) and implementing some scenarios that show that flows can be controlled by OFM application

Also , we have proved that network devices can be virtualized and implement the same functions as physical devices . As we mentioned that We have done that by making the firewall virtualized . We implemented some scenarios that show that the virtualized firewall can restrict and forwards flows .

We have integrated NFV with SDN and we made the virtualized Firewall protects the network that are controlled by the SDN controller (ODL)

We have implanted some real policies that are required and implemented by the existing network in the new network , some of these polices are listed as follws :

1 – We have protected the servers farm from unauthorized access by making the DMZ servers the only servers that receives quires from outside the network

2 – We have installed the web server only at the DMZ servers

3 – The DMZ servers only can access the server farm

6.2 Future Work :

– LTE is one of the most important technologies that has come up to the world recently , We recommend to make a study (or implementation) of SDN in LTE

– Due to programmability of SDN , that will facilitate making the network smart .So the network can predict risks , congestion ..etc and can solve the problems fast or before they happen , that will reduce downtime and can utilize the network perfectly. So we recommend to make a project in Artificial Intelligence in SDN

– The development of network will not be dependent on new hardware products , but it will depend on developing software so Software Engineering field will affect networks directly . We recommend to make a study to provide standers or steps that organize and fit software engineering with SDN

– Due to SDN network will be programmable and network requirements well be achieved by applications , so we recommend to develop an application that performs some network tasks or that adds some abilities to the network

– OpenStack is an advance framework of cloud and virtualization . and it supports a lot of services of virtualization. We highly recommend to use it in virtualization

– SDN and NFV security is a big issue and a big argument , so we recommend to study SDN and NFV security and implement it in a case study (e,g UST network)

– As it is known implementing a network in physical devices is somehow different from implementing it in a emulation program . So , we recommend to take physical network devices and upgrade them to support OpenFlow protocol then implement this project in a real world environment .

References:

1- "Learning OpenDaylight The art of deploying successful networks " , Reza Toghraee ,2017

2- " SDN: Software Defined Network", Thomas D. Nadeau ; Ken Gray , 2013

3- " Software Defined Networking with OpenFlow " , Siamak Azodolmolky ,2013

4- " Software Defined Network DESIGN AND DEPLOYMENT" , Patricia A. Morreale ;James M. Anderson, 2015

5- "Software Defined NetworkingFor Dummies, Cisco Special Edition" , John Wiley ; Sons, Inc, 2015

6- "Software-Defined Networking :The New Norm for Networks", ONF White Paper ,April 13, 2012

7- Software Defined Networking Concepts , Xenofon Foukas ;Mahesh K. Marina ;Kimon Kontovasilis.

8- Foundations of Modern Networking SDN, NFV, QoE, IoT, and Cloud , William Stallings , 2016.

9- "SDN ببساطة" المهندس /عادل الحميدي ، الدكتور/ فؤاد بنعمران،

10- Virtual Extensible LAN (VXLAN) Overview , White Paper , https://www.arista.com/assets/data/pdf/Whitepapers/Arista_Networks_VXLAN_White_Paper.pdf.

11- List of several SDN switches that support Group Table ALL feature by Mohammad Noormohammadpour and Cauligi S. Raghavendra Ming Hsieh Department of Electrical Engineering, University of Southern California.

12- Advancing Software-Defined Networks: A Survey JACOB H. COX, JR., JOAQUIN CHUNG, SEAN DONOVAN, JARED IVEY, RUSSELL J.CLARK (Member, IEEE) GEORGE RILEY , AND HENRY L. OWEN, III (Senior Member, IEEE).

13- .Naga Praveen Katta, Jennifer Rexford, and David Walker. Logic Programming for Software-Defined Networks

14- Mark Reitblatt, Marco Canini, Arjun Guha, and Nate Foster.

15- Fattire: Declarative fault tolerance for software defined networks

16- Andreas Voellmy and Paul Hudak. Nettle: Taking the Sting Out of Programming Network Routers.

17- "Building a Software-Defined Networking System with OpenDaylight Controller",Maksim Sisov , 30 March 2016

18- "Software-Defined Networking :The New Norm for Networks", ONF White Paper ,April 13, 2012

19- Foundations of Modern Networking SDN, NFV, QoE, IoT, and Cloud , William Stallings , 2016

20- "Migration Use Cases and Methods" , Migration Working Group, Open Networking Foundation,

Appendix 1

Building UST network in mininet by Python

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                      controller=RemoteController,
                      ip='192.168.201.129',
                      protocol='tcp',
                      port=6633)

    info( '*** Add switches\n')
    s4-Eng-sw1 = net.addSwitch('s4-Eng-sw1', cls=OVSKernelSwitch)
    s8-DC-sw1 = net.addSwitch('s8-DC-sw1', cls=OVSKernelSwitch)
    s3-Med-sw1 = net.addSwitch('s3-Med-sw1', cls=OVSKernelSwitch)
    s7-Dmz-sw2 = net.addSwitch('s7-Dmz-sw2', cls=OVSKernelSwitch)
    s2-Co-sw2 = net.addSwitch('s2-Co-sw2', cls=OVSKernelSwitch)
    Intf( 'eth2', node=s2-Co-sw2 )
    s9-DC-sw2 = net.addSwitch('s9-DC-sw2', cls=OVSKernelSwitch)
    s1-Co-sw1 = net.addSwitch('s1-Co-sw1', cls=OVSKernelSwitch)
    Intf( 'eth1', node=s1-Co-sw1 )
    Intf( 'eth3', node=s1-Co-sw1 )
    s5-G-sw1 = net.addSwitch('s5-G-sw1', cls=OVSKernelSwitch)
    s6-Dmz-sw1 = net.addSwitch('s6-Dmz-sw1', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')
    h2-DM-s1 = net.addHost('h2-DM-s1', cls=Host, ip='8.8.8.11/24',
defaultRoute='via 8.8.8.1')
    h6-Emp-h1 = net.addHost('h6-Emp-h1', cls=Host, ip='7.7.7.12/24',
defaultRoute='via 7.7.7.1')
    h3-DC-ser1 = net.addHost('h3-DC-ser1', cls=Host, ip='5.5.5.11/24',
defaultRoute='via 5.5.5.1')
    h1-DM-s1 = net.addHost('h1-DM-s1', cls=Host, ip='8.8.8.12/24',
defaultRoute='via 8.8.8.1')
    h4-DC-ser2 = net.addHost('h4-DC-ser2', cls=Host, ip='5.5.5.12/24',
defaultRoute='via 5.5.5.1')
    h5-Lab-h1 = net.addHost('h5-Lab-h1', cls=Host, ip='7.7.7.11/24',
defaultRoute='via 7.7.7.1')
```

```
    h7-Lab-h2 = net.addHost('h7-Lab-h2', cls=Host, ip='7.7.7.13/24',
defaultRoute='via 7.7.7.1')

    info( '*** Add links\n')
    net.addLink(s6-Dmz-sw1, s1-Co-sw1)
    net.addLink(s7-Dmz-sw2, s2-Co-sw2)
    net.addLink(s7-Dmz-sw2, s1-Co-sw1)
    net.addLink(s6-Dmz-sw1, s2-Co-sw2)
    net.addLink(s2-Co-sw2, s8-DC-sw1)
    net.addLink(s1-Co-sw1, s8-DC-sw1)
    net.addLink(s2-Co-sw2, s9-DC-sw2)
    net.addLink(s1-Co-sw1, s9-DC-sw2)
    net.addLink(s1-Co-sw1, s3-Med-sw1)
    net.addLink(s1-Co-sw1, s4-Eng-sw1)
    net.addLink(s1-Co-sw1, s5-G-sw1)
    net.addLink(s2-Co-sw2, s3-Med-sw1)
    net.addLink(s2-Co-sw2, s4-Eng-sw1)
    net.addLink(s2-Co-sw2, s5-G-sw1)
    net.addLink(h1-DM-s1, s6-Dmz-sw1)
    net.addLink(h2-DM-s1, s7-Dmz-sw2)
    net.addLink(s8-DC-sw1, h3-DC-ser1)
    net.addLink(s9-DC-sw2, h4-DC-ser2)
    net.addLink(s3-Med-sw1, h7-Lab-h2)
    net.addLink(s4-Eng-sw1, h6-Emp-h1)
    net.addLink(s5-G-sw1, h5-Lab-h1)
    net.addLink(s6-Dmz-sw1, s7-Dmz-sw2)
    net.addLink(s8-DC-sw1, s9-DC-sw2)

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s4-Eng-sw1').start([c0])
    net.get('s8-DC-sw1').start([c0])
    net.get('s3-Med-sw1').start([c0])
    net.get('s7-Dmz-sw2').start([c0])
    net.get('s2-Co-sw2').start([c0])
    net.get('s9-DC-sw2').start([c0])
    net.get('s1-Co-sw1').start([c0])
    net.get('s5-G-sw1').start([c0])
    net.get('s6-Dmz-sw1').start([c0])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

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

Appendix 2

UST Network configuration in miniedit

```
{
    "application": {
        "dpctl": "",
        "ipBase": "10.0.0.0/8",
        "netflow": {
            "nflowAddId": "0",
            "nflowTarget": "",
            "nflowTimeout": "600"
        },
        "openFlowVersions": {
            "ovsOf10": "0",
            "ovsOf11": "0",
            "ovsOf12": "0",
            "ovsOf13": "1"
        },
        "sflow": {
            "sflowHeader": "128",
            "sflowPolling": "30",
            "sflowSampling": "400",
            "sflowTarget": ""
        },
        "startCLI": "1",
        "switchType": "ovs",
        "terminalType": "xterm"
    },
    "controllers": [
        {
            "opts": {
                "controllerProtocol": "tcp",
                "controllerType": "remote",
                "hostname": "c0",
                "remoteIP": "192.168.201.129",
                "remotePort": 6633
            },
            "x": "661.0",
            "y": "79.0"
        }
    ],
    "hosts": [
        {
            "number": "3",
            "opts": {
                "defaultRoute": "5.5.5.1",
                "hostname": "h3-DC-ser1",
                "ip": "5.5.5.11/24",
                "nodeNum": 3,
                "sched": "host"
            },
            "x": "515.0",
            "y": "614.0"
        },
        {
            "number": "5",
            "opts": {
```

```
            "defaultRoute": "7.7.7.1",
            "hostname": "h5-Lab-h1",
            "ip": "7.7.7.11/24",
            "nodeNum": 5,
            "sched": "host"
        },
        "x": "1156.0",
        "y": "435.0"
    },
    {
        "number": "6",
        "opts": {
            "defaultRoute": "7.7.7.1",
            "hostname": "h6-Emp-h1",
            "ip": "7.7.7.12/24",
            "nodeNum": 6,
            "sched": "host"
        },
        "x": "1151.0",
        "y": "311.0"
    },
    {
        "number": "4",
        "opts": {
            "defaultRoute": "5.5.5.1",
            "hostname": "h4-DC-ser2",
            "ip": "5.5.5.12/24",
            "nodeNum": 4,
            "sched": "host"
        },
        "x": "692.0",
        "y": "612.0"
    },
    {
        "number": "2",
        "opts": {
            "defaultRoute": "8.8.8.1",
            "hostname": "h2-DM-s1",
            "ip": "8.8.8.11/24",
            "nodeNum": 2,
            "sched": "host"
        },
        "x": "201.0",
        "y": "241.0"
    },
    {
        "number": "1",
        "opts": {
            "defaultRoute": "8.8.8.1",
            "hostname": "h1-DM-s1",
            "ip": "8.8.8.12/24",
            "nodeNum": 1,
            "sched": "cfs"
        },
        "x": "207.0",
        "y": "123.0"
    },
    {
        "number": "7",
        "opts": {
            "defaultRoute": "7.7.7.1",
            "hostname": "h7-Lab-h2",
            "ip": "7.7.7.13/24",
            "nodeNum": 7,
            "sched": "host"
        },
        "x": "1152.0",
        "y": "195.0"
    }
],
"links": [
    {
        "dest": "s1-Co-sw1",
        "opts": {},
        "src": "s6-Dmz-sw1"
    },
    {
        "dest": "s2-Co-sw2",
        "opts": {},
        "src": "s7-Dmz-sw2"
    },
    {
        "dest": "s1-Co-sw1",
        "opts": {},
        "src": "s7-Dmz-sw2"
    },
    {
        "dest": "s2-Co-sw2",
        "opts": {},
        "src": "s6-Dmz-sw1"
    },
    {
        "dest": "s8-DC-sw1",
        "opts": {},
        "src": "s2-Co-sw2"
    },
    {
        "dest": "s8-DC-sw1",
        "opts": {},
        "src": "s1-Co-sw1"
    },
    {
        "dest": "s9-DC-sw2",
        "opts": {},
        "src": "s2-Co-sw2"
    },
    {
        "dest": "s9-DC-sw2",
        "opts": {},
        "src": "s1-Co-sw1"
    },
    {
        "dest": "s3-Med-sw1",
        "opts": {},
        "src": "s1-Co-sw1"
    },
    {
        "dest": "s4-Eng-sw1",
        "opts": {},
        "src": "s1-Co-sw1"
    },
    {
            "dest": "s5-G-sw1",
            "opts": {},
            "src": "s1-Co-sw1"
        },
        {
            "dest": "s3-Med-sw1",
            "opts": {},
            "src": "s2-Co-sw2"
        },
        {
            "dest": "s4-Eng-sw1",
            "opts": {},
            "src": "s2-Co-sw2"
        },
        {
            "dest": "s5-G-sw1",
            "opts": {},
            "src": "s2-Co-sw2"
        },
        {
            "dest": "s6-Dmz-sw1",
            "opts": {},
            "src": "h1-DM-s1"
        },
        {
            "dest": "s7-Dmz-sw2",
            "opts": {},
            "src": "h2-DM-s1"
        },
        {
            "dest": "h3-DC-ser1",
            "opts": {},
            "src": "s8-DC-sw1"
        },
        {
            "dest": "h4-DC-ser2",
            "opts": {},
            "src": "s9-DC-sw2"
        },
        {
            "dest": "h7-Lab-h2",
            "opts": {},
            "src": "s3-Med-sw1"
        },
        {
            "dest": "h6-Emp-h1",
            "opts": {},
            "src": "s4-Eng-sw1"
        },
        {
            "dest": "h5-Lab-h1",
            "opts": {},
            "src": "s5-G-sw1"
        },
        {
            "dest": "s7-Dmz-sw2",
            "opts": {},
            "src": "s6-Dmz-sw1"
        },
        {
            "dest": "s9-DC-sw2",
            "opts": {},
            "src": "s8-DC-sw1"
        }
    ],
    "switches": [
        {
            "number": "7",
            "opts": {
                "controllers": [
                    "c0"
                ],
                "hostname": "s7-Dmz-sw2",
                "netflow": "0",
                "nodeNum": 7,
                "sflow": "0",
                "switchIP": "",
                "switchType": "default"
            },
            "x": "359.0",
            "y": "240.0"
        },
        {
            "number": "3",
            "opts": {
                "controllers": [
                    "c0"
                ],
                "hostname": "s3-Med-sw1",
                "netflow": "0",
                "nodeNum": 3,
                "sflow": "0",
                "switchIP": "",
                "switchType": "default"
            },
            "x": "976.0",
            "y": "194.0"
        },
        {
            "number": "8",
            "opts": {
                "controllers": [
                    "c0"
                ],
                "hostname": "s8-DC-sw1",
                "netflow": "0",
                "nodeNum": 8,
                "sflow": "0",
                "switchIP": "",
                "switchType": "default"
            },
            "x": "515.0",
            "y": "527.0"
        },
        {
            "number": "6",
            "opts": {
                "controllers": [
                    "c0"
                ],
                "hostname": "s6-Dmz-sw1",
                "netflow": "0",
            "nodeNum": 6,
            "sflow": "0",
            "switchIP": "",
            "switchType": "default"
        },
        "x": "358.0",
        "y": "121.0"
    },
    {
        "number": "4",
        "opts": {
            "controllers": [
                "c0"
            ],
            "hostname": "s4-Eng-sw1",
            "netflow": "0",
            "nodeNum": 4,
            "sflow": "0",
            "switchIP": "",
            "switchType": "default"
        },
        "x": "980.0",
        "y": "310.0"
    },
    {
        "number": "1",
        "opts": {
            "controllers": [
                "c0"
            ],
            "externalInterfaces": [
                "eth1",
                "eth3"
            ],
            "hostname": "s1-Co-sw1",
            "netflow": "0",
            "nodeNum": 1,
            "sflow": "0",
            "switchIP": "",
            "switchType": "default"
        },
        "x": "657.0",
        "y": "241.0"
    },
    {
        "number": "5",
        "opts": {
            "controllers": [
                "c0"
            ],
            "hostname": "s5-G-sw1",
            "netflow": "0",
            "nodeNum": 5,
            "sflow": "0",
            "switchIP": "",
            "switchType": "default"
        },
        "x": "983.0",
        "y": "435.0"
    },
    {
            "number": "9",
            "opts": {
                "controllers": [
                    "c0"
                ],
                "hostname": "s9-DC-sw2",
                "netflow": "0",
                "nodeNum": 9,
                "sflow": "0",
                "switchIP": "",
                "switchType": "default"
            },
            "x": "691.0",
            "y": "526.0"
        },
        {
            "number": "2",
            "opts": {
                "controllers": [
                    "c0"
                ],
                "externalInterfaces": [
                    "eth2"
                ],
                "hostname": "s2-Co-sw2",
                "netflow": "0",
                "nodeNum": 2,
                "sflow": "0",
                "switchIP": "",
                "switchType": "default"
            },
            "x": "658.0",
            "y": "346.0"
        }
    ],
    "version": "2"
}
```