

Sudan University of Science and Technology

College of Engineering

School of Electronics Engineering



Implementation of SDN in a Campus NAC Use Case

Research Submitted in Partial fulfillment for the Requirements of the Degree of
B.Sc. (Honors) in Electronics Engineering

Prepared by:

Mohammed Adil Abdelwahab Mohammed

Mohammed Omar Mohammed AL-Hassan Akoud

Mugahed Izzeldin Osman HajAhmed

Mustafa Khalid Mustafa Abdelrahim

Supervised by:

Dr. Ahmed Abdalla Mohamed Ali Abdalla

October 2016

قَالَ اللَّهُ تَعَالَى:

﴿وَلَقَدْ كَرَّمْنَا بَنِي آدَمَ وَحَمَلْنَاهُمْ فِي الْبَرِّ وَالْبَحْرِ وَرَزَقْنَاهُمْ مِّنَ الطَّيِّبَاتِ
وَفَضَّلْنَاهُمْ عَلَى كَثِيرٍ مِّمَّنْ خَلَقْنَا تَفْضِيلًا﴾

سورة الإسراء الآية رقم (70)

Dedication

To Our caring parents...

To our beloved ones...

To each of those who supported us financially and morally...

To whom we spent with the most beautiful moments (Our Colleagues)

...

We thank you all

Acknowledgement

First of all, we would like to thank GOD for blessing us and giving us the power to work and complete this thesis.

We would like to thank our supervisor **Dr. Ahmed Abdalla Mohamed Ali** for his advice and support during the writing of this thesis. His knowledge and dedication and opinion were useful in completing this research.

We would also like to thank everyone who supported us academically regardless of that support.

Most important of all, thanks to our families for their great support all the time.

Abstract

Software defined networks aim to provide high flexibility to modify network state and behavior, conventional networks are fixed and lack such features. This project introduces and demonstrates how to implement software defined networking concept within a campus network taking Sudan University of Science and Technology network as an example. The project uses Mininet emulation environment, and OpenDayLight as a controller to control this environment, the emulation components are integrated together to construct the system. The output of this project is an SDN based network controlled by the OpenDayLight controller. Which is using OpenFlow protocol to communicate between the controller and the switches.

المستخلص

الشبكات المعرفة بالبرمجيات تهدف الى إضافة قدر عالي من المرونة لتعديل حالة الشبكة وسلوكياتها في توجيه البيانات، حيث ان الشبكات التقليدية تفنقر لهذه الخواص. هذا المشروع يطرح ويبين كيفية تطبيق مفهوم الشبكات المعرفة بالبرمجيات على شبكات المجمعات، أخذاً شبكة جامعة السودان للعلوم والتكنولوجيا كمثال لذلك. يستخدم هذا المشروع بيئة {Mininet} لمحاكاة الأجهزة الحقيقية، ومتحكم {OpenDayLight} كنموذج للمتحكمات. أجزاء بيئة المحاكاة تم تجميعها من أجهزة مختلفة لتشكل منظومة المحاكاة. مخرجات هذا المشروع عبارته عن شبكة تعتمد على الشبكات المعرفة بالبرمجيات، ويتم التحكم بها باستخدام متحكم خاص. وتستخدم بروتوكول يقوم بعملية التخاطب بين المحولات والمتحكم.

Table of Contents

Dedication.....	iii
Acknowledgement.....	iv
Abstract	v
Abstract in Arabic	vi
List of Figures.....	x
List of abbreviations.....	xiii
1. INTRODUCTION	2
1.1 Preface:	3
1.2 The problem statement:	4
1.3 Aim and objectives:	4
1.4 Thesis outlines:.....	5
2. LITERATURE REVIEW	7
2.1 Introduction	8
2.2 What is software-defined networking?	8
2.2.1 SDN definition:.....	9
2.2.2 Control plane and data plane separation:	10
2.2.3 Why SDN? And the need for Data plane and control plane separation:..	12
2.3 Active networks:.....	14
2.4 Network operating system (NOS):.....	17
2.5 SDN functional architecture:	18

2.6 SDN controllers:	19
2.7 OpenFlow API:.....	22
2.7.1 Understanding open flow messages:	23
2.8 SDN in the campus environment:	26
2.8.1 Challenges of today's campus network:.....	27
2.8.2 SDN in the campus network:.....	29
2.8.3 The role of SDN and OpenFlow:.....	30
2.8.4 Campus network use cases:	31
2.9 SDN Migration:.....	34
2.9.1 SDN Migration Considerations:	34
2.9.2 Stanford university legacy-to-hybrid migration:.....	35
3. METHODOLOGY	39
3.1 Introduction:.....	40
3.2 Research activities:.....	41
3.2.1 Studying SDN concepts	42
3.2.2 Choosing the topology.....	42
3.2.3 Choosing the suitable environment	45
3.2.4 Choosing the emulator	45
3.2.5 Choosing the controller	45
3.3 Proposed system block diagram:	46
3.4 Environment tools and technology:	47

3.4.1 Oracle VM VirtualBox	47
3.4.2 Mininet network emulator.....	48
3.4.3 OpenDayLight controller.....	48
3.4.4 Wireshark.....	49
3.5 Configuration and Software.....	49
3.5.1 The controller	49
3.5.2 The server.....	59
3.5.3 The physical Host	62
4. RESULTS	68
4.1 Controller to switches Communication:	69
4.2 Topology Connectivity	71
4.3 HTTP connectivity.....	72
4.4 FTP Connectivity.....	74
5. CONCLUSION AND FUTURE WORK	77
5.1 Conclusion	78
5.2 Future work	79
REFERENCES.....	80

List of Figures

Figure 2-1: (a)Networking the old way (b)Networking the SDN way.....	13
Figure 2-2: SDN functional architecture illustrating the infrastructure, control, and application elements of which the network is comprised.	18
Figure 2-3: Examples of the most well-known SDN controllers	20
Figure 2-4: OpenFlow Messages.....	23
Figure 2-5: OpenFlow Negotiation	23
Figure 2-6: Message Types-Feature request.....	24
Figure 2-7: Message Types-Feature reply	24
Figure 2-8: Message Types-Set Config	25
Figure 2-9: Message Types-Multipart Request.....	26
Figure 2-10: Campus network architecture.....	28
Figure 2-11: LEFT: A typical university network today. RIGHT: An SDN-based architecture.....	32
Figure 2-12 Migration steps	35
Figure 3-1: Research Activities	41
Figure 3-2: SUST's Campus Network Topology	43
Figure 3-3: Simplified Topology in Mininet.....	44
Figure 3-4: System Block Diagram	46
Figure 3-5: Updating packages in Ubuntu	50
Figure 3-6: Installing JVM	51
Figure 3-7: downloading a clone of the OpenDayLight controller's source code.....	54
Figure 3-8: OpenDayLight Installation Command.....	55

Figure 3-9: Downloading Packages for Opendaylight's Controller Installation	56
Figure 3-10: Running the Controller	57
Figure 3-11: OSGi running	58
Figure 3-12: GUI of the controller	58
Figure 3-13: Roles on the windows 2008 server	59
Figure 3-14: Web server (IIS) HTTP Service	60
Figure 3-15: FTP Server Configuration	61
Figure 3-16: DNS Server Configuration	62
Figure 3-17: Host IP Address and preferred DNS server assignment	63
Figure 3-18: Command-Default route adding	63
Figure 3-19 Active routes before adding default route	64
Figure 3-20 Active networks after adding the default route	64
Figure 3-21: Browsing the FTP site on the Host Browser.....	65
Figure 3-22: Browsing the Default webpage using IP address	65
Figure 3-23 Browsing the Default webpage using the domain name	66
Figure 4-1: Displaying the Openflow packets in Wireshark.....	69
Figure 4-2: Content of Openflow message types	70
Figure 4-3: Different Areas of the Topology(DMZ, Internet, Internal Area).....	71
Figure 4-4: Successful ICMP Ping from Internal Area to DMZ.....	72
Figure 4-5: Blocked ICMP ping from internet host to internal Area.....	72
Figure 4-6: Testing Connectivity from Physical Host to HTTP Server	73
Figure 4-7: Testing Connectivity from Physical Host to HTTP Server via URL	73

Figure 4-8: HTTP Verification in Wireshark.....	74
Figure 4-9: NSLOOKUP verifying DNS Server.....	74
Figure 4-10: Host Accessing the FTP server	75
Figure 4-11: FTP Verification in Wireshark.....	75

List of abbreviations:

SDN	Software Defined Networking
WAN	Wide Area Network
NVF	Network Virtualization Function
NAT	Network Address Translation
DNS	Domain Name service
NAC	Network Access Control
LAN	Local Area Network
WLAN	Wireless Local Area Network
API	Application Program Interface
GPS	Global Positioning System
ONF	Open Network Foundation
RCP	Rich Client Platform
BGP	Border Gateway Protocol
IETF	Internet Engineering Task Force
ForCES	Forwarding and Control Element Separation
PCE	Path Computation Element
MPLS	Multiprotocol Label Switching
GMPLS	Generalized Multiprotocol Label Switching
RIP	Routing Internet Protocol
OSPF	Open Shortest Path Forwarding
NOS	Network Operating System
Cisco ASR	Cisco Aggregation Service Router
ACL	Access Control List

QoS	Quality of Service
AN	Active Network
CPU	Central Processing Unit
OS	Operating System
PCEP	Path Computation Element Protocol
JVM	Java Virtual Machine
TCP	Transmission Control Protocol
TLS	Total Logistic Services
BYOD	Bring Your Own Device
CLI	Command Line Interface
VLAN	Virtual Local Area Network
VRF	Virtual Routing and Forwarding
IP	Internet Protocol
AP	Access Point
TCO	Total Cost ownership
IEEE	Institute of Electrical and Electronics Engineers
WiMAX	Worldwide Interoperability for Microwave Access
ISP	Internet Service Provider
DMZ	Demilitarized Zone
FTP	File Transfer Protocol
DHCP	Dynamic Host Configuration Protocol
OVS	Open View Switch
ODL	OpenDayLight
USB	Universal Serial Bus

VM	Virtual Machine
GUI	Graphical User Interface
NFV	Network Function Virtualization
PPA	Personal Package Archives
POM	Project Object Model
MVN	Maven
OSGi	open Service Gateway Initiative
HTTP	Hypertext Transfer Protocol
IIS	Internet Information services
ICMP	Internet Control Message Protocol
URL	Uniform Resource Locator
CAPEX	Capital expenditure
OPEX	Operational expenditure

CHAPTER ONE

INTRODUCTION

Chapter One

1. INTRODUCTION

- 1.1 Preface
- 1.2 The problem statement
- 1.3 Aim and objectives
- 1.4 Thesis outlines

This chapter provides a brief overview of the literature review, problem definition, Methodology, aim and objectives, in addition to thesis outline.

1.1 Preface:

Software-defined networks (SDN): Is a modern architectural approach that optimizes and simplifies network operations by more closely binding the interaction (i.e., provisioning, messaging, and alarming) among applications and network services and devices, whether they are real or virtualized. Its common deployment model is by employing a point of logically centralized network control which then orchestrates, mediates, and facilitates communication between applications wishing to interact with network elements and wishing to convey information to those applications. The controller then exposes and abstracts network functions and operations via modern, application-friendly and bidirectional programmatic interfaces.

Among many benefits, SDN eliminates the rigidity present in traditional network and make it easier to build application for enterprise networks, data centers, internet exchange points, home networks and backbone/WAN. basically because it enables customizing the data plane to perform functions other than match-action like traffic shaping.

SDN changes the way of designing, configuring and managing networks. By decoupling the control plane from the data plane the chance of creating secure network is increased. And with a centralized controller the overall view and management of a network is becoming much easier. While SDN discusses the centralization of the controller, Network Virtualization Function (NVF) in contrast discusses the

centralization of Services. It offers a new way to design, deploy and manage networking services. NFV couples the network functions, such as network address translation (NAT), firewalling, intrusion detection, domain name service (DNS), and caching in a unified device that may be a real hardware or a virtualized device (controller). SDN & NVF work in parallel to make an open source environment that support innovation.

Campus NAC (Campus Network Access Control) is the ability to control access as well as service quality to LAN and WLAN for employees, contractors and visitors based on their roles and privileges in an organization. [1]

1.2 The problem statement:

To enable employees, contractors and visitors to access different services based on their privilege using Network Access Control in an SDN network.

1.3 Aim and objectives:

The aim of this project is to test and examine the behavior of different types of packets (control packets, Data packets, ... etc.) within SDN environment.

The objectives of this project are:

- To make a brief implementation of SDN on a campus network.
- To Implement NAC policies on the SDN network.
- To test and evaluate.

1.4 Thesis outlines:

This thesis approaches the aforementioned issues starting from the motivation of this thesis and a broader definition of technologies and introduction to the context, followed by a proposed system design, description of the implementation tools and measurements analysis. Hereby, the work has been structured in four main chapters as follows:

Chapter 2: A theoretical background of the proposed work is presented. Also this chapter presents some SDN-related concepts that is relevant to this thesis, and gives an overview of the related work and research performed on those aspects. It also explores some technologies and concepts that forms the road map of SDN.

Chapter 3: Describes the tools and technologies used in the implementation phase. Both network virtualization and SDN tools were used. The test cases scenarios are presented with explanation of the commands used and detailed overview of the controller architecture. In this chapter, all the steps taken to implement the network are included, from obtaining information to configuring the network to use SDN.

Chapter 4: Shows the results obtained from the use case network. This chapter introduces the expectations regarding the SDN benefits to the applied use case.

Chapter 5: Aims to draw the final remarks and conclusions of the presented work. Proposed optimizations and complementary future work are also presented.

CHAPTER TWO

LITERATURE REVIEW

2. LITERATURE REVIEW

- 2.1 What is software-defined networking?
- 2.2 Active networks
- 2.3 Network operating system (NOS)
- 2.4 SDN functional architecture
- 2.5 SDN controllers
- 2.6 OpenFlow API
- 2.7 SDN in the campus environment
- 2.8 SDN Migration

2.1 Introduction

The concept of Software-Defined Networking is not new and completely revolutionary; rather it arises as the result of contributions, ideas, and developments in research networking.

SDN changes the behavior of the network from configurability to programmability. Although most projects in this field focus on finding the implementation of basic services. Not a lot of work conducted the implementation of Network Access Control (NAC) in a SDN environment. However, our research will focus in this area.

In this chapter we discuss previous researches related to the context of this thesis. It covers architectural themes in networking where Software-defined networking originated, recent work on SDN, as well as network management and control, as well as the need for network security.

2.2 What is software-defined networking?

The term software-defined networking (SDN) has been coined in recent years. However, the concept behind SDN has been evolving since 1996, driven by the desire to provide user-controlled management of forwarding in network nodes. Implementations by research and industry groups include Ipsilon (proposed General Switch Management protocol, 1996), The Tempest (a framework for safe, resource-assured, programmable networks, 1998) and Internet Engineering Task Force (IETF) Forwarding and Control Element Separation, 2000, and Path Computation Element, 2004. Most recently, Ethane (2007) and

OpenFlow (2008) have brought the implementation of SDN closer to reality. Ethane is a security management architecture combining simple flow-based switches with a central controller managing admittance and routing of flows. OpenFlow enables entries in the Flow Table to be defined by a server external to the switch. SDN is not, however, limited to any one of these implementations, but is a general term for the platform. [2]

2.2.1 SDN definition:

SDN is described with the Open Networking Foundation (ONF)[3] as “Software-Defined Networking (SDN) is an emerging network architecture where network control is decoupled from forwarding and is directly programmable”.

Per this definition, SDN is defined by two characteristics, namely decoupling of control and data planes, and programmability on the control plane. Nevertheless, neither of these two signatures of SDN is totally new in network architecture, as detailed in the following:

- First, several previous efforts have been made to promote network programmability. One example is the concept of active networking that attempts to control a network in a real-time manner using software. Similarly, software routing suites on conventional PC hardware, such as Click [4], XORP [5], Quagga [6], and BIRD [7], also attempt to create extensible software routers by making network devices programmable. Behavior of these network devices can be modified by loading different or modifying existing routing software.
- Second, the spirit of decoupling between control and data planes has been proliferated during the last decade. Caesar et al. first

presented a Routing Control Platform (RCP) in 2004[8], in which Border Gateway Protocol (BGP) inter-domain routing is replaced by centralized routing control to reduce complexity of fully distributed path computation. In the same year, IETF released the Forwarding and Control Element Separation (ForCES) framework, which separates control and packet forwarding elements in a ForCES Network [9] – [10]. In 2005, Greenberg et al. proposed a 4D approach [11] – [12], introducing a clean slate design of the entire network architecture with four planes. These planes are “decision”, “dissemination”, “discovery”, and “data”, respectively, which are organized from top to bottom. In 2006, the Path Computation Element (PCE) architecture was presented to compute label switched paths separately from actual packet forwarding in MPLS and GMPLS networks [13]. In 2007, Casado et al. presented Ethane, where simple flow-based Ethernet switches are supplemented with a centralized controller to manage admittance and routing of flows [14] – [15]. In this latest development, the principle of data-control plane separation has been explicitly stated. Commercial networking devices have also adopted the idea of data-control plane separation. For example, in the Cisco ASR 1000 series routers and Nexus 7000 series switches, the control plane is decoupled from the data plane and modularized, allowing coexistence of an active control plane instance and a standby one for high fault tolerance and transparent software upgrade.

2.2.2 Control plane and data plane separation:

SDN focuses on four key features:

- Separation of the control plane from the data plane.

- A centralized controller and view of the network.
- Open interfaces between the devices in the control plane (controllers) and those in the data plane.
- Programmability of the network by external applications.

Before diving on the concept of separating control plane from data plane we should first understand the difference between them.

What is control plane?

- Makes decisions about where traffic is sent.
- Control plane packets are *destined to* (like telnet) or locally *originated by* the router itself.
- The control plane functions include the system configuration, management, and exchange of routing table information.
- The route controller exchanges the topology information with other routers and constructs a routing table based on a routing protocol, for example, RIP, OSPF or BGP.
- Control plane packets are processed by the router to update the routing table information.
- It is the *Signaling* of the network.
- Since the control functions are not performed on each arriving individual packet, they do not have a strict speed constraint and are less time-critical.

What is Data plane?

- Also known as Forwarding Plane.
- Forwards traffic to the next hop along the path to the selected destination network according to control plane logic.

- Data plane packets go *through* the router.
- The routers/switches use what the control plane built to dispose of incoming and outgoing frames and packets.

2.2.3 Why SDN? And the need for Data plane and control plane separation:

The fundamental purpose of the communication network is to transfer information from one point to another. Within the network the data travels across multiple nodes, and efficient and effective data transfer (forwarding) is supported by the control provided by network applications/ services.

2.2.3.1 NETWORKING THE OLD WAY:

In traditional networks, as shown in 10Figure 2-1, the control and data planes are combined in a network node.

The control plane is responsible for configuration of the node and programming the paths to be used for data flows. Once these paths have been determined, they are pushed down to the data plane. Data forwarding at the hardware level is based on this control information.

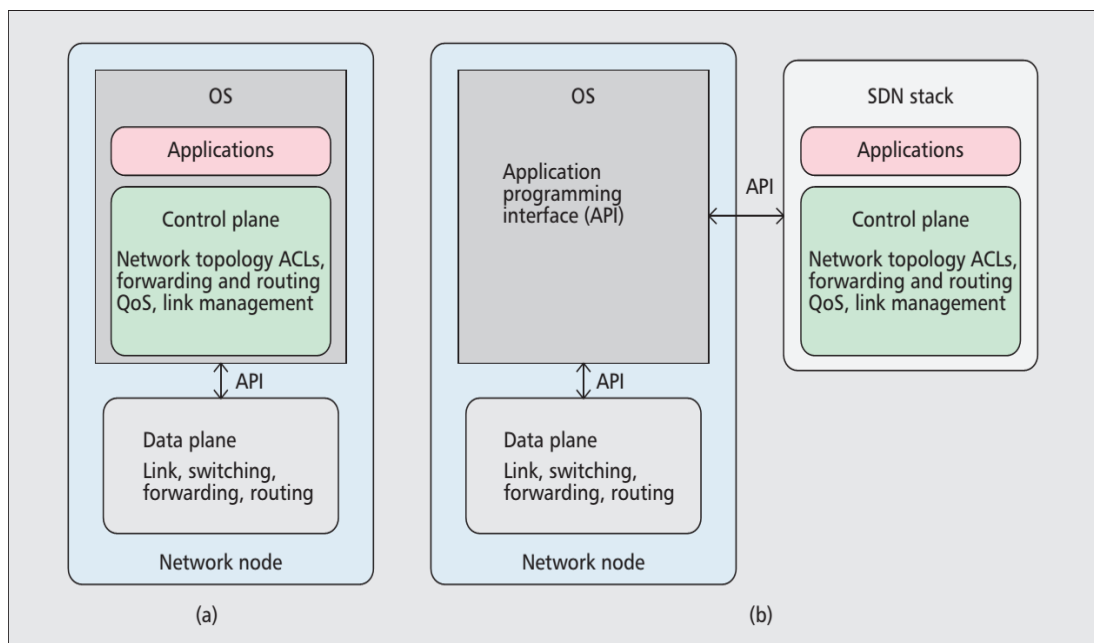
In this traditional approach, once the flow management (forwarding policy) has been defined, the only way to make an adjustment to the policy is via changes to the configuration of the devices.

This has proven restrictive for network operators who are keen to scale their networks in response to changing traffic demands, increasing use of mobile devices, and the impact of “big data.”

2.2.3.2 NETWORKING THE SDN WAY:

From these service-focused requirements, SDN has emerged. Control is moved out of the individual network nodes and into the separate, centralized controller. SDN switches are controlled by a network operating system (NOS) that collects information using the API shown in 10Figure 2-1 and manipulates their forwarding plane, providing an abstract model of the network topology to the SDN controller hosting the applications.

The controller can therefore exploit complete knowledge of the network to optimize flow management and support service-user requirements of scalability and flexibility. For example, bandwidth can be dynamically allocated into the data plane from the application.



10Figure 2-1: (a)Networking the old way (b)Networking the SDN way

SDN holds great promise in terms of simplifying network deployment and operation along with lowering the total cost of managing enterprise and carrier networks by providing programmable network services. However, a number of challenges remain to be addressed. [2]

2.3 Active networks:

The basic goals of active networking (AN) are to create networking technologies that, in contrast to current networks, are easy to evolve and which allow application specific customization. To achieve these goals, AN uses a simple idea, that the network would be easier to change and customize if it were programmable. [16]

Conventional networks are not "programmable" in any meaningful sense of the word. The concepts of software data plane and programmable nodes were introduced with active networks where the nodes of this "active" network are programmed to perform custom operations on the messages that pass through the node. For example, a node could be programmed or customized to handle packets on an individual user basis or to handle multicast packets differently than other packets. "Smart packets" use a special self-describing language that allows new kinds of information to be carried within a packet and operated on by a node.

Active networks have been implemented as overlay networks due to their architecture; it is composed of execution environments, a node operating system capable of supporting one or more execution environments. It also consists of active hardware, capable of routing or switching as well as executing code within active packets. This differs

from the traditional network architecture which seeks robustness and stability by attempting to remove complexity and the ability to change its fundamental operation from underlying network components. Network processors are one means of implementing active networking concepts.

Although Active Networks have the high-level goals of improving evolve-ability and customizability, there are a number of low-level concerns that must be balanced to achieve these high-level goals.

The first of these concerns is flexibility .AN systems aim to significantly improve the flexibility with which we can build networks. The second concern is safety and security. It is crucial that while adding flexibility, we not compromise the safety or security of the resulting system. The third concern is performance. If adding flexibility results in a system that cannot achieve its performance goals, it will be pointless. The final concern is usability. It is important that the resulting system not be so complex as to be unusable. The other main perspective we wish to develop is how the combination of disciplines discussed above come together to help address these concerns.

Active networks and SDN have the same motivation which is:

- Accelerating innovation: traditional networks have the drawback that intermediate nodes (e.g., routers, switches) are vertically integrated closed systems whose functions are rigidly built into the embedded software and hardware by intermediate node vendors therefore the development and deployment of new services in such networks requires a long standardization process. Moreover, the

range of these services is limited because they can't anticipate and provide support for all future applications.

- The decision to provide a well-designed and architecturally open platform –efforts had already been going on to achieve this–.
- To gain the benefits of being able to put application and network knowledge in the same place. This can be extremely beneficial to the network. Stock quotes and online auctions are typical examples. In online auctions the server collects and processes client bids for available items. This server also responds to clients for request for current price of a specific item. Because of the network delay experienced by a packet responding to such a query, its information may be out of date by the time it reaches the client. In active networks, when a server 'senses' that is it heavily loaded, it can activate filter in nearby nodes to drop bids lower than the latest bid and periodically update them with current price of the item by sending active packets.
- To provide an integrating mechanism for security, authentication and monitoring. Thus eliminating the need for multiple security/authentication systems that operate independently at each communication layer protocol.

Saying that, it is clear that active networks offered some intellectual contributions that relates to SDN, the three main contributions are:

- Programmable functions in the network that lowers the barrier to innovation.

- Network virtualization and the ability to de-multiplex software programs based on packet headers.
- The vision of a unified architecture for middle-box orchestration.

2.4 Network operating system (NOS):

Modern network devices are complex entities composed of both hardware and software. Thus, designing an efficient hardware platform is not, by itself, sufficient to achieve an effective, cost-efficient and operationally tenable product. The control plane plays a critical role in the development of features and in ensuring device usability. Thus the need for networking operating system arises. Developing a flexible, long-lasting and high-quality network OS provides a foundation that can gracefully evolve to support new needs in its height for up and down scaling, width

for adoption across many platforms, and depth for rich integration of new features and functions.

Network Operating Systems extend the facilities and services provided by computer operating systems to support a set of computers, connected by a network. The environment managed by a network operating system consists of an interconnected group of machines that are loosely connected. By loosely connected, we mean that such computers possess no hardware connections at the CPU – memory bus level, but are connected by external interfaces that run under the control of software. Each computer in this group run an autonomous operating system, yet cooperate with each other to allow a variety of facilities

including file sharing, data sharing, peripheral sharing, remote execution and cooperative computation. Network operating systems are autonomous operating systems that support such cooperation.

Network operating systems (specifically those in routers) have been through several generations starting from Monolithic architecture going through the control plane modularity phase and as new challenges started to appear on the surface in the last decade the need for enhanced flexibility, scalability and continuous operation led to the creation of third generation network operating system. [17]

2.5 SDN functional architecture:

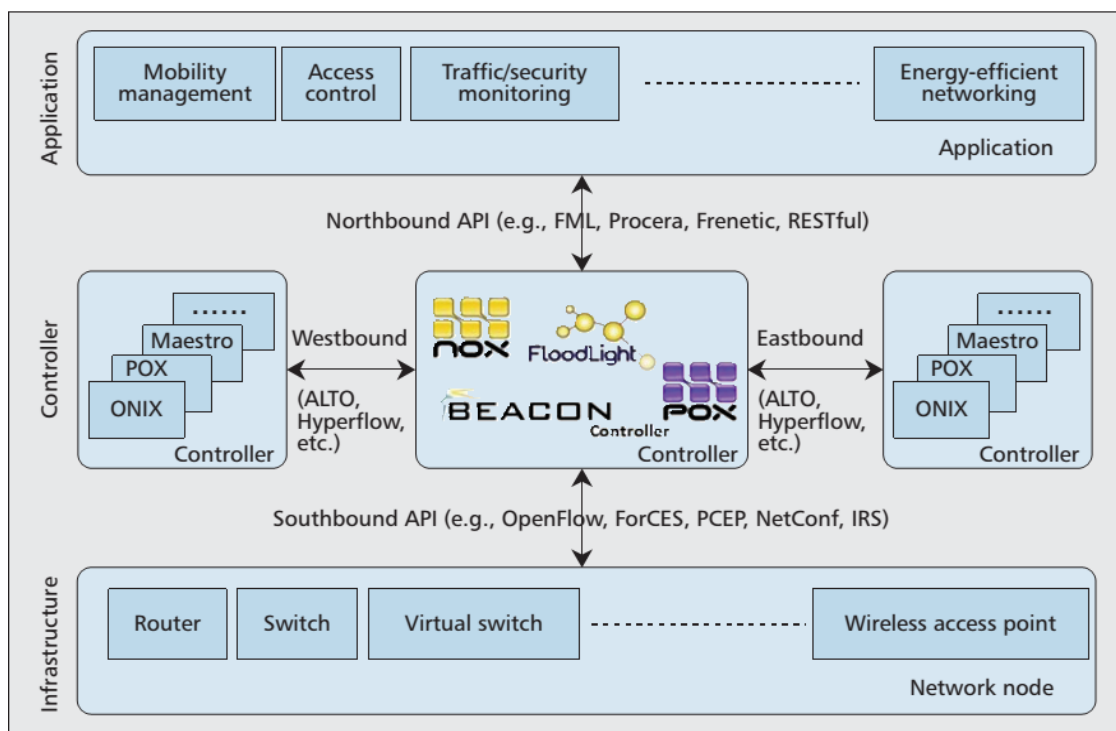


Figure 2-2: SDN functional architecture illustrating the infrastructure, control, and application elements of which the network is comprised.

The future SDN architecture is described in Figure 2-2. This architecture encompasses the complete network platform. The bottom tier of Figure 2-2 involves the physical network equipment including Ethernet switches and routers. This forms the data plane. The central tier

consists of the controllers that facilitate setting up and tearing down flows and paths in the network. The controllers use information about capacity and demand obtained from the networking equipment through which the traffic flows. The central tier links with the bottom tier via an application programming interface (API) referred to as the southbound API. Connections between controllers operate with east and westbound APIs. The controller application interface is referred to as the northbound API. Functional applications such as energy-efficient networking, security monitoring, and access control for operation and management of the network are represented at the top of Figure 2-2 highlighting the user control/management separation from the data plane. The transition from the traditional network to the state of the art in SDN today is presented. [2]

2.6 SDN controllers:

In this section an overview of some basic SDN controllers is given. SDN controllers differ in their basic architecture, programming model, and other concepts. Every controller has its own ideal situation and the choice of the controller is affected by many considerations like the choice of programming language which can sometimes affect performance, the learning curve of the controller can get steep sometimes so this should be taken into consideration as well. The user base and community support is also a factor. Finally, focus should be considered before making this choice in terms of Southbound and Northbound interfaces (policy layer) they support and whether it is used for education, research or production environments.

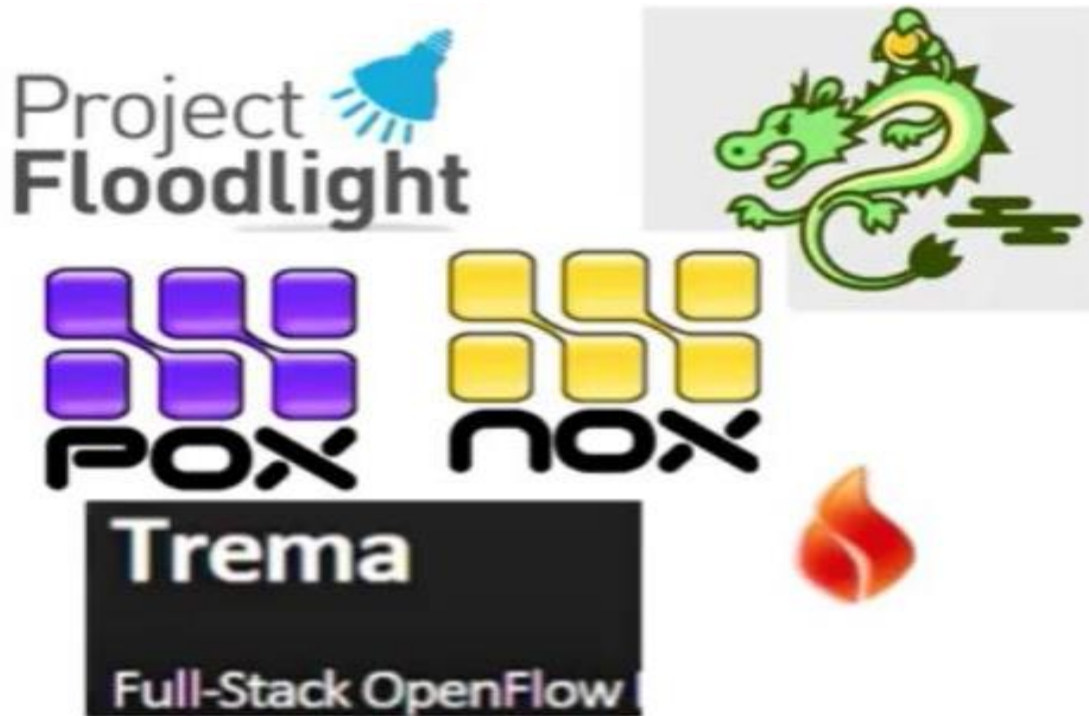


Figure 2-3: Examples of the most well-known SDN controllers

Some examples of the most well-known SDN controllers are NOX/POX, Ryu, OpenDayLight, and Floodlight.

- *NOX*: is a first-generation OpenFlow controller, it's one of the most widely used controllers because it is stable, open source and it supports OpenFlow. It is available in two versions either NOX-Classis which is implemented in C++ or Python but is no longer supported. Or either, a newer supported version of NOX which is implemented in C++ only but with a cleaner code base and better performance.

- *POX*: is essentially the same as NOX but it's implemented in Python only and it's relatively easier to deal with it in terms of writing and reading codes. it is a good choice if the performance is not an issue.

- *Ryu*: is an open-source Python controller that supports OpenFlow and Openstack. it had relatively poor performance with respect to other commercial grade controllers.

- *Floodlight* is an open-source Java controller that supports OpenFlow and is maintained by Big Switch Networks. It's considered the optimum choice for multi-tenant's cloud environment because it supports OpenStack and this type of implementation, it is also suitable when production level performance is needed and it is well documented, however, it has a steep learning curve.

- *OpenDayLight*: The OpenDayLight Project is a recent open-source project founded by some of the big vendors such as: Big Switch Networks, Brocade, Cisco, Citrix, Ericsson, HP, IBM, Juniper Networks, Microsoft, NEC, Red Hat and VMware. OpenDayLight is developed as a modular, pluggable, and flexible controller platform. This controller is completely programmed and it is integrated within its own Java Virtual Machine (JVM). Hereby, it can be deployed on any hardware and operating system platform that has Java environment installed. Chapter 3 Background 22 Table 2-1 shows a summary of some properties of some controllers.

Table 2-1: Summary of some properties of some controllers

	NOX	POX	Ryu	Floodlight	ODL
Language	C++	Python	Python	Java	Java
Performance	Fast	Slow	Slow	Fast	Fast
Distributed	No	No	Yes	Yes	Yes
OpenFlow	1.0 (CPqD: 1.1, 1.2, 1.3)	1.0	1.0, 1.1, 1.3, 1.4	1.0	1.0, 1.3
Learning Curve	Moderate	Easy	Moderate	Steep	Steep

2.7 OpenFlow API:

Before the emergence of OpenFlow, the ideas underlying SDN faced a tension between the vision of fully programmable networks and pragmatism that would enable real-world deployment. OpenFlow struck a balance between these two goals by enabling more functions than earlier route controllers and building on existing switch hardware through the increasing use of merchant-silicon chipsets in commodity switches. Although relying on existing switch hardware did somewhat limit flexibility, OpenFlow was almost immediately deployable, allowing the SDN movement to be both pragmatic and bold. The creation of the OpenFlow API⁵¹ was followed quickly by the design of controller platforms such as NOX³⁷ that enabled the creation of many new control applications.

2.7.1 Understanding open flow messages:

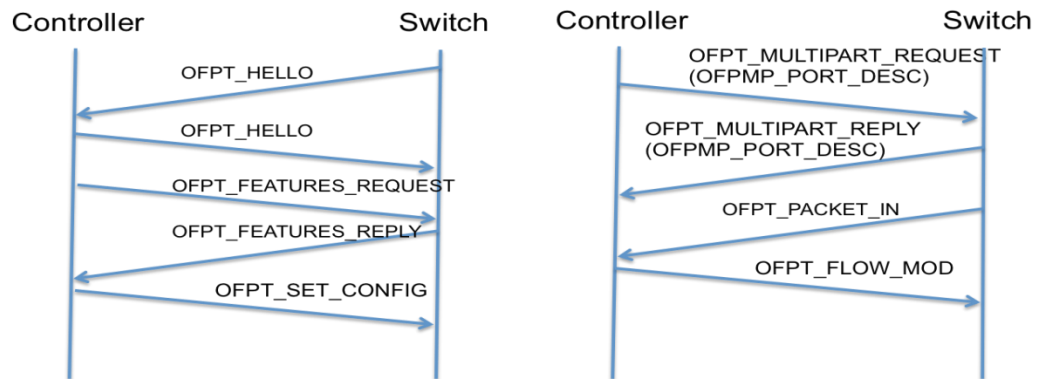


Figure 2-4: OpenFlow Messages

The switch initiates a standard TCP (or TLS) connection to the controller. When an OpenFlow connection is established, each entity must send an OFPT_HELLO message with the protocol version set to the highest OpenFlow protocol version supported by the sender. In Figure 2-5, we can see that OpenFlow version 1.3 has been negotiated.

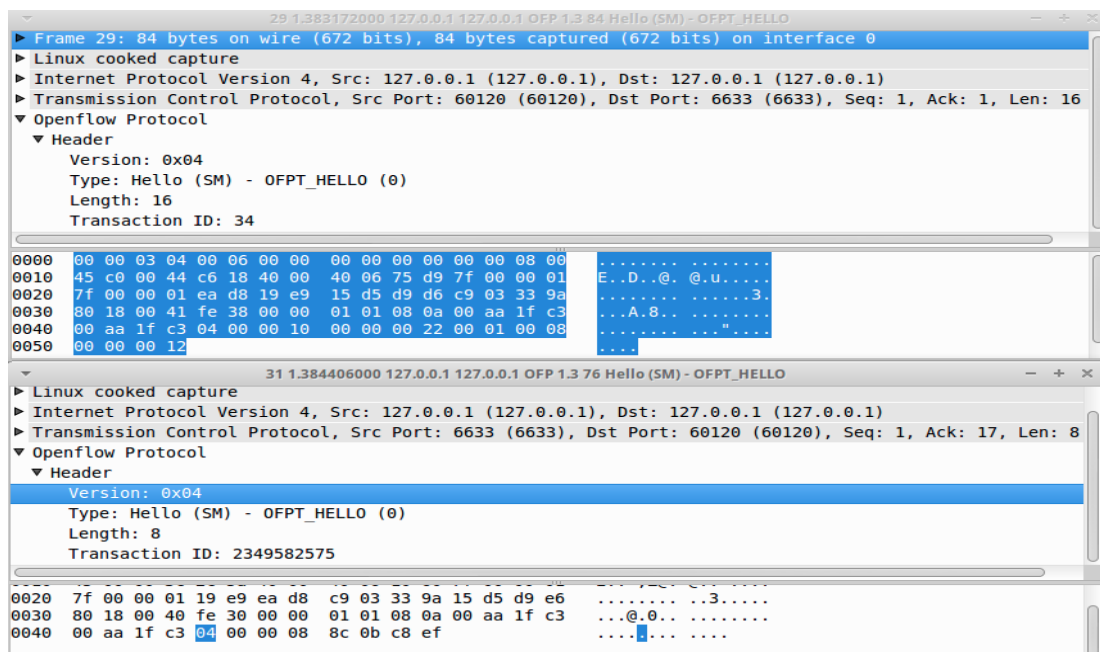


Figure 2-5: OpenFlow Negotiation

Feature Request– Reply

After successfully establishing a session, the controller sends an OFPT_FEATURES_REQUEST message. This message only contains an OpenFlow header and does not contain a body.

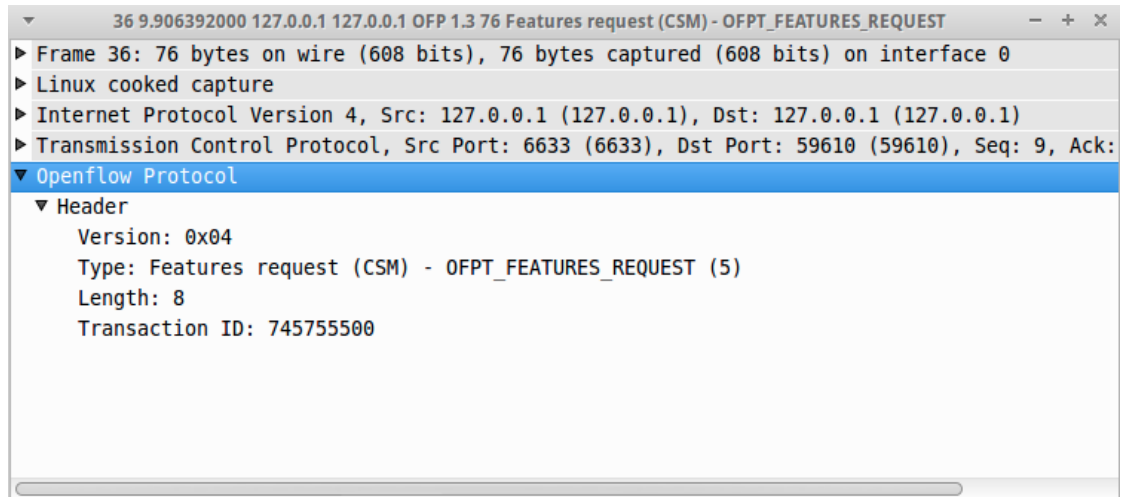


Figure 2-6: Message Types-Feature request

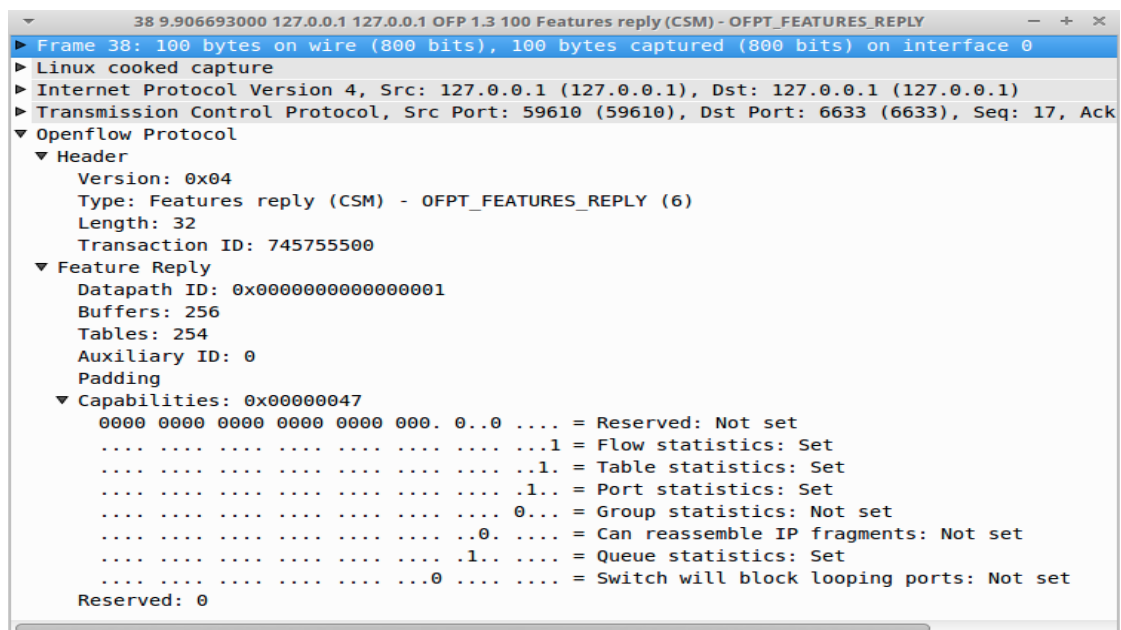


Figure 2-7: Message Types-Feature reply

The switch responds with an OFPT_FEATURES_REPLY message. Notice the Data path ID and the switch capabilities sent as part of the Feature reply message.

Set Configuration

Next, the controller sends the OFPT_SET_CONFIG message to the switch. This includes the set of flags and Max bytes of packet that the data path should send to the controller.

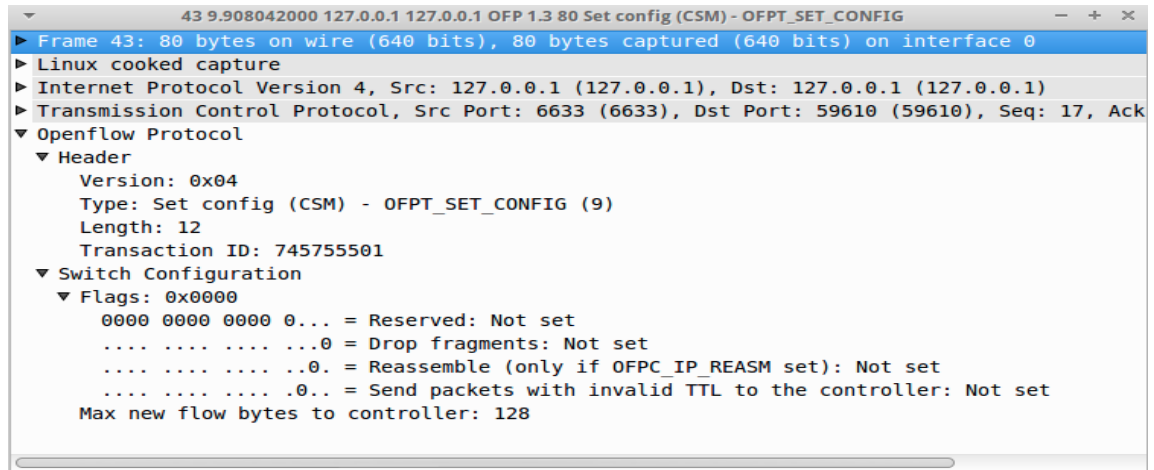


Figure 2-8: Message Types-Set Config

Multipart Request – Reply

The controller may request state from the data path using the OFPT_MULTIPART_REQUEST message. The message types handled by this message include various statistics (FLOW/ TABLE/ PORT/ QUEUE/METER etc.) or description features (METER_CONFIG/ TABLE_FEATURES/ PORT_DESC etc.). In our simple_switch_13.py, RYU internally sends a MULTIPART_REQUEST to request port description.


```

45 9.947447000 127.0.0.1 127.0.0.1 OFP 1.3 84 Multipart request (CSM) - OFPT_MULTIPART_
▶ Frame 45: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on int
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (1
▶ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 59610 (596
▼ Openflow Protocol
  ▼ Header
    Version: 0x04
    Type: Multipart request (CSM) - OFPT_MULTIPART_REQUEST (18)
    Length: 16
    Transaction ID: 745755502
  ▼ Multipart request
    Type: Port description - OFPMP_PORT_DESC (13)
  ▼ Flags: 0x0000
    .... .... .... ...0 = More requests to follow: Not set
  Padding
  Body: <MISSING>

```

Figure 2-9: Message Types-Multipart Request

The switch replies with the PORT_DESCRIPTION of all active ports in the switch. Note: in OF 1.0, the port descriptions were returned as part of the FEATURE_REPLY message. Now this is handled separately as MULTIPART_* in OF 1.3. [18]

2.8 SDN in the campus environment:

Today's campus networks are facing major challenges. Mobile clients, BYOD, video, and the ever-growing number of connected devices and applications are rapidly changing the network landscape, no matter whether the campus is corporate or educational. These dramatic changes tax the ability of current solutions to deliver agility, performance, and seamless user experience. One of the primary reasons for these challenges is that network technology evolution is simply not keeping pace with evolving demands. Software Defined Networking

(SDN) can alleviate these challenges, offering flexibility and the ability to develop new capabilities quickly and cost-effectively.

Over the past few years, IT organizations at enterprises and educational institutions have come under increasing pressure from end users to provide access to applications and data from anywhere and at any time. As mobile devices such as smartphones and tablets proliferate in campus environments, users increasingly access and store sensitive data on these devices—which are often owned by the user, not the organization. Not only must campus networks be secure, scalable, and manageable, they must also maintain isolation among an ever-increasing diversity of users, applications, services, devices, and access technologies. Consequently, networks that serve campuses must evolve to address these changing requirements. [19]

2.8.1 Challenges of today's campus network:

Typical campus network architectures are structured into three layers—core, aggregation/distribution, and access—that connect diverse endpoints, as shown in Figure 2-10. Typically, Layer 2 is used for the access layer, and Layer 3 is used for the core layer. This not only increases management costs and complexity (because wired and wireless networks are separate), it also precludes a seamless user experience (because the two networks provide different capabilities and feature sets).

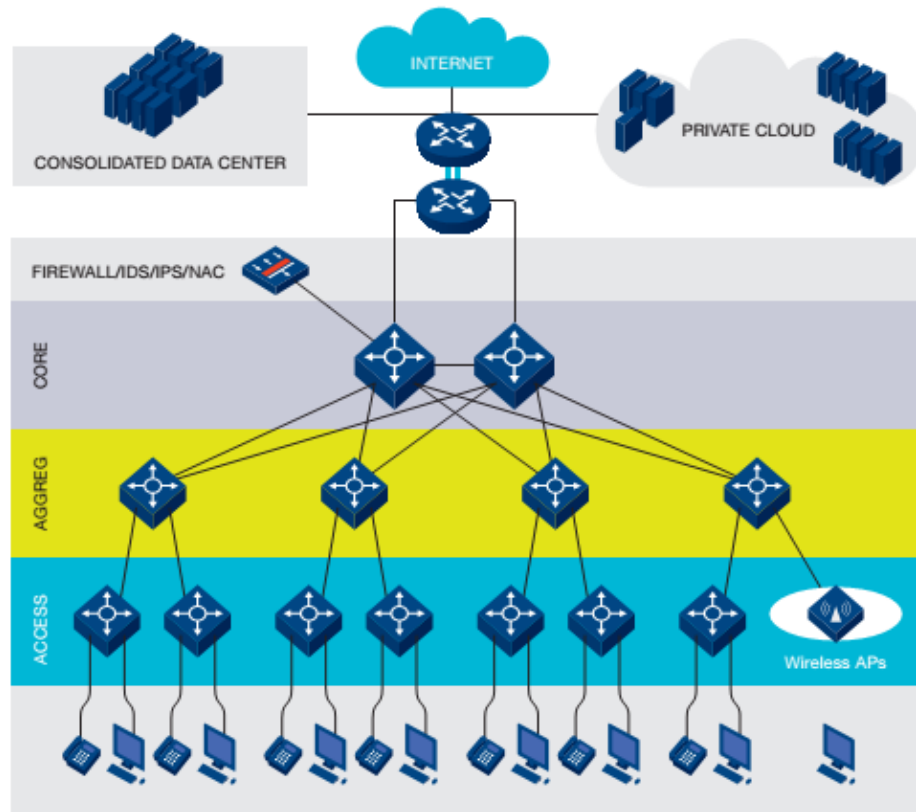


Figure 2-10: Campus network architecture

Because campus networks are by nature heterogeneous, they are often difficult to manage, leading to excess costs along with scalability and reliability problems. Network configuration changes are subject to lengthy provisioning times and configuration errors because network devices must be configured individually, typically through the CLI or proprietary element management systems.

Organizations are presently addressing these challenges in a fragmented fashion with wireless LAN controllers and Wi-Fi access points, VLANs for Layer 2 isolation, and Virtual Routing and Forwarding (VRF) for Layer 3 traffic isolation. These strategies might be efficient for specific circumstances, but often at the cost of scalability and flexibility.

An OpenFlow-enabled Software Defined Network offers a much simpler approach to traffic isolation and unified management. By separating the control, management, and service layers from the data-plane layer, OpenFlow eliminates the limitations and operational overhead of VLANs and VRFs. [19]

2.8.2 SDN in the campus network:

An OpenFlow based SDN network architecture simplifies the campus network while offering significantly greater flexibility.

- Rapid service deployment and tear down without impacting other logical networks, thanks to network virtualization.
- Improved service availability because alternate paths can be pre-computed, which also improves responsiveness compared with traditional network convergence upon topology changes.
- Traffic isolation of logical networks at both Layer 2 and Layer 3.
- Optimal resource utilization, because management, services, and applications are virtualized to maximize utilization while minimizing space and power consumption.

OpenFlow-based SDN introduces the multi-layer flow paradigm, which provides a higher level of control. By virtualizing the campus network in slices, granular policies can be applied to individual and/or groups of flows at the centralized controller, decoupling policy from hardware. For instance, access policies can be enforced for different departments, different types of access (wireless vs. wired), or even remote versus local users. Such policies are much simpler to enforce, especially for the increasingly mobile workforce. [19]

2.8.3 The role of SDN and OpenFlow:

OpenFlow-based SDN can overcome the limitations with existing campus networks. Typically, a logical network is created by associating a physical port of a switch or VLAN to a specific logical network ID, with its own routing protocol instance and forwarding table. Whenever a packet needs to be forwarded, it will be associated with the logical network based on the port it arrived on or the VLAN ID. This approach has several limitations. A port or a VLAN can belong to only one logical network and therefore cannot support multiple flows that terminate on different logical networks. In addition, these methods are proprietary and cannot interoperate with each other.

With SDN, the controller can determine the logical network for every flow, then tunnel the traffic to the end of the logical network. It becomes easier to define logical networks as needed, avoiding the need to create a routing protocol instance in every router for each logical network. This approach is scalable and much more flexible than VLAN/VRF approaches. In addition, SDN-based logical networks can easily be created, updated, and terminated based on dynamic requirements. By programming the traffic forwarding rules across the data forwarding devices, it becomes easy to reorder service execution and implement service chaining.

Another advantage is that with SDN, one can easily deploy policies across logical networks to enable traffic across these networks.

[19]

2.8.4 Campus network use cases:

2.8.4.1 Traffic isolation:

In this example, the campus is that of an educational institution. A typical university network serves diverse tenants, including faculty, students, medical facilities, libraries, a police department, restaurants, and bookstores. These individual tenants may need private addressing schemes that may overlap. This requires the university network to isolate traffic among multiple tenants and operate logical networks over a single physical network.

SDN enables policies to be enforced per application, which in turn allows access only to specific network resources (or a specific share of network resources). Figure 2-11 depicts a typical university network where a single physical network is shared by many diverse entities in a single location.

Campus networks require logically partitioned networks, each with its own policy. Currently, solutions such as MPLS or VRF-Lite are used to create logical network slices over a single physical network. Deploying and managing these technologies is static, time-consuming, and very cumbersome. SDN/OpenFlow-enabled switches enable these logical networks to be created on demand in a matter of minutes instead of weeks. These switches can also enforce flexible policies to control and limit interaction among the logical networks. [20]

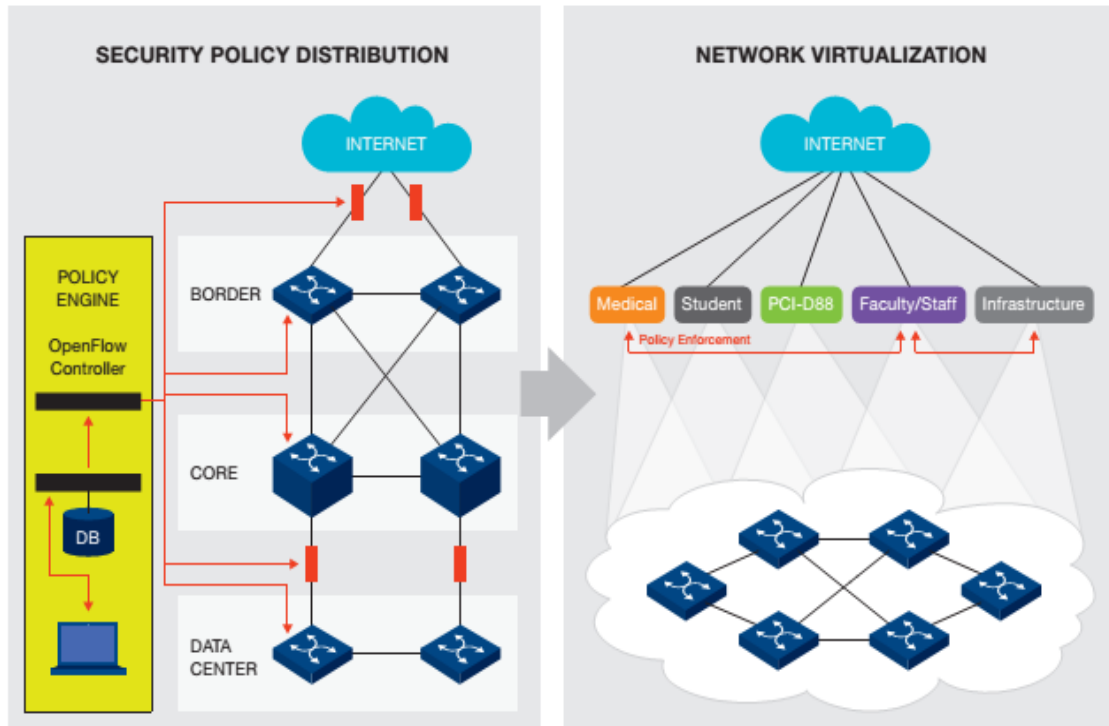


Figure 2-11: LEFT: A typical university network today. RIGHT: An SDN-based architecture.

2.8.4.2 Security and policy enforcement:

Most of the security policies today are limited to a VLAN or an interface. These are statically configured without considering the context of an application. Though there are enhancements using 802.1x dynamic policies and identity management systems, these still do not offer flexible security policy management and enforcement. In a SDN environment, the SDN controller understands the context of a flow (user, device, location, time, application, and potentially other external factors) and enables network admins to configure fine granular policies and enforce these at the access or at any intermediate switch as needed. [20]

2.8.4.3 BYOD and seamless mobility:

Employees are bringing their own devices and applications, and are expecting seamless connectivity and mobility. Though there are several solutions to achieve this, many of these are developed by vendors, and provide limited flexibility. With ever-growing number of types of devices, operating systems and software patches, vendors cannot catch up quickly enough to meet customers' needs. Organizations need the ability to program their own policies without waiting for vendors to release software upgrades. SDN allows organizations to develop their own enhancements to BYOD applications to meet their particular requirements.

Today, mobility is achieved either using tunneling among WLAN controllers or mobile IP technology. While controller-based mobility works, this mode will change as customers are demanding unified wired and wireless solutions. The data plane will be local to an AP. The mobile IP mechanism works but cannot scale and has other limitations. With SDN, a centralized controller knows end-user devices and application flows. It would be able to program the network to forward the traffic appropriately without hair-pinning the traffic on an anchor AP. [20]

2.8.4.4 Video streaming and collaboration applications:

Video and collaboration applications have become critical for the success of an organization. Most of these applications typically are more efficient when they use multicast technology. IP multicast technology is mature and available, but it is still difficult to deploy and troubleshoot. It is not as widely deployed as IP unicast technology, and it has forced many organizations to deploy video applications using other mechanisms

based on IP unicast forwarding. SDN is an ideal solution for these types of applications, as the SDN controller knows the topology, sources and listeners, and can build an efficient multicast topology and program the network on an on-demand basis. [20]

2.9 SDN Migration:

Open Software-Defined Networking (SDN) has been well accepted by the networking industry as the way to transform enterprise, data center, service provider, carriers and campus networks. The objective of this SDN transformation is to enable differentiated new services faster than ever before, simplify the network, and lower the total cost of ownership (TCO). The key attributes for a network that has been migrated to SDN are programmability, openness, heterogeneity, and maintainability. SDN will also facilitate the re-architecture required to address the increasing demand on the network due to dynamic connectivity. [21]

2.9.1 SDN Migration Considerations:

In SDN migration a number of challenges must be confronted, including cost, performance, service availability, management, and security. Addressing security vulnerabilities is among the highest priorities for network operators.

The key steps involved in an SDN migration are:

- **Identify and prioritize core requirements of the target network.** Not all requirements of the traditional starting network may be met, at least initially, by the target software-defined network.
- **Prepare the starting network for migration.** The starting network might need to be moved to a clean intermediate standard state from which the rest of the migration can proceed.
- **Implement a phased network migration.** Migrating individual devices will necessitate device-specific drivers and methods.
- **Validate the results.** Once migration is completed, the target network must be validated against a documented set of requirements or expectations. [21]

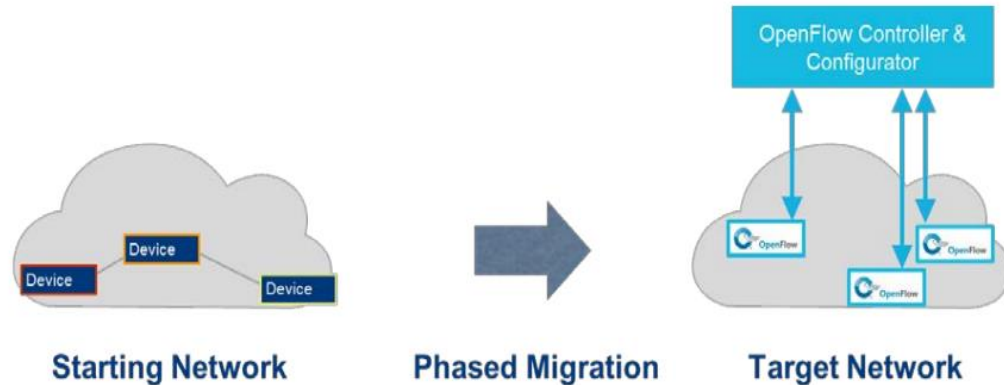


Figure 2-12 Migration steps

2.9.2 Stanford university legacy-to-hybrid migration:

One of Stanford University's primary motivations for SDN migration was to gain better insight into and verification of OpenFlow as a viable technology. Stanford deployed a fully functional SDN network using OpenFlow controllers over part of its campus. This migration was

focused initially on wireless users, followed by select wired users spanning two independent buildings. This migration encompassed several IEEE 802.1q VLANs, which were interconnected—as is commonly done—with a Layer 3 router.

In one of the buildings, Stanford deployed six 48-port 1GE OpenFlow-enabled switches from various vendors. The second building deployed one 48-port OpenFlow-enabled switch. One Layer 2 domain was used at each building to support about 34 Wi-Fi access points. These access points supported 802.11g interfaces running Linux-based switching software and one WiMAX base station in one of the buildings.

The target campus deployment specified network availability to exceed 99.9% with a fail-safe scheme to revert back to the legacy network in case of significant outages. In addition, network performance was specified to be comparable to the legacy network benchmarks without any impact to user experience.

The migration was accomplished in five phases. The first phase exposed traffic visibility to facilitate network configuration changes for selected users. In phase 2, VLANs and users were migrated to OpenFlow-based SDN through the following steps:

- 1- Deployed OpenFlow support on relevant hardware via a software update.
- 2- Verified OpenFlow support on switches for the configured VLANs and to test endpoint reachability.
- 3- Migrated users to the OpenFlow-enabled network.

4- Enabled OpenFlow for the new subnet by configuring the OpenFlow controller.

5- Validated migration objectives for reachability, performance, and stability using network management tools. [21]

CHAPTER THREE

METHODOLOGY

3. METHODOLOGY

- 3.1 Introduction
- 3.2 Research activities
- 3.3 Proposed system block diagram
- 3.4 Environment tools and technology
- 3.5 Configuration and Software

3.1 Introduction:

This chapter demonstrates how to implement basic SDN in a campus network using network emulator and OpenDayLight controller. First, Mininet emulator -which works based on Linux operating system, will be used. The data plane and control plane will be separated, All the devices involved in the data plane have basic similar features in their design, there is a specialized hardware for the packet processing, and over the hardware resides an operating system that receives information from the hardware and runs a software application. The software usually contains thousands lines of complex code to determine the next hop for the packet arriving to reach its destination. These codes are usually follow a certain standard protocol or determined by the vendor, the functions of the control plane will be carried out by the central controller (Open daylight). The controller will be programmed using python programming language.

To give a better understanding of how this project works this chapter is divided into two sections, the first section gives an abstraction of what was done throughout the project; the second section gives the proposed topology and the environment in which the project is implemented.

3.2 Research activities:

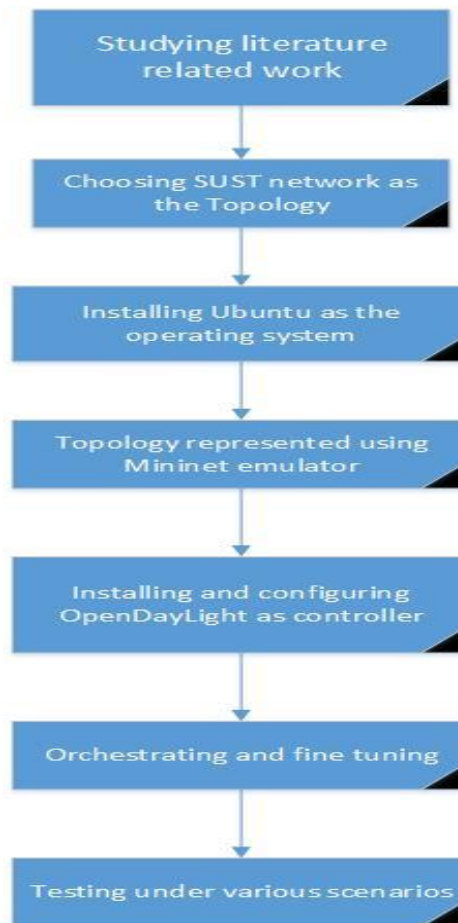


Figure 3-1: Research Activities

Figure 3-1 describes a general overview of how this project was assembled, in order to bring out the final implementation of the SDN topology within a campus network.

3.2.1 Studying SDN concepts

This part includes different activities and wide research, by attending different workshops, studying Nick Feamester course, and consulting many experts in the SDN area, enough information has been gathered in order to start the implementation.

The rehearsed researches revealed that most of the implementation of the SDN technology was applied on data centers and service providers rather than campus or enterprise networks.

Based on this literature review both emulator, controller and other necessary tools has been selected, these tools will be discussed later.

3.2.2 Choosing the topology

First of all, multiple networks were thought-out to be the network implemented in this project, eventually SUST network was selected to be simplified and tested. Secondly, a clearance was needed in order to get information about Sudan University of Science and Technology (SUST) network topology, once this clearance was acquired, several visits has been scheduled to the university's main network department and data center, the result of it was acquiring the main SUST's campus network topology. As shown in Figure 3-2:

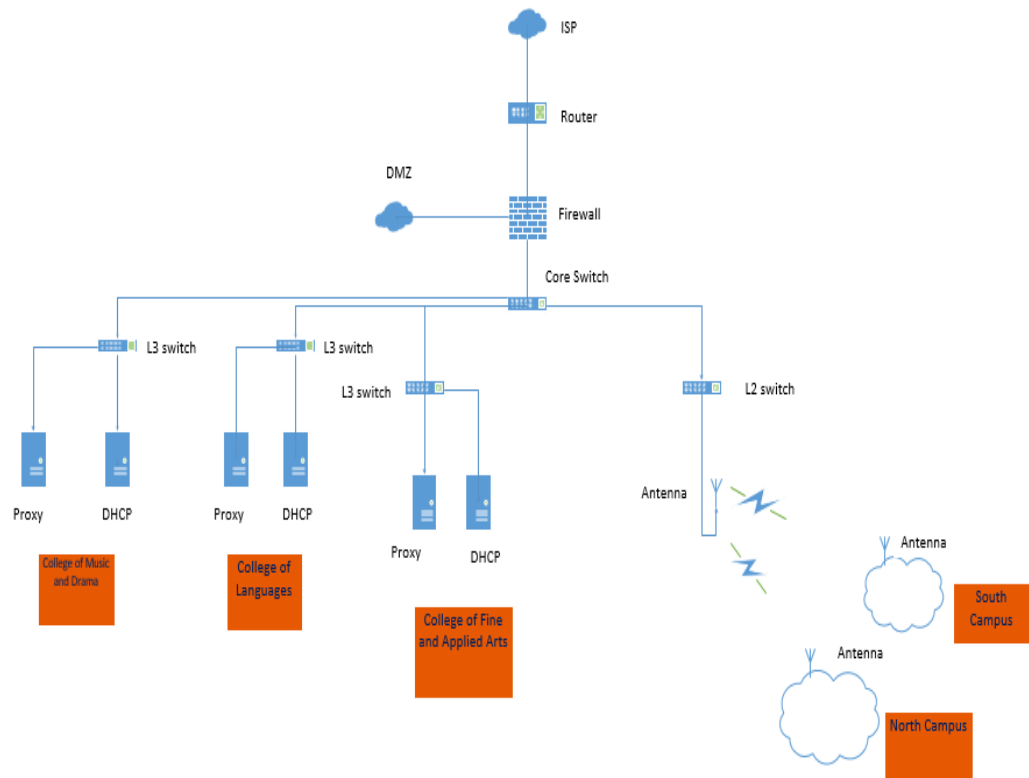


Figure 3-2: SUST's Campus Network Topology

The university network contains main router connected to the Internet Service Provider (ISP). This router is connected to the main firewall, the firewall has 3-legs connected to three areas; the DMZ area, trusted internal area and the untrusted outside internet, inside the DMZ area we find the data center that include mail server, web server containing the university website www.sustech.edu , in addition to an FTP server.

Inside the trusted area each college is connected to a core switch through a layer 3 switch, near colleges are connected with fiber optic cables, and far ones are connected with antennas. Each college contains two servers: Proxy server and DHCP server.

This topology was simplified in order to give clear implementation of SDN concept, the simplified topology contain basic three leg topology that have three areas, The DMZ area, The trusted area, and the untrusted area. This topology will be explained in more details later. As shown in Figure 3-3 below.

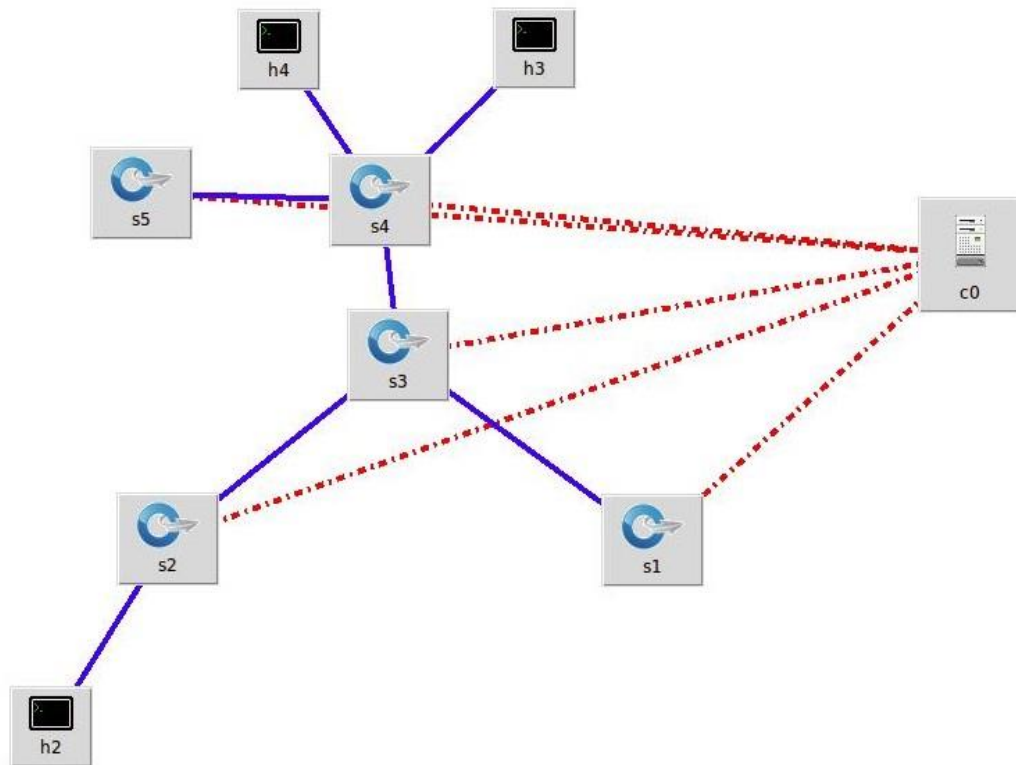


Figure 3-3: Simplified Topology in Mininet

3.2.3 Choosing the suitable environment

After studying different operating systems, Ubuntu as a stable, reliable operating system was chosen to run the virtual topology using Mininet (the emulator).

3.2.4 Choosing the emulator

Three emulators were under consideration (Mininet, NS-3, Estinet). Mininet was chosen because it provides multiple namespaces for each OpenVswitch and host, in addition to that it is widely used in research.

Mininet is an SDN emulator that runs virtual switches that support OpenFlow protocol -explained in the next section- Which is used for constructing our topology.

3.2.5 Choosing the controller

There is a collection of SDN controllers available to implement OpenFlow protocol to control the switches and act as the aggregated control plane, this collection includes OpenDayLight, POX, NOX, Floodlight... Etc. the comparison between all these controllers is discussed in chapter two of this project, the output of this comparison led to choosing OpenDayLight.

3.3 Proposed system block diagram:

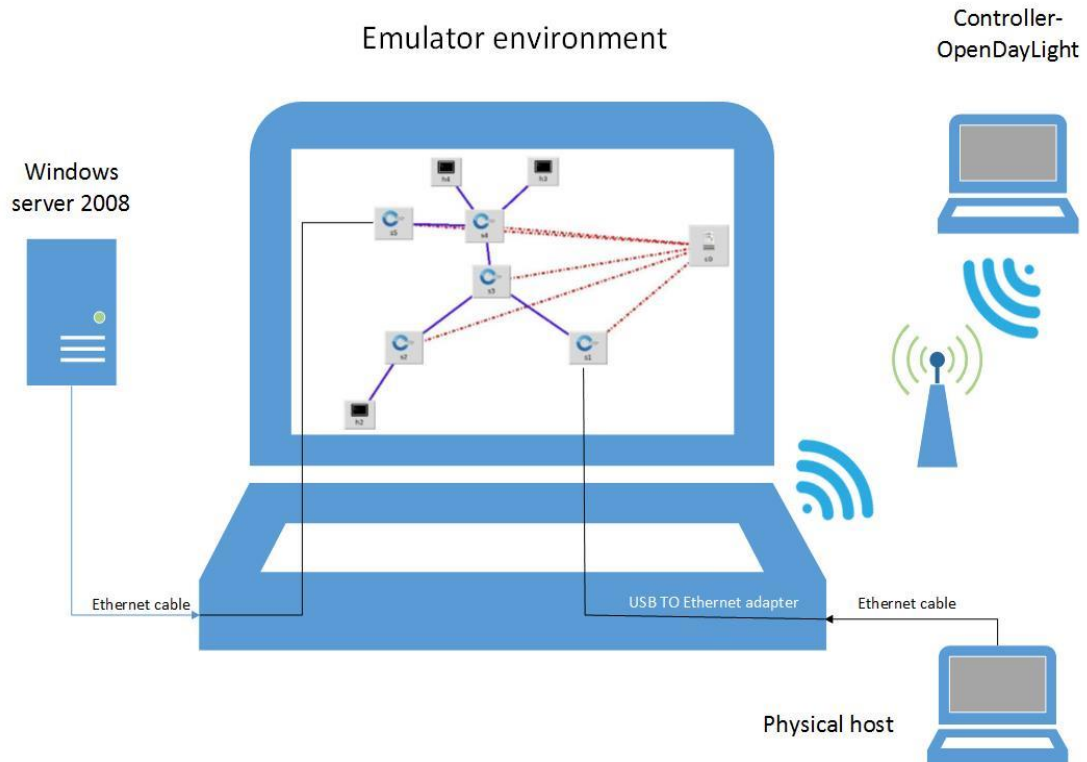


Figure 3-4: System Block Diagram

The system block diagram is composed of a main computer containing the Mininet emulator as shown in Figure 3-4; this topology is the output of Miniedit which is an extension of Mininet that creates visual topologies from codes. Part of the topology- which is a server containing Windows server 2008 and a host with windows operating system- are departed from the emulation environment in order to be represented as physical devices, all of the Open Virtual Switches (OpenVswitch- OVS) are connected with the main controller through OpenFlow protocol. The controller used is OpenDayLight (ODL).

The server is connected directly to the laptop containing the emulation environment with Ethernet connection, also the physical host is connected to an external Ethernet adapter, this adapter is attached to

the main laptop with USB port, the controller is connected to the main laptop with the wireless adapter.

3.4 Environment tools and technology:

Brief information of the tools and technologies, and how they are used and deployed is presented.

3.4.1 Oracle VM VirtualBox

VirtualBox is a cross-platform virtualization application. For one thing, it installs on the existing Intel or AMD-based computers, whether they are running Windows, Mac, Linux or Solaris operating systems. Secondly, it extends the capabilities of the computer so that it can run multiple operating systems (inside multiple virtual machines) at the same time. Here Oracle VM VirtualBox is used to run Ubuntu operating system in the Laptop that represents the controller.

3.4.2 Mininet network emulator

Mininet is a network emulator. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. Mininet tool allows complex topology testing without any physical connections, only using Python Application Programming Interface (API) and it also includes a Command Line Interface (CLI) which is topology-aware.

One feature of Mininet is Miniedit; it allows editing and configuring the topology through graphical user interface (GUI).

Mininet is used in the project as the emulation environment, it installs in the main laptop and it acts as the emulation of the topology.

3.4.3 OpenDayLight controller

The OpenDayLight Project is a collaborative open source project hosted by The Linux Foundation. The goal of the project is to accelerate the adoption of software-defined networking (SDN) and create a solid foundation for Network Functions Virtualization (NFV). The software is written in Java.

OpenDayLight acts as the controller that is able to configure the switches, it can manipulate and redirect the flows on each switch using OpenFlow protocol, the controller can be thought of as an aggregation of all the control planes of the switches.

The controller here is installed on a separate individual laptop; it connects to the main laptop containing Mininet through the wireless adapter.

3.4.4 Wireshark

Wireshark is a network protocol analyzer for windows and UNIX; it offers several benefits that make it appealing for everyday use. It is aimed at both the journeyman and the expert packet analyst, and offers a variety of features to entice each.

3.5 Configuration and Software

In this part all steps to build and integrate the system will be explained in details.

3.5.1 The controller

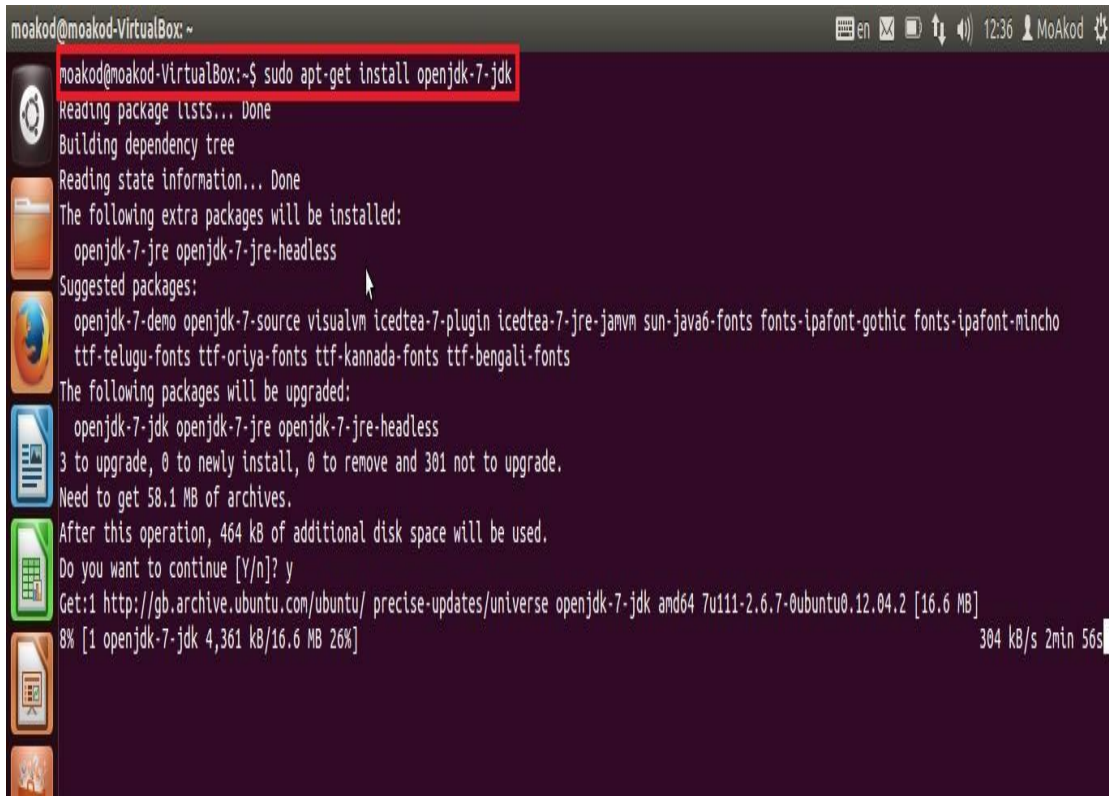
The controller is written in java language thus in order to install it the first step is to install java virtual machine (JVM), to install JVM on Ubuntu operating system the following commands on Figure 3-5 and Figure 3-6 should be executed:



```
moakod@moakod-VirtualBox: ~  
moakod@moakod-VirtualBox:~$ sudo apt-get update  
Get:1 http://extras.ubuntu.com precise release.gpg [71 B]  
Get:2 http://security.ubuntu.com precise-security Release.gpg [198 B]  
Get:3 http://gb.archive.ubuntu.com precise Release.gpg [198 B]  
Get:4 http://archive.canonical.com precise Release.gpg [198 B]  
Hit http://extras.ubuntu.com precise Release  
Get:5 http://security.ubuntu.com precise-security Release [55.5 kB]  
Get:6 http://archive.canonical.com precise Release [8,180 B]  
Get:7 http://gb.archive.ubuntu.com precise-updates Release.gpg [198 B]  
Get:8 http://gb.archive.ubuntu.com precise-backports Release.gpg [198 B]  
Hit http://extras.ubuntu.com precise/main amd64 Packages  
Get:9 http://archive.canonical.com precise/partner amd64 Packages [6,999 B]  
Hit http://gb.archive.ubuntu.com precise Release  
Hit http://extras.ubuntu.com precise/main i386 Packages  
Get:10 http://gb.archive.ubuntu.com precise-updates Release [55.4 kB]  
Ign http://extras.ubuntu.com precise/main TranslationIndex  
Get:11 http://archive.canonical.com precise/partner i386 Packages [7,867 B]  
Get:12 http://archive.canonical.com precise/partner TranslationIndex [202 B]  
Get:13 http://security.ubuntu.com precise-security/main Sources [143 kB]  
50% [10 Release 33.3 kB/55.4 kB 60%] [13 Sources 26.2 kB/143 kB 18%] [Waiting for headers] [Waiting for headers] 19.9 kB/s 6s
```

Figure 3-5: Updating packages in Ubuntu

This command -highlighted with red- on Figure 3-5 downloads the package lists from the repositories and "updates" them to get information on the newest versions of packages and their dependencies. It will do this for all repositories and PPAs. From <http://linux.die.net/~man/8/apt-get>. After this command downloads all the dependences, the following command is executed:

A terminal window titled 'moakod@moakod-VirtualBox: ~' showing the execution of the command 'sudo apt-get install openjdk-7-jdk'. The terminal output includes: 'Reading package lists... Done', 'Building dependency tree', 'Reading state information... Done', 'The following extra packages will be installed: openjdk-7-jre openjdk-7-jre-headless', 'Suggested packages: openjdk-7-demo openjdk-7-source visualvm icedtea-7-plugin icedtea-7-jre-jamvm sun-java6-fonts fonts-ipafont-gothic fonts-ipafont-mincho ttf-telugu-fonts ttf-oriya-fonts ttf-kannada-fonts ttf-bengali-fonts', 'The following packages will be upgraded: openjdk-7-jdk openjdk-7-jre openjdk-7-jre-headless', '3 to upgrade, 0 to newly install, 0 to remove and 301 not to upgrade.', 'Need to get 58.1 MB of archives.', 'After this operation, 464 kB of additional disk space will be used.', 'Do you want to continue [Y/n]? y', 'Get:1 http://gb.archive.ubuntu.com/ubuntu/ precise-updates/universe openjdk-7-jdk amd64 7u111-2.6.7-0ubuntu0.12.04.2 [16.6 MB]', and a progress bar for the first package: '8% [1 openjdk-7-jdk 4,361 kB/16.6 MB 26%]' with a speed of '304 kB/s' and time of '2min 56s'.

```
moakod@moakod-VirtualBox:~$ sudo apt-get install openjdk-7-jdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  openjdk-7-jre openjdk-7-jre-headless
Suggested packages:
  openjdk-7-demo openjdk-7-source visualvm icedtea-7-plugin icedtea-7-jre-jamvm sun-java6-fonts fonts-ipafont-gothic fonts-ipafont-mincho
  ttf-telugu-fonts ttf-oriya-fonts ttf-kannada-fonts ttf-bengali-fonts
The following packages will be upgraded:
  openjdk-7-jdk openjdk-7-jre openjdk-7-jre-headless
3 to upgrade, 0 to newly install, 0 to remove and 301 not to upgrade.
Need to get 58.1 MB of archives.
After this operation, 464 kB of additional disk space will be used.
Do you want to continue [Y/n]? y
Get:1 http://gb.archive.ubuntu.com/ubuntu/ precise-updates/universe openjdk-7-jdk amd64 7u111-2.6.7-0ubuntu0.12.04.2 [16.6 MB]
8% [1 openjdk-7-jdk 4,361 kB/16.6 MB 26%] 304 kB/s 2min 56s
```

Figure 3-6: Installing JVM

This command installs JVM version 7, Apache maven should be installed to manage the controller objects.

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

To install Maven, the following steps should be taken:

1. Download Apache Maven 3.0.4 binary from repository using the following command:

```
wget http://archive.apache.org/dist/maven/binaries/apache-maven-3.0.4-bin.tar.gz
```

2. Unzip the binary with tar:

```
tar -zxf apache-maven-3.0.4-bin.tar.gz
```

3. copy the application directory to /usr/local:

```
sudo cp -R apache-maven-3.0.4 /usr/local
```

4. Make a soft link in /usr/bin for universal access of mvn:

```
sudo ln -s /usr/local/apache-maven-3.0.4/bin/mvn /usr/bin/mvn
```

5. Verify mvn installation:

```
mvn -version  
Apache Maven 3.0.5 ...Etc.
```

OpenDayLight maintains its own repositories outside of Maven Central, which means maven, cannot resolve OpenDayLight artifacts by default. Since OpenDayLight is organized as multiple inter-dependent projects, building a particular project usually means pulling in some artifacts. In order to make this work, your maven installation needs to know the location of OpenDayLight repositories and has to be taught to use them.

This is achieved by making sure `~/.m2/settings.xml` looks something like the one on the GitHub project:

<https://github.com/opendaylight/odlparent/blob/master/settings.xml>

You can do that quickly with the following commands:

```
cp -n ~/.m2/settings.xml{,.orig}; \wget -q -O -  
https://raw.githubusercontent.com/opendaylight/odlparent/master/  
settings.xml > ~/.m2/settings.xml [22]
```

After maven has been successfully installed, the next step is to install Git to be able to download the source code of the OpenDayLight controller. Using this command:

```
sudo apt-get install git
```

Git is an open source, distributed version control system designed to handle everything from small to very large projects with

speed and efficiency. Every Git clone is a full-fledged repository with complete history and full revision tracking capabilities, not dependent on network access or a central server.

After installing git the following command is executed to download a clone of the OpenDayLight controller's source code. As shown in Figure 3-7.

A terminal window screenshot showing the execution of a git clone command. The prompt is 'moakod@moakod-VirtualBox: ~'. The command entered is 'git clone https://github.com/opendaylight/controller.git'. The output shows progress: 'Cloning into controller ...', 'remote: Counting objects: 196279, done.', 'remote: Compressing objects: 100% (149/149), done.', and 'Receiving objects: 0% (1724/196279), 692.01 KiB | 54 KiB/s'. The command line is highlighted with a red box.

```
moakod@moakod-VirtualBox: ~  
moakod@moakod-VirtualBox:~$ git clone https://github.com/opendaylight/controller.git  
Cloning into controller ...  
remote: Counting objects: 196279, done.  
remote: Compressing objects: 100% (149/149), done.  
Receiving objects: 0% (1724/196279), 692.01 KiB | 54 KiB/s
```

Figure 3-7: downloading a clone of the OpenDayLight controller's source code

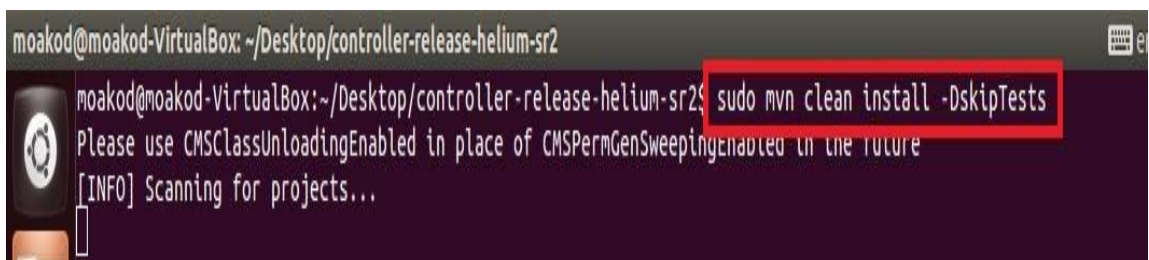
After downloading the source code, specific parameters should be sent to the JVM, this can be done by going to (.mvn) folder and editing the (jvm.config) file:

```
cd ~/Desktop/controller-release-heluim-sr2/.mvn
```

In the `jvm.config` file write the following parameters:

```
-XX:+CMSClassUnloadingEnabled  
  
-XX:+CMSPermGenSweepingEnabled  
  
-XX:+UseConcMarkSweepGC  
  
-XX:PermSize=256m  
  
-XX:MaxPermSize=512m  
  
-Xms512m  
  
-Xmx1024m
```

These parameters extend the permgen memory of jvm to 512MB. After modifying the JVM parameters, from the root folder of OpenDayLight controller we execute using the installation command in Figure 3-8 below:

A terminal window screenshot showing a command prompt. The prompt is `moakod@moakod-VirtualBox: ~/Desktop/controller-release-helium-sr2`. The command entered is `sudo mvn clean install -DskipTests`, which is highlighted with a red box. Below the command, the output shows: `Please use CMSClassUnloadingEnabled in place of CMSPermGenSweepingEnabled in the future` and `[INFO] Scanning for projects...`.

```
moakod@moakod-VirtualBox: ~/Desktop/controller-release-helium-sr2  
moakod@moakod-VirtualBox:~/Desktop/controller-release-helium-sr2$ sudo mvn clean install -DskipTests  
Please use CMSClassUnloadingEnabled in place of CMSPermGenSweepingEnabled in the future  
[INFO] Scanning for projects...
```

Figure 3-8: OpenDayLight Installation Command

The installation process takes time because it downloads packages, as seen in the Figure 3-9, the total number of downloads is 538MB.

```

moakod@moakod-VirtualBox: ~/Desktop/controller-release-helium-sr2
[INFO] sal-dom-xsql-config ..... SUCCESS [ 0.116 s ]
[INFO] sal-remotercp-connector ..... SUCCESS [ 2.535 s ]
[INFO] config-persist-adapter ..... SUCCESS [ 1.044 s ]
[INFO] config-module-archetype ..... SUCCESS [ 0.261 s ]
[INFO] protocol_plugins.stub ..... SUCCESS [ 0.742 s ]
[INFO] dummy-console ..... SUCCESS [ 0.572 s ]
[INFO] releasepom ..... SUCCESS [ 0.113 s ]
[INFO] OpenDaylight :: Karaf :: Branding ..... SUCCESS [ 1.535 s ]
[INFO] opendaylight-karaf-resources ..... SUCCESS [ 1.665 s ]
[INFO] opendaylight-karaf-empty ..... SUCCESS [ 7.588 s ]
[INFO] features-base ..... SUCCESS [ 9.685 s ]
[INFO] Features :: AD-SAL Features ..... SUCCESS [ 2.398 s ]
[INFO] OpenDaylight :: Features :: Network Service Functions ..... SUCCESS [ 0.368 s ]
[INFO] features-config ..... SUCCESS [ 0.539 s ]
[INFO] features-protocol-framework ..... SUCCESS [ 0.408 s ]
[INFO] features-netconf ..... SUCCESS [ 0.566 s ]
[INFO] features-config-persist ..... SUCCESS [ 0.533 s ]
[INFO] features-config-netty ..... SUCCESS [ 0.564 s ]
[INFO] features-akka ..... SUCCESS [ 0.713 s ]
[INFO] features-mdsal ..... SUCCESS [ 0.554 s ]
[INFO] features-flow ..... SUCCESS [ 0.501 s ]
[INFO] features-restconf ..... SUCCESS [ 1.056 s ]
[INFO] distribution.opendaylight-karaf ..... SUCCESS [ 9.070 s ]
[INFO] controller-features ..... SUCCESS [ 1.946 s ]
[INFO] extras-features ..... SUCCESS [ 2.031 s ]
[INFO] features-adsal-compatibility ..... SUCCESS [ 1.475 s ]
[INFO] features-netconf-connector ..... SUCCESS [ 0.600 s ]
[INFO] features-controller ..... SUCCESS [ 0.303 s ]
[INFO] odl-model-project ..... SUCCESS [ 1.500 s ]
[INFO] opendaylight-configfile-archetype ..... SUCCESS [ 0.327 s ]
[INFO] archetypes-parent ..... SUCCESS [ 0.427 s ]
[INFO] distribution-karaf-archetype ..... SUCCESS [ 1.638 s ]
[INFO] opendaylight-karaf-features-archetype ..... SUCCESS [ 0.435 s ]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 10:48 min
[INFO] Finished at: 2016-09-04T13:17:59+01:00
[INFO] Final Memory: 538M/1015M
[INFO] -----
moakod@moakod-VirtualBox:~/Desktop/controller-release-helium-sr2$

```

Figure 3-9: Downloading Packages for Opendaylight's Controller Installation

After the controller is successfully downloaded, simple steps are taken to run it as shown in Figure 3-10, go to the OpenDayLight folder:

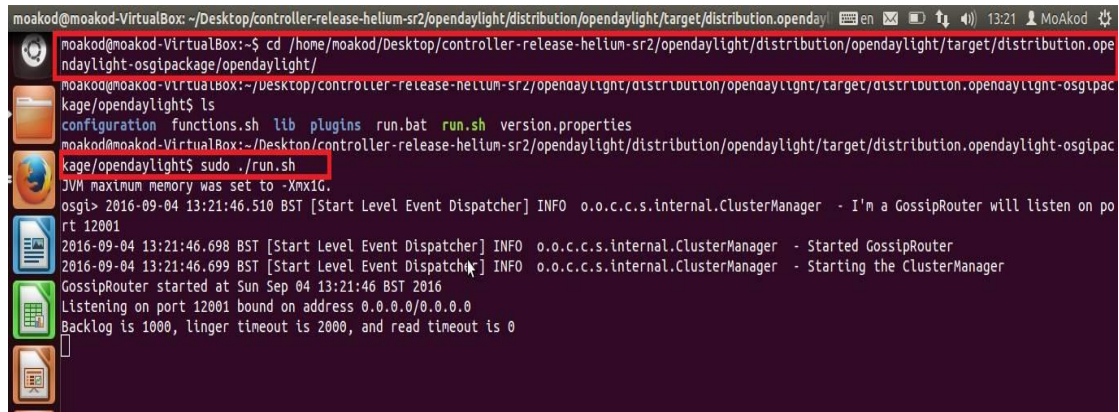
```

cd ~/Desktop/controller-release-helium-
sr2/opendaylight/distribution/opendaylight/target/distribution.op
endaylight-osgipackage/opendaylight/

```

And then run the `./run.sh` command:

```
sudo ./run.sh
```



```
moakod@moakod-VirtualBox: ~/Desktop/controller-release-helium-sr2/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipackage/opendaylight/
moakod@moakod-VirtualBox:~/Desktop/controller-release-helium-sr2/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipac
kage/opendaylight$ ls
configuration functions.sh lib plugins run.bat run.sh version.properties
moakod@moakod-VirtualBox:~/Desktop/controller-release-helium-sr2/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipac
kage/opendaylight$ sudo ./run.sh
JVM maximum memory was set to -Xmx1G.
osgi> 2016-09-04 13:21:46.510 BST [Start Level Event Dispatcher] INFO o.o.c.c.s.internal.ClusterManager - I'm a GossipRouter will listen on po
rt 12001
2016-09-04 13:21:46.698 BST [Start Level Event Dispatcher] INFO o.o.c.c.s.internal.ClusterManager - Started GossipRouter
2016-09-04 13:21:46.699 BST [Start Level Event Dispatcher] INFO o.o.c.c.s.internal.ClusterManager - Starting the ClusterManager
GossipRouter started at Sun Sep 04 13:21:46 BST 2016
Listening on port 12001 bound on address 0.0.0.0/0.0.0.0
Backlog is 1000, linger timeout is 2000, and read timeout is 0
█
```

Figure 3-10: Running the Controller

The controller runs on OSGi (Open Service Gateway Initiative) which is a Java framework for developing and deploying modular software programs and libraries. OSGi has a specification for modular components called bundles, which are commonly referred to as plug-ins.

3.5.2 The server

The server that was used had three services installed, DNS service, HTTP and FTP services as shown in Figure 3-13. The webserver (IIS) role contains both the HTTP and the FTP services.

The HTTP server contains the webpage of SUST as a default webpage (<http://sustech.edu>) as shown in Figure 3-14. The site design and configuration was taken as an example from the original sustech.edu page. So it is for demonstration purposes only.

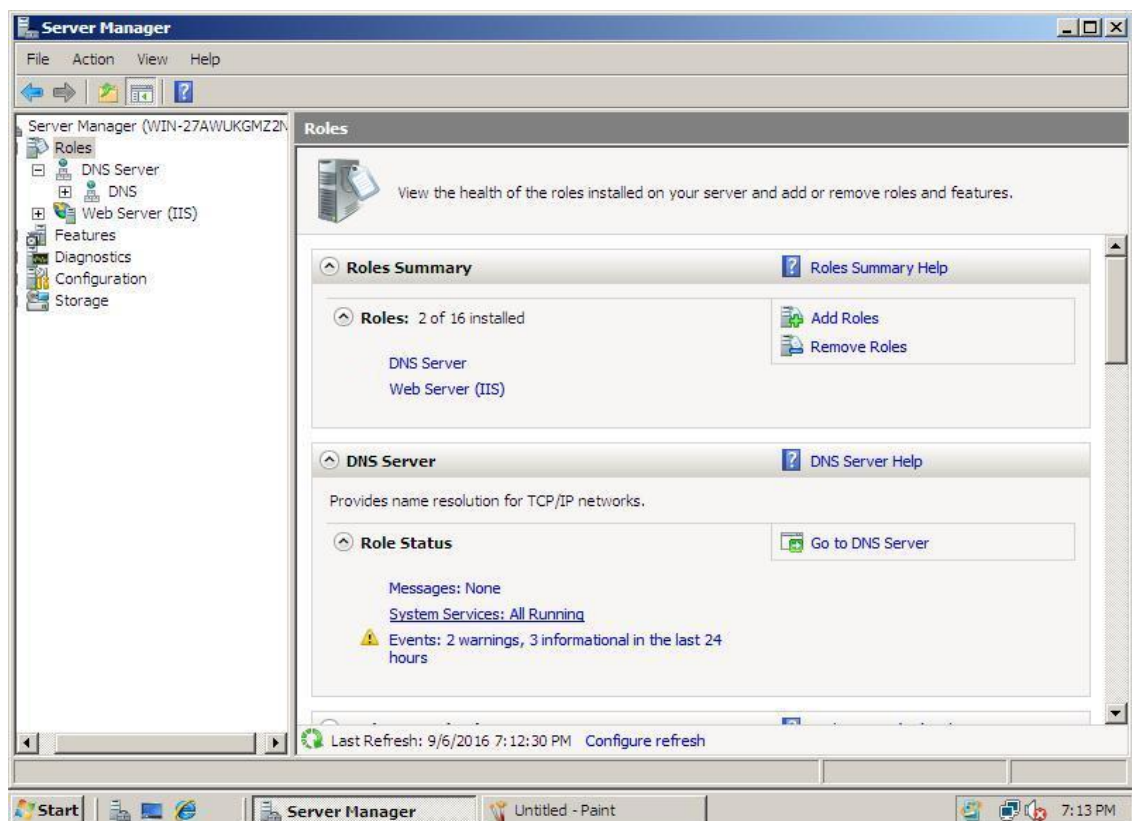


Figure 3-13: Roles on the windows 2008 server

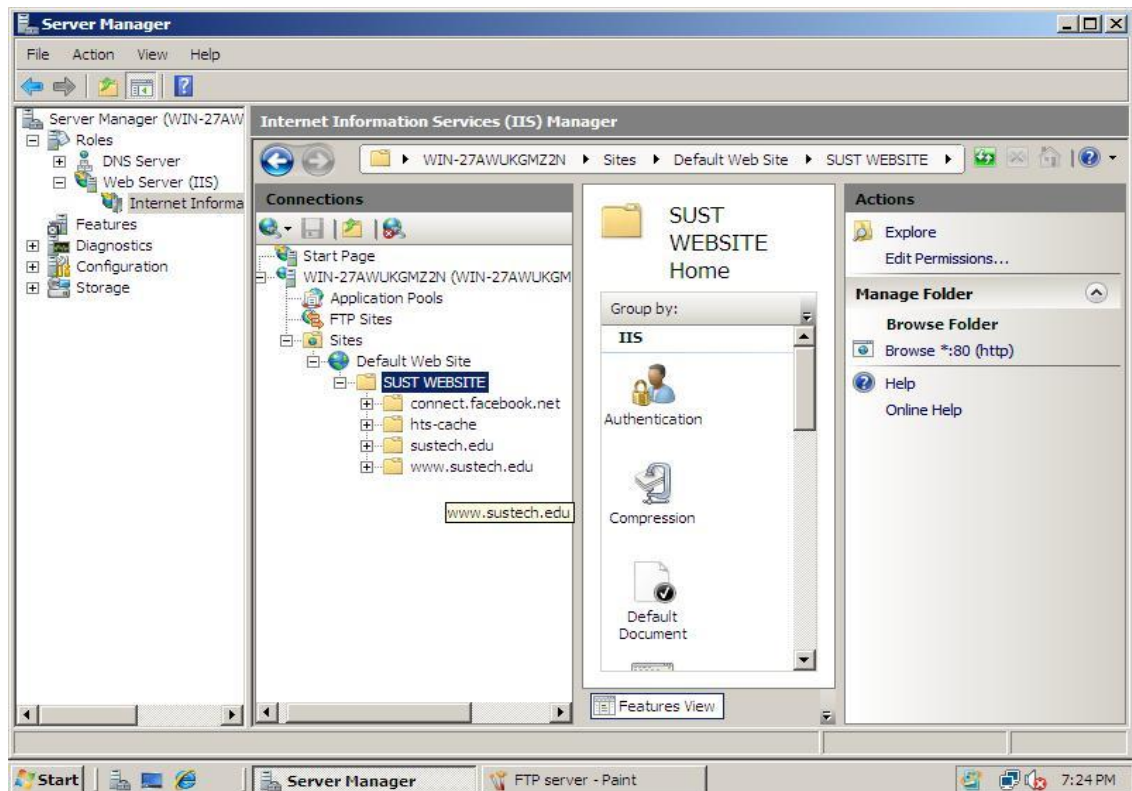


Figure 3-14: Web server (IIS) HTTP Service

As for the FTP service, a default FTP site was made including several folders as shown in Figure 3-15, configuration of the FTP role included configuring the IP address of the server and selecting the default FTP site which is (<ftp://192.168.20.2>) and locating that file in the containing folder in our server disk (C://ftp/).

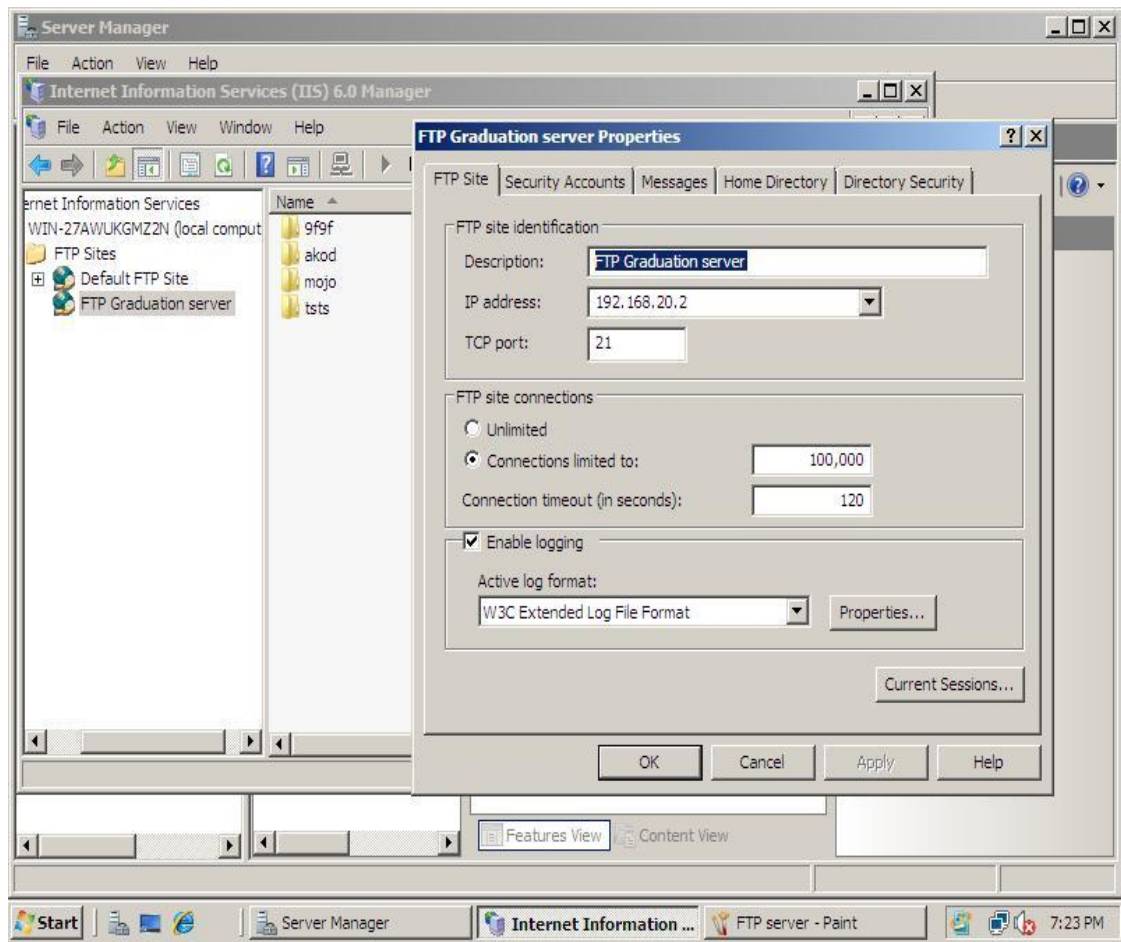


Figure 3-15: FTP Server Configuration

Several steps were performed in order to configure the DNS server, setting up the forward lookup zone with the domain name, which is in this case (m4.net) coupled with the IP address of the server. Then configuring the reverse lookup zone which is the IP address of the server (20.168.192.in-addr.arpa). That's all that was needed to configure a simple DNS service for the purpose of presenting a working environment of a client-server connection with several services in action.

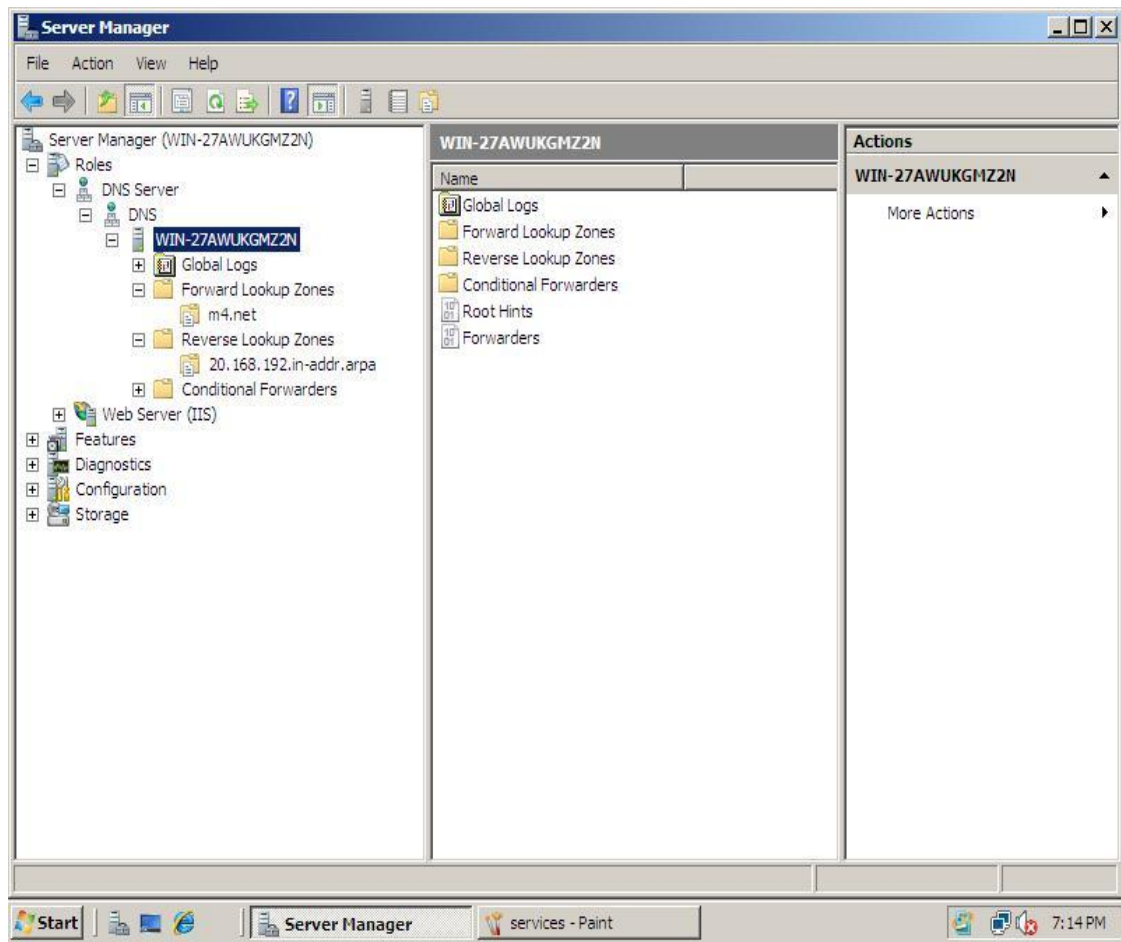


Figure 3-16: DNS Server Configuration

3.5.3 The physical Host

A laptop was used as our host connected directly to the computer that contains the Mininet emulator environment via a USB to Ethernet convertor. First the IP address of the host was configured as (192.168.10.1) and the preferred DNS server as the IP of the server that was used for the DMZ services including the DNS server (192.168.20.2). As shown in Figure 3-17:

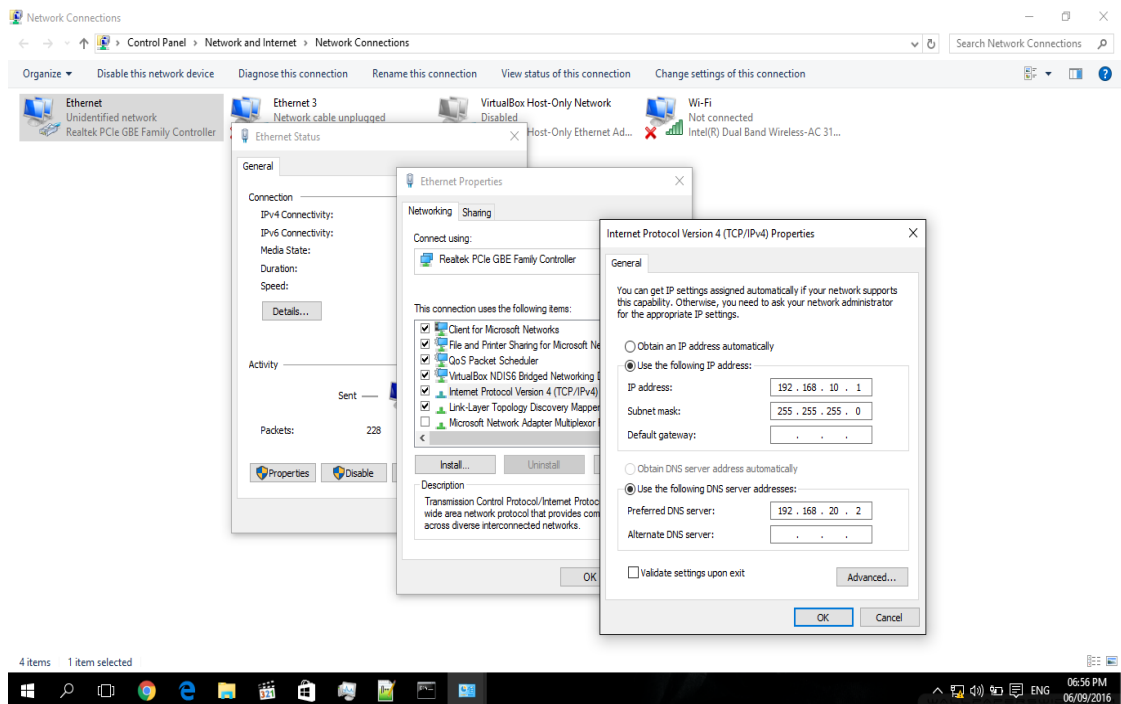


Figure 3-17: Host IP Address and preferred DNS server assignment

A default route was configured using the command in Figure 3-18 to tell packets where to go by default. If the host was connected directly to the device containing the Mininet emulation environment without using this command, packets wouldn't know the default gateway and wouldn't see the virtual network that contains the rest of the devices. And the known networks would only be the host itself and the loop back, as shown in Figure 3-18. But after adding the server route to the virtual network all packets sent to that network would know how to get there.

```
C:\WINDOWS\system32>route add 0.0.0.0 mask 0.0.0.0 192.168.10.1
OK!
```

Figure 3-18: Command-Default route adding

The added route is the one with the low metric which has a destination network of (0.0.0.0 with network mask 0.0.0.0) which means any unknown networks. Figure 3-19 and Figure 3-20 show the routing

table of the host before and after adding the default route leading to the emulation environment.

```

Administrator: Command Prompt
C:\WINDOWS\system32>route print
=====
Interface List
10...e0 94 67 b3 d2 98 .....Microsoft Wi-Fi Direct Virtual Adapter
6...50 7b 9d 87 ec 18 .....Realtek PCIe GBE Family Controller
12...00 ff e0 78 5d 0d .....Anchorfree HSS VPN Adapter
1.....Software Loopback Interface 1
13...e0 94 67 b3 d2 97 .....Intel(R) Dual Band Wireless-AC 3165
21...00 00 00 00 00 00 e0 Microsoft Teredo Tunneling Adapter
3...00 00 00 00 00 00 e0 Microsoft ISATAP Adapter #3
=====

IPv4 Route Table
=====
Active Routes:
Network Destination        Netmask          Gateway           Interface        Metric
127.0.0.0                  255.0.0.0       On-link          127.0.0.1        306
127.0.0.1                  255.255.255.255 On-link          127.0.0.1        306
127.255.255.255           255.255.255.255 On-link          127.0.0.1        306
192.168.10.0              255.255.255.0   On-link          192.168.10.1     276
192.168.10.1              255.255.255.255 On-link          192.168.10.1     276
192.168.10.255            255.255.255.255 On-link          192.168.10.1     276
224.0.0.0                  240.0.0.0       On-link          127.0.0.1        306
224.0.0.0                  240.0.0.0       On-link          192.168.10.1     276
255.255.255.255           255.255.255.255 On-link          127.0.0.1        306
255.255.255.255           255.255.255.255 On-link          192.168.10.1     276
=====
Persistent Routes:
None

```

Figure 3-19 Active routes before adding default route

```

Select Administrator: Command Prompt
C:\WINDOWS\system32>route print
=====
Interface List
10...e0 94 67 b3 d2 98 .....Microsoft Wi-Fi Direct Virtual Adapter
6...50 7b 9d 87 ec 18 .....Realtek PCIe GBE Family Controller
12...00 ff e0 78 5d 0d .....Anchorfree HSS VPN Adapter
1.....Software Loopback Interface 1
13...e0 94 67 b3 d2 97 .....Intel(R) Dual Band Wireless-AC 3165
21...00 00 00 00 00 00 e0 Microsoft Teredo Tunneling Adapter
3...00 00 00 00 00 00 e0 Microsoft ISATAP Adapter #3
=====

IPv4 Route Table
=====
Active Routes:
Network Destination        Netmask          Gateway           Interface        Metric
0.0.0.0                    0.0.0.0          On-link          192.168.10.1     21
127.0.0.0                  255.0.0.0       On-link          127.0.0.1        306
127.0.0.1                  255.255.255.255 On-link          127.0.0.1        306
127.255.255.255           255.255.255.255 On-link          127.0.0.1        306
192.168.10.0              255.255.255.0   On-link          192.168.10.1     276
192.168.10.1              255.255.255.255 On-link          192.168.10.1     276
192.168.10.255            255.255.255.255 On-link          192.168.10.1     276
224.0.0.0                  240.0.0.0       On-link          127.0.0.1        306
224.0.0.0                  240.0.0.0       On-link          192.168.10.1     276
255.255.255.255           255.255.255.255 On-link          127.0.0.1        306
255.255.255.255           255.255.255.255 On-link          192.168.10.1     276
=====
Persistent Routes:
None

```

Figure 3-20 Active networks after adding the default route

After configuring the host with IP address, preferred DNS server and default route, it shall be able to connect to the emulation environment and DMZ servers.

It can now connect and display the folders in the FTP site using the URL <ftp://192.168.20.2> (IP of the server) as shown in Figure 3-21:

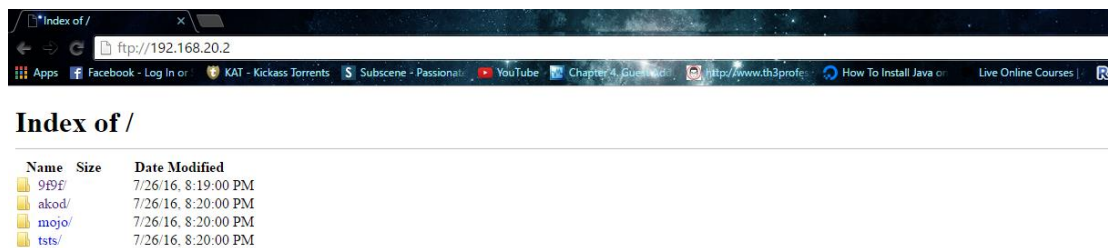


Figure 3-21: Browsing the FTP site on the Host Browser

And it can also browse the default webpage using either the IP address of the server or the domain name of the webpage (<http://192.168.20.2> or <http://m4.net>) as shown in Figure 3-22 and Figure 3-23 below:

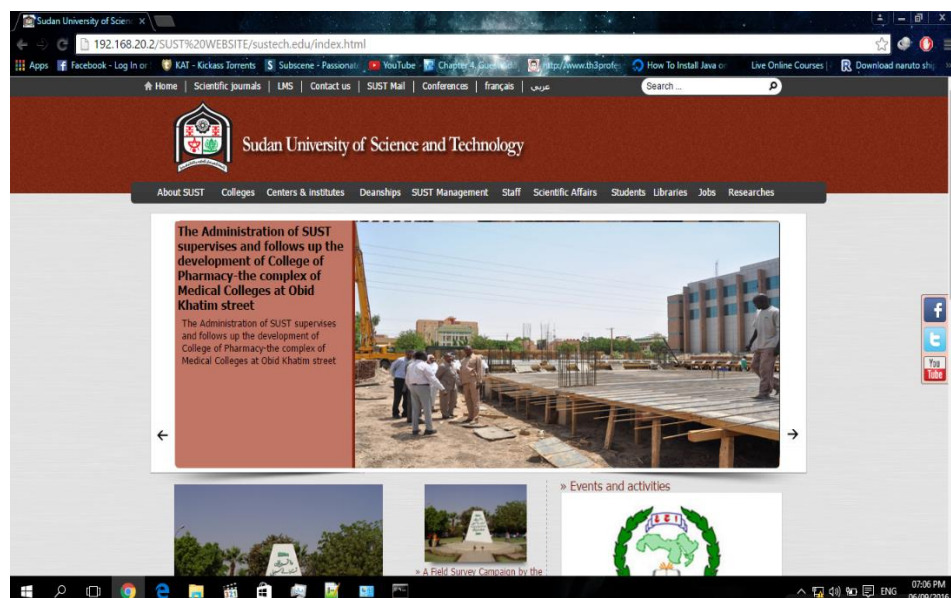


Figure 3-22: Browsing the Default webpage using IP address

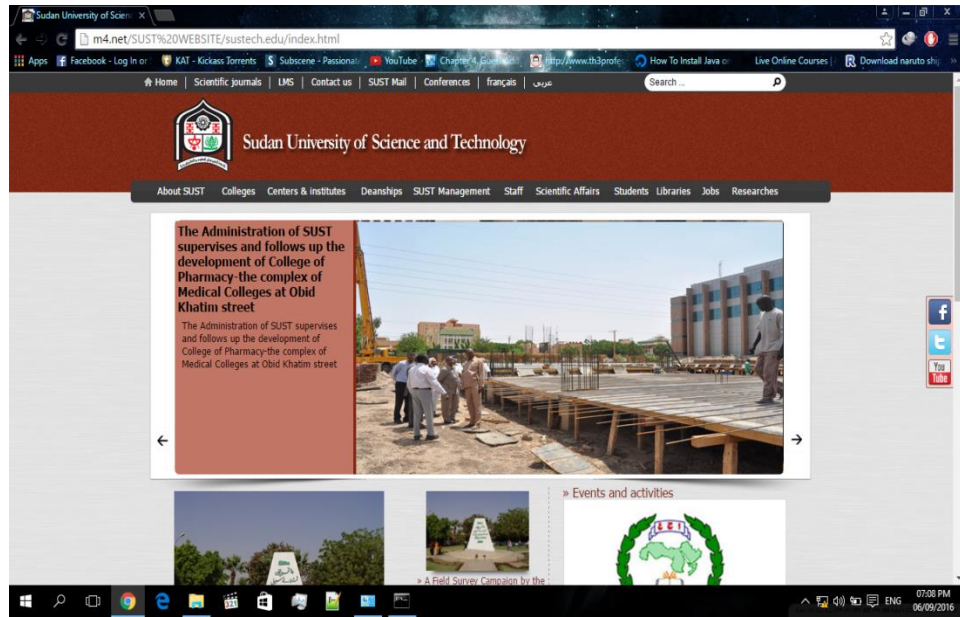


Figure 3-23 Browsing the Default webpage using the domain name

CHAPTER FOUR

RESULTS

4. RESULTS

4.1 Controller to switches Communication

4.2 Topology Connectivity

4.3 HTTP connectivity

4.4 FTP Connectivity

This chapter verifies the pervious setup and checks the limitations of the proposed topology. The verification process will start by analyzing the communication between the controller and the OpenV Switches, then a simple connectivity test will be held from various nodes to another, as topology designed, there is no open reachability between hosts, some hosts may not reach another ones, the policies used will be showed in this chapter ,this connectivity test will be followed by checking the HTTP protocol configuration on the server, and also capturing a sample of HTTP packets traveling from one of the hosts to the HTTP server, and also the same process will be repeated for the FTP server. The DNS verification will be embedded on HTTP verification. The last part of this chapter will be showing a sample of how the controller is configured to drop traffic based on specific policies.

4.1 Controller to switches Communication:

The switch initiates a standard TCP (or TLS) connection to the controller. When an OpenFlow connection is established, each entity must send a hello message; an OFPT_HELLO message with the protocol version set to the highest OpenFlow protocol version supported by the sender. The hello message is followed by a reply OFPT_ECHO_REPLY.

No.	Time	Source	Destination	Protocol	Length	Info
2576	80.190571	192.168.43.167	192.168.43.3	OpenFlow	74	Type: OFPT_ECHO_REPLY
2577	80.195852	192.168.43.167	192.168.43.3	OpenFlow	86	Type: OFPT_STATS_REQUEST
2578	80.224689	192.168.43.167	192.168.43.3	OpenFlow	78	Type: OFPT_STATS_REQUEST
2586	80.253912	192.168.43.167	192.168.43.3	OpenFlow	98	Type: OFPT_STATS_REQUEST
2589	80.262767	192.168.43.167	192.168.43.3	OpenFlow	86	Type: OFPT_STATS_REQUEST

Figure 4-1: Displaying the Openflow packets in Wireshark

This figure is a capture from Wireshark packet capturing application, the source here is a single IP address that describes the

emulation environment, which means this link between 192.168.43.167 and 192.168.43.3 carries the traffic between all the switches and the single controller. As seen in Figure 4-1 above the protocol used is OpenFlow, and the message type is an OFPT_ECHO_REPLY.

Here as seen below in Figure 4-2 is a closer look at the content of one of the OpenFlow message types, OFPT_PACKET_OUT, which can be seen as an instruction from the controller to the switch.

No.	Time	Source	Destination	Protocol	Length	Info
2117	69.749863	a6:cc:9e:a6:15:06	Broadcast	OpenFlow	132	Type: OFPT_PACKET_OUT
2119	69.750443	a6:cc:9e:a6:15:06	Broadcast	OpenFlow	132	Type: OFPT_PACKET_OUT
2120	69.752116	a6:cc:9e:a6:15:06	Broadcast	OpenFlow	132	Type: OFPT_PACKET_OUT
2122	69.752812	a6:cc:9e:a6:15:06	Broadcast	OpenFlow	132	Type: OFPT_PACKET_OUT
2123	69.753114	a6:cc:9e:a6:15:06	Broadcast	OpenFlow	132	Type: OFPT_PACKET_OUT


```

> Frame 2120: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits) on interface 0
> Ethernet II, Src: HonHaiPr_ac:7f:d1 (d8:5d:e2:ac:7f:d1), Dst: HonHaiPr_of:09:93 (18:f4:6a:0f:09:93)
> Internet Protocol Version 4, Src: 192.168.43.167, Dst: 192.168.43.3
> Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 34504 (34504), Seq: 7773, Ack: 128223, Len: 66
  OpenFlow 1.0
    .000 0001 = Version: 1.0 (0x01)
    Type: OFPT_PACKET_OUT (13)
    Length: 66
    Transaction ID: 1066680283
    Buffer Id: 0xffffffff
    In port: 65535
    Actions length: 8
    Actions type: Output to switch port (0)
    Action length: 8
    Output port: 3
    Max length: 0
  Ethernet II, Src: a6:cc:9e:a6:15:06 (a6:cc:9e:a6:15:06), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  Address Resolution Protocol (request)
  
```

Figure 4-2: Content of Openflow message types

This instruction is an OpenFlow packet sent from the controller to all the switches, it is a broadcast packet, this can be seen on the highlighted red boxes on Figure 4-2, additional information that can be seen from the figure is the OpenFlow protocol version which is version 1.0, also the type of OpenFlow message is OFPT_PACKET_OUT, which is a request to send packet out of specific port, here as mentioned it is a broadcast.

4.2 Topology Connectivity

To verify the connectivity in this topology first a demonstration on the policies should be provided, Figure 4-3 shows three different areas, the internal area (GREEN area in the figure), the DMZ area (RED area in the figure), and the internet (BLACK area in the figure), devices within a single area can communicate without any restrictions. Also internal devices and internet devices can reach the server on the DMZ area. Only one devices of the internet area can reach the internal area, and the other one is rejected.

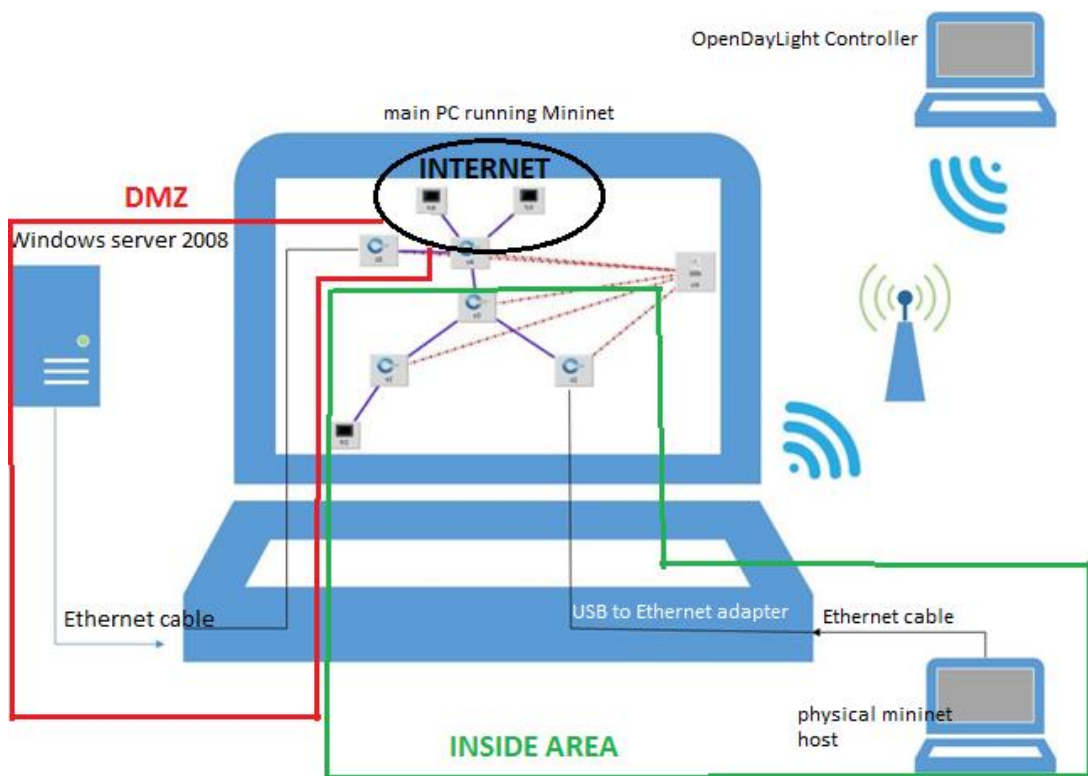


Figure 4-3: Different Areas of the Topology(DMZ, Internet, Internal Area)

The following figure, Figure 4-4 shows a successful ICMP echo and echo reply messages (ICMP ping) sent from the inside area (device IP 192.168.30.1) to the server in the DMZ (IP 192.168.20.2).

```

Host: h3
64 bytes from 192.168.20.2: icmp_seq=111 ttl=128 time=0,791 ms
64 bytes from 192.168.20.2: icmp_seq=112 ttl=128 time=0,680 ms
64 bytes from 192.168.20.2: icmp_seq=113 ttl=128 time=0,664 ms
64 bytes from 192.168.20.2: icmp_seq=114 ttl=128 time=0,674 ms
64 bytes from 192.168.20.2: icmp_seq=115 ttl=128 time=0,686 ms
64 bytes from 192.168.20.2: icmp_seq=116 ttl=128 time=0,681 ms
64 bytes from 192.168.20.2: icmp_seq=117 ttl=128 time=0,657 ms
64 bytes from 192.168.20.2: icmp_seq=118 ttl=128 time=0,670 ms
64 bytes from 192.168.20.2: icmp_seq=119 ttl=128 time=0,668 ms
64 bytes from 192.168.20.2: icmp_seq=120 ttl=128 time=0,704 ms
64 bytes from 192.168.20.2: icmp_seq=121 ttl=128 time=0,667 ms
64 bytes from 192.168.20.2: icmp_seq=122 ttl=128 time=0,691 ms
64 bytes from 192.168.20.2: icmp_seq=123 ttl=128 time=0,694 ms
64 bytes from 192.168.20.2: icmp_seq=124 ttl=128 time=0,689 ms
64 bytes from 192.168.20.2: icmp_seq=125 ttl=128 time=0,702 ms
64 bytes from 192.168.20.2: icmp_seq=126 ttl=128 time=0,692 ms
64 bytes from 192.168.20.2: icmp_seq=127 ttl=128 time=0,671 ms
64 bytes from 192.168.20.2: icmp_seq=128 ttl=128 time=0,718 ms
64 bytes from 192.168.20.2: icmp_seq=129 ttl=128 time=0,685 ms
64 bytes from 192.168.20.2: icmp_seq=130 ttl=128 time=0,685 ms
64 bytes from 192.168.20.2: icmp_seq=131 ttl=128 time=0,718 ms
64 bytes from 192.168.20.2: icmp_seq=132 ttl=128 time=0,687 ms
64 bytes from 192.168.20.2: icmp_seq=133 ttl=128 time=0,692 ms
64 bytes from 192.168.20.2: icmp_seq=134 ttl=128 time=0,672 ms
64 bytes from 192.168.20.2: icmp_seq=135 ttl=128 time=0,708 ms
64 bytes from 192.168.20.2: icmp_seq=136 ttl=128 time=0,685 ms
64 bytes from 192.168.20.2: icmp_seq=137 ttl=128 time=0,682 ms
64 bytes from 192.168.20.2: icmp_seq=138 ttl=128 time=0,704 ms
64 bytes from 192.168.20.2: icmp_seq=139 ttl=128 time=0,688 ms
64 bytes from 192.168.20.2: icmp_seq=140 ttl=128 time=0,704 ms
^C
--- 192.168.20.2 ping statistics ---
140 packets transmitted, 140 received, 0% packet loss, time 139003ms
rtt min/avg/max/mdev = 0,650/1,969/170,587/14,302 ms

```

Figure 4-4: Successful ICMP Ping from Internal Area to DMZ

Figure 4-5 shows a blocked ICMP messages from the internet host 2.2.2.2 to the internal area.

```

C:\WINDOWS\system32>ping 2.2.2.2

Pinging 2.2.2.2 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 2.2.2.2:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

```

Figure 4-5: Blocked ICMP ping from internet host to internal Area

4.3 HTTP connectivity

The initial configuration of the HTTP server configuration demonstrated in chapter 3, here a simple test is made from the physical host to the server, by opening a web browser in the host and typing the IP address of server.

The first test is done without involving domain name system (DNS).

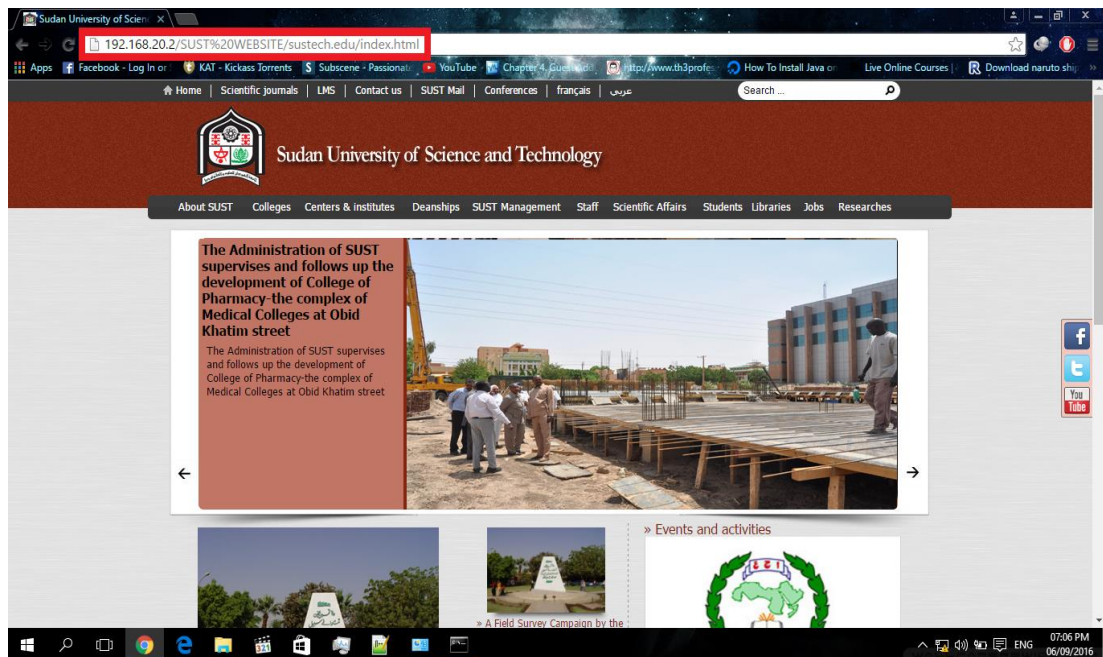


Figure 4-6: Testing Connectivity from Physical Host to HTTP Server

Figure 4-6 shows the first test were only the IP address is used.

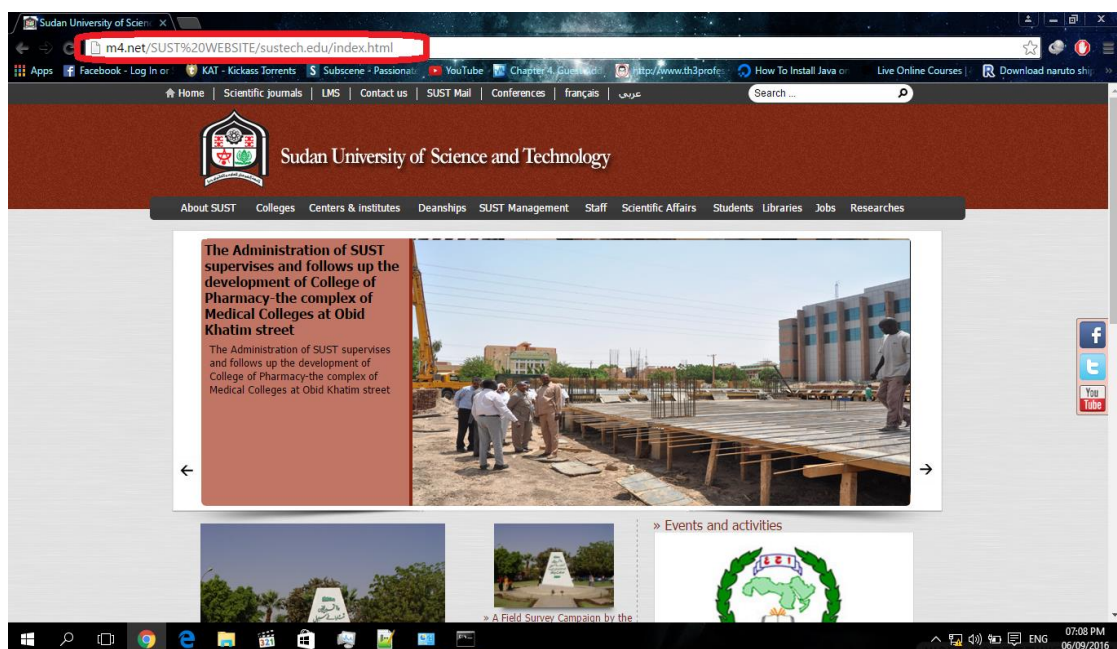


Figure 4-7: Testing Connectivity from Physical Host to HTTP Server via URL

Figure 4-7 shows the second test where a uniform resource locator (URL) is used instead of IP address.

HTTP protocol can also be verified from Wireshark as in Figure 4-8 below

16	0.004092000	192.168.10.1	192.168.20.2	TCP	60 51096 > http [ACK] Seq=1 Ack=1 Win=65536 Len=0
17	0.004714000	192.168.10.1	192.168.20.2	TCP	60 51699 > http [ACK] Seq=1 Ack=1 Win=65536 Len=0
18	0.007317000	192.168.10.1	192.168.20.2	TCP	60 51700 > http [ACK] Seq=1 Ack=1 Win=65536 Len=0
19	0.013398000	192.168.10.1	192.168.20.2	HTTP	537 GET / HTTP/1.1
20	0.015965000	192.168.20.2	192.168.10.1	HTTP	242 HTTP/1.1 304 Not Modified
21	0.067145000	192.168.10.1	192.168.20.2	TCP	60 51695 > http [ACK] Seq=484 Ack=189 Win=65280 Len=0

Figure 4-8: HTTP Verification in Wireshark

All the above can verify the HTTP protocol connectivity, The DNS system can also be verified further by using NSLOOKUP tool in the host.

Figure 4-9 shows NSLOOKUP tool which tells the IP address corresponding to m4.net.

```
C:\WINDOWS\system32>nslookup m4.net
Server: m4.net
Address: 192.168.20.2
```

Figure 4-9: NSLOOKUP verifying DNS Server

4.4 FTP Connectivity

File transfer protocol FTP can also be verified from the web browser of the physical host; Figure 4-10 shows a successful FTP access from host to server.



Index of /

Name	Size	Date Modified
9f9f/		7/26/16, 8:19:00 PM
akod/		7/26/16, 8:20:00 PM
mojo/		7/26/16, 8:20:00 PM
tsts/		7/26/16, 8:20:00 PM

Figure 4-10: Host Accessing the FTP server

Also a Wireshark can provide a better resolution that shows the source IP and the destination IP. Figure 4-11 shows the Wireshark capture.

13	3.060306000	192.168.10.1	192.168.20.2	FTP	60 Request: SYST
14	3.060858000	192.168.20.2	192.168.10.1	FTP	70 Response: 215 Windows_NT
15	3.061473000	192.168.10.1	192.168.20.2	FTP	60 Request: PWD
16	3.062005000	192.168.20.2	192.168.10.1	FTP	85 Response: 257 "/" is current directory.
17	3.062481000	192.168.10.1	192.168.20.2	FTP	62 Request: TYPE I
18	3.062976000	192.168.20.2	192.168.10.1	FTP	74 Response: 200 Type set to I.

Figure 4-11: FTP Verification in Wireshark

CHAPTER FIVE

CONCLUSION AND FUTURE WORK

5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

5.2 Future work

5.1 Conclusion

This thesis explores the implementation of SDN in a campus network, and the migration from a traditional to SDN based network. The process starts with gathering information about the traditional campus network, its design, implementation, configuration, and deployment. From that information, a sample from the network is taken to transition it to an SDN based network, the sample consists of the major parts of the network connecting to the ISP (the main router, the firewall, and the DMZ), and two departments to represent their connection to the main router. The topology is deployed in a Miniedit emulator. The network is configured (as detailed in chapter 3) by configuring the OpenDayLight controller for all the policies needed.

The presented work also highlights the problems of dynamic networks in terms of configurability and the need to look at SDN as an approach or architecture to not only simplify the network but also make it more reactive to the requirements of workload and services placed in the network. One of the main questions addressed was to demonstrate that SDN presents a smooth solution for controlling and programming dynamic networks.

As the simulation environment is set and running a few network access control polices (NAC) has been performed on different locations on the network to demonstrate an aspect of network security on the network. These polices are verified through several tests that testify reachability, the reachability test is done on different protocols including (ICMP, HTTP, FTP).

To test and evaluate the system functionality as a whole a group of packet capturing has been performed on the data plane and the control plane separately, the control plane capturing is performed to verify OpenFlow functionality, and the packet capturing on the data plane is performed to verify normal dataflow based on the packet eligibility to reach the destination.

5.2 Future work

Although SDN is rapidly evolving to provide a solid foundation for future network solutions, especially in the field of increasing flexibility and support for event-based dynamic network control there is always a room for future work.

This thesis gave a base for migration from traditional to SDN based network, we believe that future work should include using different types of controllers which has advantages over the OpenDayLight to compare which type is better for campus networks.

Also we think that a thorough comparison between a traditional network and an SDN based network is very important to observe the differences and the best choice for campuses.

Another important implementation is to add a redundant controller to the network. Redundancy is crucial for SDN controllers to achieve lossless and low delay performance. So the number of OpenFlow switches managed by one controller should be limited. Also redundancy provides higher availability, so if one controller is down, the network will keep running normally. Therefore, we can argue that adding a redundant controller or even several controllers is one of the important issues (if not the most important) that should be focused on in the future.

REFERENCES

- [1] Software Defined Networking [Video file]. (2014, June 10). In www.youtube.com. Retrieved January 15, 2016, from <https://www.youtube.com/playlist?list=PLpherdrLyny-OTgZzILTcbMIDtLdNuXcT>
- [2] Sakir Sezer, Sandra Scott-Hayward, P. K. Chouhan, et al. “Are we ready for SDN?” Implementation challenges for software-defined networks *Communications Magazine, IEEE*, Vol. 51, No. 7. July 2013.
- [3] Palo Alto. “Software-defined networking: The new norm for networks,” White Paper, Apr. 2012.
- [4] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek, “The click modular router,” *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp.263–297, Aug. 2000.
- [5] M. Handley, O. Hodson, and E. Kohler, “XORP: An open platform for network research,” *ACM SIGCOMM Computer Commun. Rev.*, vol.33,no. 1, pp. 53–57, Jan. 2003.
- [6] Quagga Routing Software Suite. [Online]. Available: <http://www.nongnu.org/quagga/>
- [7] The BIRD Internet Routing Daemon. [Online]. Available: <http://bird.network.cz/>
- [8] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der

Merwe, “The case for separating routing from routers,” in Proc. ACM SIGCOMM Workshop FDNA, 2004, pp. 5–12.

[9] L. Yang, R. Dantu, T. Anderson, and R. Gopal, Forwarding and Control Element Separation (ForCES) Framework, Apr. 2004, RFC 3746.[Online]. Available:

http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[10] A. Doria et al., Forwarding and Control Element Separation (ForCES) Protocol Specification, Mar. 2010, RFC 5810. [Online]. Available:

http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[11] J. Rexford et al., “Network-wide decision making: Toward a wafer-thin control plane,” in Proc. HotNets, 2004, pp. 59–64.

[12] H. Yan et al., “Tesseract: A 4D network control plane,” in Proc. 4th USENIX Conf. NSDI, 2007, p. 27.

[13] A. Farrel, J.-P. Vasseur, and J. Ash, A Path Computation Element (PCE)-Based Architecture, Aug. 2006, RFC 4655. [Online]. Available:

http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[14] M. Casado et al., “SANE: A protection architecture for enterprise networks,” in Proc. 15th Conf. USENIX-SS, Berkeley, CA, USA, 2006

- [15] M. Casado et al., “Rethinking enterprise network control,” *IEEE/ACM Trans. Netw.*, vol. 17, no. 4, pp. 1270–1283, Aug. 2009. vol. 15, p. 10.
- [16] Smith, J. M., & Nettles, S. M. (2014, January 30). Active networking : one view of the past, present, and future. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 34(1), 4–18.
- [17] J. Networks, “Network Operating System Evolution,” Juniper Networks, 2010.
- [18] OpenFlow version 1.3 tutorial | SDN Hub. Retrieved October 10, 2016, from <http://sdnhub.org/tutorials/openflow-1-3/>
- [19] SDN in the Campus Environment ONF Solution Brief September 30, 2013
- [20] @sdxcentral. (2013, July 2). Six Campus Networks SDN Use Cases That You Need to Know About. Retrieved October 18, 2016, from <https://www.sdxcentral.com/articles/contributed/sdn-use-cases-campus-networks/2013/07/> (website)
- [21] SDN Migration Considerations and Use Cases ONF Solution Brief November 21, 2014 ONF TR - 506
- [22] Getting Started: Development Environment Setup. Retrieved April 25, 2016, from https://wiki.opendaylight.org/view/gettingstarted:development_environment_setup#edit_your_.7e.2f.m2.2fsettings.xml