



SUDAN UNIVERSITY OF SCIENCE & TECHNOLOGY
COLLEGE OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

Software-Defined Networking for Datacenters

**A THESIS SUBMITTED AS PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF B.Sc. (HONORS) IN COMPUTER
SYSTEMS AND NETWORKS**

OCTOBER 2017

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

SUDAN UNIVERSITY OF SCIENCE & TECHNOLOGY
FACULTY OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

Software-Defined Networking for Datacenters

PREPARED BY:

STUDENT: Khaled Fath-Alrhman Mohamed Ahmed

STUDENT: Obay Mohamed Al-Hassan Abadi

SUPERVISED BY:

Dr. Niemah Izzeldin Mohamed Osman

**A THESIS SUBMITTED AS A PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF B.Sc. (HONORS) IN COMPUTER
SYSTEMS & NETWORKS**

SIGNATURE OF SUPERVISOR:

.....

DATE:

.....

الآية

قال تعالى :

{وَاخْفِضْ لَهُمَا جَنَاحَ الذُّبَابِ مِنَ الرَّحْمَةِ وَقُلْ رَبِّ ارْحَمْهُمَا كَمَا رَبَّيْتَنِي صَغِيرًا }

الاسراء (24)

الوفاء:

الي كل من اضاء بعلمه عقل غيره او هدى بالجواب الصحيح حيرة سائليه ، فأظهر
بسماعته تواضع العلماء وبرحابته سماحة العارفين.

شكر وعرفان

اشكر الله ان جمعني بكم في هذه الجامعة من معلمين ومربين يعجز لساني عن شكركم وتستحي نفسي من نكران جميلكم، كنتم لي خير معلمين فتعلمت منكم طول البال والصبر، وعرفانا مني بسعة صدركم وتفهمكم ولين جنبكم اقول:

ولو انني اوتيت حسن الخطابة*** ورافيت بدر النطق في الشعر والنثر

كنت بعد القول الا مقصرا*** ومعترفا بالعجز عن واجب الشكر

نشكر اولاً بعد الله عز وجل الدكتوراة : نعمة عز الدين - الموقرة و المحترمة علي المعلومات والتوجيهات بخصوص هذا المشروع ونشكر جميع الاساتذة- المحترمين، على مساعدتهم لنا وكنتم عند حسن الظن بكم مشرفين ومعلميين واداريين ونتمني ان نكون قد اوفينا هذا البحث حقه.

Abstract

Software-Defined Networking (SDN) is a new networking design approach which is spreading fast. This approach is based on the separation of data and control planes, which offer the network operator certain advantages in terms of centralized programmatic control. This centralized approach of management and control maintains a global view of the network rather than managing tens of thousands of lines of configuration scattered among thousands of network devices. Software-defined networks aim to provide high flexibility to modify network state and behavior.

This thesis aims to explore the new emerging paradigm Software-Defined Networking and test its implementation and demonstrate how to implement SDN concepts within a datacenter network taking Sudan University of Science and Technology datacenter network as a case study. The project also implements load balancing with the help of software. The new SDN approach reduces the cost, offers flexibility in configuration, reduces time to deploy, provides automation and facilitates building a network without requiring the knowledge of any vendor-specific software/hardware.

The project uses Mininet emulation environment, and POX 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 datacenter based network controlled by the POX controller.

المستخلص

تطور مؤخرًا بسرعة كبيرة مفهوم جديد في تصميم الشبكات يدعى الشبكات المعرفة برمجياً. يعتمد هذا المفهوم على الفصل بين وحدتي البيانات والتحكم، والذي يتيح لمشغل الشبكة مزايا من حيث التحكم البرمجي المركزي. يوفر هذا النهج المركزي من حيث الإدارة والتحكم نظرة شاملة للشبكة بدلاً من إدارة عشرات الآلاف من خطوط الضبط المنتشرة بين الآلاف من أجهزة الشبكات. الشبكات المعرفة بالبرمجيات تهدف إلى إضافة قدر عالي من المرونة لتعديل حالة الشبكة وسلوكياتها في توجيه البيانات.

يهدف هذا البحث إلى استكشاف نموذج الشبكات المعرفة برمجياً الناشئ حديثاً ويختبر ويبين كيفية تطبيقها على شبكات مراكز البيانات، أخذاً شبكة مركز البيانات في جامعة السودان للعلوم والتكنولوجيا كدراسة حالة. كما يقوم المشروع بتنفيذ موازنة التحميل بمساعدة البرمجيات. يقلل النهج الجديد من تكلفة الشبكة و يتيح المرونة في الضبط ويقلل الوقت اللازم للتشغيل ويوفر التشغيل الآلي ويسهل بناء الشبكة دون الحاجة لمعرفة الأجهزة أو البرمجيات الخاصة بالمورد.

يستخدم هذا المشروع بيئة (MININET) لمحاكاة الأجهزة الحقيقية، ومتحكم (POX) للتحكم في هذه البيئة. تم تجميع أجزاء بيئة المحاكاة من أجهزة مختلفة لتكوين النظام. ويتم التحكم بها باستخدام متحكم خاص. بنهاية هذا المشروع تم الحصول على شبكة مركز بيانات معرف برمجياً يتم التحكم فيها بواسطة متحكم (POX).

Table of Contents

ABSTRACT.....	IV
المستخلص.....	V
CHAPTER 1 INTRODUCTION.....	1
1.1 PREFACE	2
1.2 THE PROBLEM STATEMENT	3
1.3 PROPOSED SOLUTION	4
1.4 AIMS AND OBJECTIVES	4
1.5 SCOPE	5
1.6 METHODOLOGY	5
1.7 THESIS OUTLINE	5
CHAPTER 2 SOFTWARE-DEFINED NETWORKING	7
2.1 INTRODUCTION TO SDN	8
2.1.1 SDN Background.....	8
2.1.2 SDN Precursors	9
2.1.3 SDN Definition	9
2.1.4 Network Programmability	10
2.1.5 Control plane and data plane separation.....	10
2.1.6 SDN Motivation.....	11
2.2 SDN ARCHITECTURE.....	13
2.3 SDN CONTROLLERS.....	15
2.3.1 NOX:.....	16
2.3.2 POX:	16
2.3.3 Ryu:.....	16
2.3.4 Floodlight:.....	16
2.3.5 OpenDayLight:	17
2.4 OPENFLOW PROTOCOL.....	18
2.4.1 OpenFlow Structure.....	18
2.4.2 OpenFlow Switch	18
2.5 SDN IN DATACENTERS	20
2.6 SDN MIGRATION.....	22
2.6.1 Google legacy-to-hybrid migration	23
2.7 NETWORK VIRTUALIZATION.....	25
2.8 SUMMARY.....	26
CHAPTER 3 LITERATURE REVIEW, TOOLS AND TECHNIQUES	27
3.1 INTRODUCTION	28
3.2 LITERATURE REVIEW	28
3.2.1 Event-Driven Network Control Using Software-Defined Networking.....	28

3.2.2	Software Defined Networking based Data-Center Services	29
3.2.3	Programmable and Scalable Software-Defined Networking Controllers ...	29
3.2.4	Implementation of Remote Configuration Using SDN Approach	30
3.2.5	Comparing a Commercial and an SDN-Based Load Balancer in Campus Network.....	30
3.2.6	Implementation of SDN in a Campus NAC Use Case	31
3.3	TOOLS AND TECHNIQUES	33
3.3.1	Oracle VM VirtualBox	33
3.3.2	Linux Operating System	33
3.3.3	Mininet.....	34
3.3.4	POX Controller.....	34
3.3.5	Python	35
3.3.6	Wireshark Network Analyzer	35
3.3.7	USB to Ethernet Adapter	36
3.4	SUMMARY.....	36
CHAPTER 4 DESIGN AND METHODOLOGY		37
4.1	INTRODUCTION	38
4.2	THE NETWORK TOPOLOGY.....	38
4.3	OPERATING SYSTEM.....	39
4.4	THE EMULATOR.....	40
4.5	THE CONTROLLER	41
4.6	LOAD BALANCING.....	42
4.6.1	Load balancing algorithms and methods	43
4.6.2	Round-Robin algorithm:	43
4.7	SOFTWARE CONFIGURATION.....	45
4.7.1	Mininet Installation.....	45
4.7.2	POX Controller Installation.....	48
4.7.3	The Servers	49
4.8	SUMMARY.....	50
CHAPTER 5 IMPLEMENTATION AND RESULTS		51
5.1	INTRODUCTION	52
5.2	IMPLEMENTATION.....	52
5.3	SIMULATION RESULTS.....	53
5.3.1	The Hub Scenario	54
5.3.2	The Switch Scenario	55
5.3.3	Load Balancer Scenario.....	56
5.4	EXTERNAL DEVICES RESULTS.....	57
5.4.1	Hub Connectivity.....	58
5.4.2	Switch Connectivity.....	59
5.4.3	HTTP Connectivity.....	60

5.4.4 FTP Connectivity.....	61
5.4.5 Load Balancer.....	62
5.5 SUMMARY.....	62
CHAPTER 6 CONCLUSIONS AND FUTURE WORK.....	63
6.1 CONCLUSIONS.....	64
6.2 FUTURE WORK.....	65
REFERENCES:	66

List of Figures

FIGURE 1.1 A FULLY CONNECTED HIERARCHICAL TOPOLOGY OF A DATACENTER NETWORK	4
FIGURE 2.1 (A) TRADITIONAL NETWORK VIEW COMPARED WITH (B) SDN NETWORK VIEW.....	11
FIGURE 2.2 THE OPEN SDN ARCHITECTURE.....	13
FIGURE 2.3 EXAMPLES OF THE MOST WELL-KNOWN SDN CONTROLLERS.....	15
FIGURE 2.4 OPENFLOW PACKET MATCH FIELDS.....	18
FIGURE 2.5 OPENFLOW SWITCH MODULES.....	19
FIGURE 2.6 OPENFLOW CONTROLLER AND SWITCH.....	20
FIGURE 2.7 THE HIERARCHICAL TOPOLOGY OF A DATACENTER NETWORK.....	21
FIGURE 2.8 SDN MIGRATION STEPS.....	23
FIGURE 2.9 GOOGLE B4 STARTING NETWORK.....	24
FIGURE 2.10 GOOGLE B4 PHASED DEPLOYMENT MIXED NETWORK.....	24
FIGURE 2.11 GOOGLE B4 TARGET NETWORK.....	25
FIGURE 3.1 USB TO ETHERNET ADAPTER.....	36
FIGURE 4.1 SUST DATACENTER NETWORK TOPOLOGY.....	38
FIGURE 4.2 SERVICES INSIDE THE DMZ.....	39
FIGURE 4.3 LOAD BALANCER.....	42
FIGURE 4.4 ROUND-ROBIN ALGORITHM FLOWCHART.....	44
FIGURE 4.5 UPDATING PACKAGES ON UBUNTU.....	45
FIGURE 4.6 INSTALLING GIT.....	46
FIGURE 4.7 INSTALLING MININET.....	47
FIGURE 4.8 MINIEDIT.....	47
FIGURE 4.9 RUNNING POX CONTROLLER.....	48
FIGURE 4.10 DOCUMENT ROOT FOR SUST'S PAGE.....	49
FIGURE 5.1 NETWORK TOPOLOGY WITH EXTERNAL DEVICES.....	52
FIGURE 5.2 POX CONTROLLER GUI.....	53
FIGURE 5.3 REQUESTING THE SERVICE IN HUB SCENARIOS.....	54
FIGURE 5.4 BROADCASTING OF PACKETS IN THE HUB SCENARIOS.....	54
FIGURE 5.5 REQUESTING THE SERVICE IN THE SWITCH SCENARIOS.....	55
FIGURE 5.6 EXCHANGE OF PACKETS IN THE SWITCH SCENARIO.....	55
FIGURE 5.7 RUN LOAD BALANCER.....	56
FIGURE 5.8 DISTRIBUTING OF TRAFFIC AMONG SERVERS.....	57
FIGURE 5.9 ICMP PING BY THE HUB.....	58
FIGURE 5.10 BROADCASTING OF PACKETS IN THE HUB SCENARIO.....	58

FIGURE 5.11 ICMP PING BY SWITCH.....	59
FIGURE 5.12 SOURCE AND DESTINATION EXCHANGE PACKETS IN THE SWITCH SCENARIOS.....	59
FIGURE 5.13 HTTP REQUEST.....	60
FIGURE 5.14 EXCHANGE OF HTTP PACKETS.....	60
FIGURE 5.15 FTP REQUEST.....	61
FIGURE 5.16 EXCHANGE OF FTP PACKETS.....	61
FIGURE 5.17 DISTRIBUTING OF TRAFFIC AMONG SERVERS.....	62

List of Tables

TABLE 2.1 COMPARISON BETWEEN SDN AND CONVENTIONAL NETWORKING	12
TABLE 2.2 SUMMARY OF SOME PROPERTIES OF SOME CONTROLLERS.....	17
TABLE 3.1 COMPARISON OF PREVIOUS WORK AND PROPOSED SYSTEM.....	32

List of Abbreviations

ADC	Application Delivery Controller
API	Application Programming Interface
CLI	Command-Line Interface
CPU	Central Processing Unit
DNS	Domain Name Server
DMZ	Demilitarized Zone
E/IBGP	External/ Internal Border Gateway Protocol
FIB	Forwarding Information Base
FTP	File Transfer Protocol
ForCES	Forwarding and Control Element Separation
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
IETF	Internet Engineering Task Force
IDS	Intrusion Detection System
ICMP	Internet Control Message Protocol
ISIS	Intermediate System to Intermediate System
JVM	Java Virtual Machine
NAT	Network Address Translation
NAC	Network Access Control
NEC	Nippon Express Co., Ltd.
NVF	Network Virtualization Function
NOS	Networking Operating System
ONF	Open Networking Foundation
OS	Operating System
OVS	Open Virtual Switch
OVSDB	Open-vSwitch Database Management Protocol
PCEP	Path Computation Element Protocol
PHD	Piled Higher and Deeper
QoS	Quality of Service
RCP	Routing Control Platform
SDN	Software-Defined Networking
SNMP	Simple Network Management Protocol
SSL	Secure Sockets Layer
SUST	Sudan University of Science and Technology
TCAM	Ternary Content-Addressable Memory
TCO	Total Cost of Ownership

VM	Virtual Machine
VMS	Virtual Memory System
VLAN	Virtual local Area Network
WAN	Wide Area Network

Chapter 1 INTRODUCTION

- 1.1** Preface
- 1.2** The problem statement
- 1.3** Proposed Solution
- 1.4** Aims and Objectives
- 1.5** Scope
- 1.6** Methodology
- 1.7** Thesis Outline

This chapter provides a brief overview of SDN, problem statement and the solution proposed, aims and objectives, and the technologies that are used to develop the project, in addition to thesis outline.

1.1 Preface

In networking devices, there exist three planes: data plane, control plane and management plane. Data plane refers to the hardware part where forwarding takes place, while control plane refers to the software part where all network logic and intelligence takes place. Typically, in networking devices, control plane consists of firmware developed and maintained by vendors only. Management plane is typically a part of control plane and is used for network monitoring and controlling purposes. In this thesis, we focus on the data and control planes.

Software-defined networking (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 network elements 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.

SDN changes the way of designing, configuring and managing networks. By decoupling the control plane from the data plane the chance of providing secure network is increased. And with a centralized controller the overall view and management of a network becomes much easier. While SDN discusses the centralization of the controller, Network Virtualization Function (NMF) 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 NAT, firewalling, IDS, DNS, and caching in a unified device that may be a real hardware or a virtualized device (controller).

Among many benefits, SDN eliminates the rigidity present in traditional network and make it easier to build applications for enterprise networks, datacenters, 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 [1].

1.2 The problem statement

Datacenter networks are designed for satisfying the data transmission demand of densely interconnected hosts in the datacenter. The network topology and routing mechanism can affect the performance and latency significantly. Network engineers also adopt load balancing methods in the design of routing algorithms. However, the requirement of load balancing routing in datacenter networks cannot be fully satisfied by traditional approaches. The main reason is the lack of efficient ways to obtain network traffic statistics from each network device.

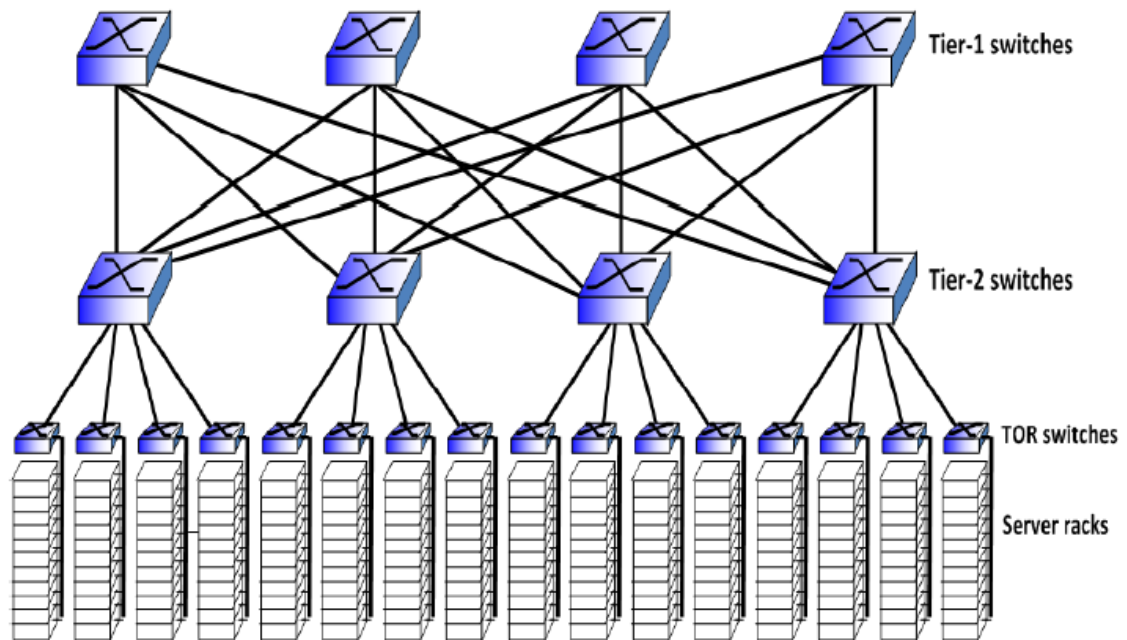


Figure 1.1 A fully connected hierarchical topology of a datacenter network

1.3 Proposed Solution

As a solution, The OpenFlow protocol enables monitoring traffic statistics by a centralized controller. To achieve high performance and low latency, we propose a load balancer for OpenFlow based datacenter networks.

1.4 Aims and Objectives

The aim of this project is to test and examine the behavior of SDN environment on datacenter topology and different types of packets (Control packets, Data packets, etc.).

The objectives of this project are to:

- Develop a brief implementation of SDN on datacenter network.
- Implement load balancing on the SDN network.
- Test and evaluate the network connectivity.

1.5 Scope

The project deploys Software-Defined Networking in the Sudan University datacenter. It implements three major datacenter functions: hub, switch and load balancer in the datacenter network.

1.6 Methodology

In this project we simulate a datacenter network using Mininet network simulator and this network is managed and controlled by POX controller. The controller is developed using Python programming language to implement three different scenarios: Hub, Switch and Load Balancer. We use round-robin algorithm to apply load balancing and we use Wireshark network analyzer to test and evaluate the connectivity. In addition, three actual servers are connected to the emulator through external interfaces, and the connectivity of HTTP and FTP services are verified using Wireshark.

1.7 Thesis Outline

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 six main chapters as follows:

Chapter Two:

The chapter presents some basic concepts of Software-Defined Networking and some other concepts that are relevant to this thesis.

Chapter Three:

This chapter reviews related works to SDN and describes the tools and technologies used in the implementation phase.

Chapter Four:

The chapter describe the implementation phase. Both network virtualization and SDN tools were used. The test case scenarios are presented with explanation of the commands used and detailed overview of the Mininet emulator and controller architecture.

Chapter Five:

This chapter shows the results obtained from the three network use cases Hub, Switch and Load Balancer. It also introduces the expectations regarding the SDN benefits to the applied use cases and consists of an evaluation of the obtained results.

Chapter Six:

The chapter aims to draw the final remarks and conclusions of the presented work. Proposed optimization and complementary future work are also presented.

Chapter 2 SOFTWARE-DEFINED NETWORKING

- 2.1** Introduction to SDN
- 2.2** SDN Architecture
- 2.3** SDN controllers
- 2.4** OpenFlow Protocol
- 2.5** SDN in datacenters
- 2.6** SDN Migration
- 2.7** Network virtualization

2.1 Introduction to SDN

The concept of Software-Defined Networking is not new and completely revolutionary; rather it arises as the result of contributions, ideas, and development in research networking. The three most important states in the evolution of SDN: Active Networks (mid-90s to early 2000), separation of data and control planes (2001–2007), and the OpenFlow API and networking operating systems "NOS" (2007–2010).

Recently, SDN has developed to become a major research topic for both researcher and network industry. In this chapter, we describe the basic concepts of Software-Defined Networking and some other concepts that are relevant to this thesis. This includes SDN controllers, datacenter technology and OpenFlow protocol, SDN migration and network virtualization.

2.1.1 SDN Background

Network configuration and installation requires highly skilled personnel to adept the configuration of many network elements. Where interactions between network nodes (switches, routers, etc.) are complex, a more systems-based approach encompassing elements of simulation is required. With the current programming interfaces on much of today's networking equipment, this is difficult to achieve. In addition, operational costs involved in provisioning and managing large multivendor networks covering multiple technologies have been increasing over recent years.

The term Software-Defined Networking (SDN) has been coined recently. 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), 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].

2.1.2 SDN Precursors

SDN improved the flexibility of network control by simplifying the designed network hardware. While SDN has received a large amount of attention from the industry, it is worth noting that decoupling control logic and programmable network ideas have been around for quite some time. In this section, we provide an overview of early programmable networking efforts, precursors to the current SDN paradigm that laid the foundation for many of the ideas we are seeing today.

The Open Signaling (OPENSIG) working group began in 1995. The core of their proposal was to provide access to the network hardware via open, programmable network interfaces. In the mid-1990s, the Active Networking initiative proposed the idea of a network infrastructure that would be programmable for customized services. The 4D project advocated the separation between the routing decision logic and the protocols governing the interaction between network elements [3].

2.1.3 SDN Definition

The Open Networking Foundation (ONF) is a nonprofit consortium dedicated to development, standardization, and commercialization of SDN. ONF has provided the most explicit and well received definition of SDN as follows:

“Software-Defined Networking (SDN) is an emerging network architecture where network control is decoupled from forwarding and is directly programmable” [4].

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.

2.1.4 Network Programmability

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. SwitchWare is an active networking solution, allowing packets flowing through a network to modify operations of the network dynamically. Similarly, software routing suites on conventional PC hardware, such as Click, XORP, Quagga, and BIRD, 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.

2.1.5 Control plane and data plane separation

The concept of decoupling between control and data planes has been proliferated during the last decade. In 2004, Caesar et al. presented a Routing Control Platform (RCP) to replace BGP inter-domain routing with centralized routing control to reduce complexity. In the same year, IETF presented the Forwarding and Control Element Separation (ForCES) framework, to separate control and packet forwarding elements [5].

2.1.6 SDN Motivation

SDN has the potential to simplify network management, and enable innovation in the evolution of computer networks. It is based on the principle of separating the control and data planes.

2.1.6.1 Networking the Old Way

In traditional networks, as shown in Figure 2.1(a), the control and data planes are combined in a network node.

The control plane is responsible for the 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 the use of mobile devices, and the impact of “big data.”

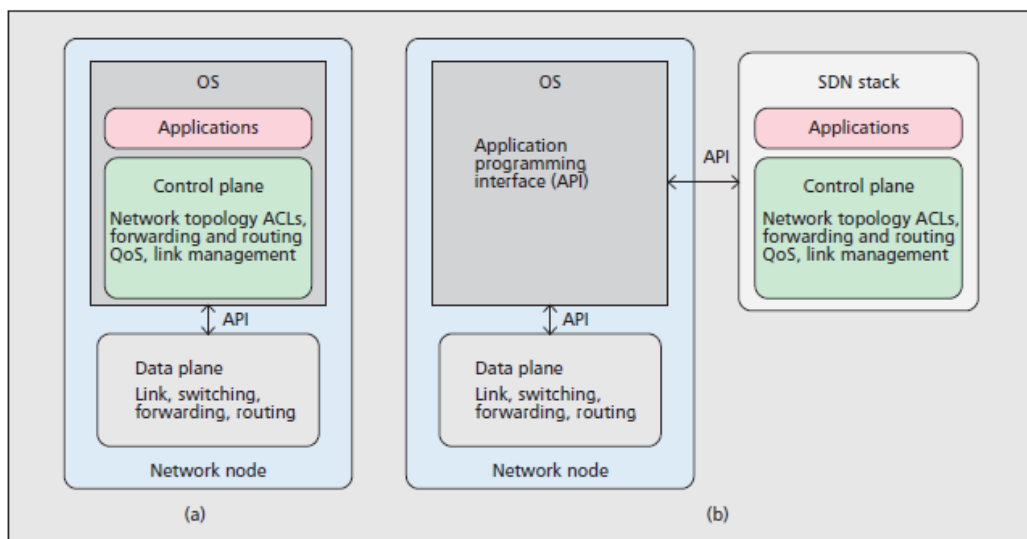


Figure 2.1 (a) Traditional network view compared with (b) SDN network view.

2.1.6.2 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 Figure 2.1(b) and manipulates their forwarding plane, providing an abstract model of the network topology to the SDN controller hosting the application. Table 2.1 compares SDN and conventional networking.

Table 2.1 Comparison between SDN and Conventional Networking

	SDN	Traditional network
Features	Decouple data and control plane, and programmability.	A new protocol per problem, complex network control.
Configuration	Automated configuration with centralized validation.	Error prone manual configuration.
Performance	Dynamic global control with cross layer information.	Limited information, and relatively static configuration.
Innovation	Easy software implementation for new ideas, sufficient test environment with isolation, and quick deployment using software upgrade.	Difficult hardware implementation for new ideas, limited testing environment, long standardization process.

2.2 SDN Architecture

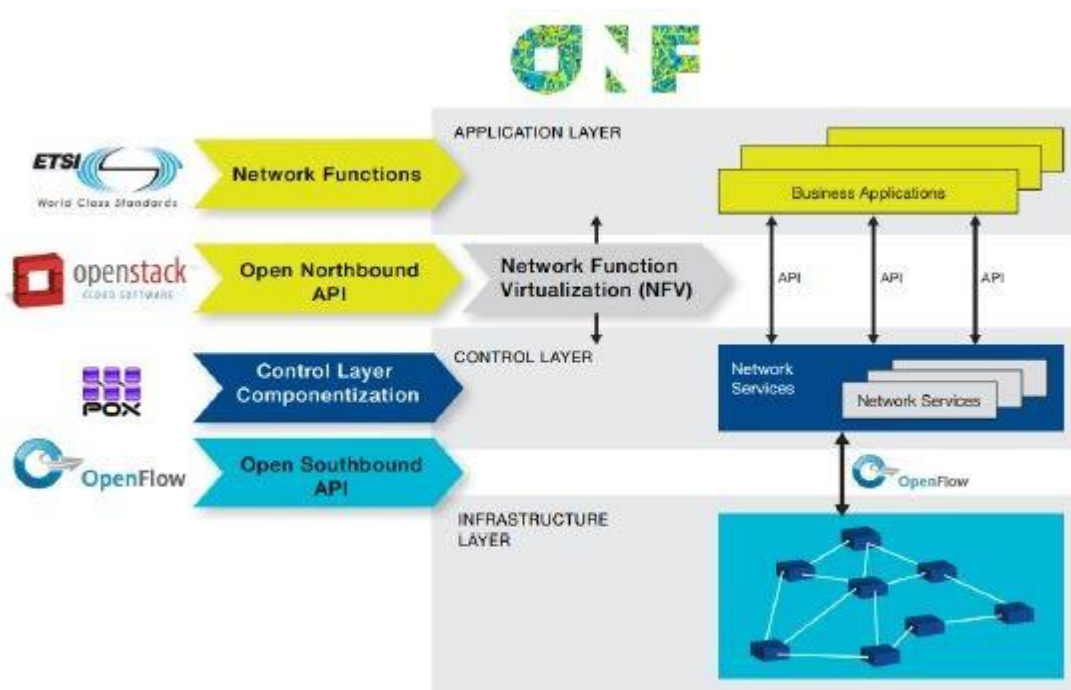


Figure 2.2 The Open SDN architecture.

Figure 2.2 depicts the SDN architecture. As shown in the figure, there are three different layers:

- **Application Layer:** Encompasses solutions that focus on the expansion of network services. These solutions are mainly software applications that communicate with the controller.
- **Control Plane Layer:** Includes a logically-centralized SDN controller, which maintains a global view of the network that takes requests through clearly defined APIs from application layer and performs consolidated management and monitoring of network devices via standard protocols.
- **Infrastructure or Data-plane Layer:** Involves the physical network equipment, including Ethernet switches and routers. Provides programmable and high speed hardware and software, which is compliant with industry standards.

At the bottom layer, the physical network consists of the hardware forwarding devices which store the forwarding information base (FIB) state of the network data plane (e.g., Ternary Content-Addressable Memory TCAM Entries and configured port speeds). Data plane interface or standards-based protocols, typically termed as ‘southbound protocols’, define the control communications between the controller platform and data plane devices such as physical and virtual switches and routers. There are various southbound protocols such as Openflow, Path Computation Element Protocol PCEP, Simple Network Management Protocol SNMP, Open-vSwitch Database Management Protocol OVSDB, etc.

The control plane layer is the core of the SDN, and is realized by the controllers of each domain, which collect the physical network state distributed across every control domain. This component is sometimes called the Network Operating System (NOS), as it enables the SDN to present an abstraction of the physical network state to an instance of the control application (running in the Application Layer), in the form of a global network view.

Northbound Open Application Programming Interface APIs refer to the software interfaces between the software modules of the controller and the SDN applications. These interfaces are published and open to customers, partners, and the open source community for development. The application and orchestration tools may utilize these APIs to interact with the SDN controller.

Application layer covers an array of applications to meet different customer demands such as network automation, flexibility and programmability, etc. Some of the domains of SDN applications include traffic engineering, network virtualization, network monitoring and analysis, network service discovery, access control, etc. The control logic for each application instance may be run as a separate process directly on the controller hardware within each domain.

2.3 SDN controllers

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 such as the choice of programming language which can sometimes affect performance, and the learning curve of the controller. The user base and community support is also a factor. Finally, the choice of controller should consider the Southbound and Northbound interfaces (policy layer) it supports and whether it is used for education, research or production environment.



Figure 2.3 Examples of the most well-known SDN controllers.

There are several popular SDN controllers written in different languages offering a wide variety of services. The most well know controllers are shown in Figure 2.3, summarized in **Table 2.2** and include:

2.3.1 NOX:

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

2.3.2 POX:

Which is based on NOX and is a good choice for rapid development and for developers who want to prototype network controller software. This controller is written in Python. Its high-level SDN API allows developers to quickly turn their ideas into reality and helps them to write their own OpenFlow controller.

2.3.3 Ryu:

Is an open-source Python controller that supports OpenFlow and Openstack. It has relatively poor performance with respect to other commercial grade controllers.

2.3.4 Floodlight:

An Apache-licensed OpenFlow Java-based controller that can be used for controller development in enterprise networks. This controller is a derivative of another Java-based controller named Beacon. Floodlight has been used in a commercial product from Big Switch Networks as a core of their network controller software.

2.3.5 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 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.

Table 2.2 Summary of some properties of some controllers.

	NOX	POX	Ryu	Floodlight	OpenDayLight
Language	C++	Python	Python	Java	Java
Performance	Fast	Slow	Slow	Fast	Fast
Distributed	No	No	Yes	Yes	Yes
OpenFlow version	1.0	1.0	1.0,1.1,1.3, 1.4	1.0	1.0, 1.3
Learning Curve	Moderate	Easy	Moderate	Steep	Steep

2.4 OpenFlow Protocol

In SDN standards, OpenFlow Protocol is defined as one of the communication protocols which enable the SDN/OpenFlow controller to interact with the switching infrastructure from multi-vendors. The network can better adapt to the changing business requirements by adjusting the network using OpenFlow protocol. It is utilized by SDN controller to install flows to the switch/router for implementing network functions such as data forwarding, traffic partitioning, control flows, etc.[6].

2.4.1 OpenFlow Structure

OpenFlow provides the programming interface with better management of the forwarding plane. It interacts with the switching infrastructure using a secure channel to take efficient forwarding decisions. These decisions are given in the form of instructions to the switching infrastructure to add/modify flow tables/entries.

2.4.2 OpenFlow Switch

OpenFlow switch is a switch that can communicate with the SDN controller using OpenFlow protocol. Forwarding in OpenFlow switch is done using Flow tables, Group tables and a Metering tables. Every flow table in a switch is designed to have a set of flow entries where each flow has match fields, counters, and a set of instructions that are applied on matching packets. Figure 2.4 illustrates matching fields in an OpenFlow Packet.

Ingress Port	ETHERNET		IP LAYER				TCP		UDP		
	SA	DA	IPV4		IPV6		Protocol	Source Port	Destination Port	Source Port	Destination Port
	ETHER TYPE		SA	DA	SA	DA					

Figure 2.4 OpenFlow Packet Match Fields.

Figure 2.5 illustrates the OpenFlow Switch. The switch consists of four main modules:

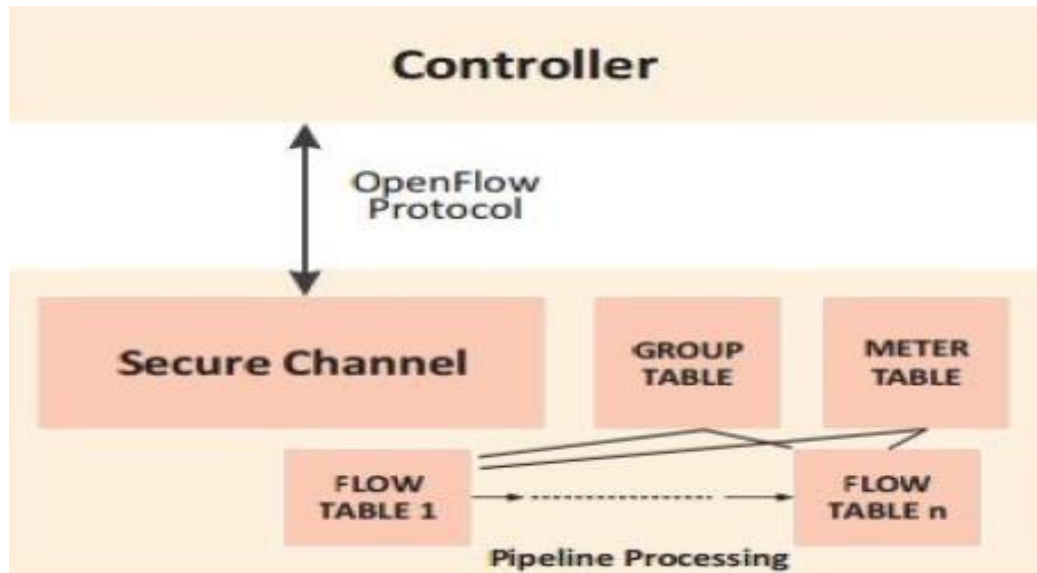


Figure 2.5 OpenFlow Switch Modules.

1. **Secure Channel:** to send commands and packets between the controller and the Open Switch, OpenFlow Channels running over Secure Sockets Layer (SSL) is utilized. Due to this reason, OpenFlow Channel is also referred to as Secure Channel.

2. **Flow Table:** which contains match fields, counters and a set of instructions to apply on the matching flow. To find the corresponding flow rule, tables are looked up from Table 0 to Table N. If no flow rule exists then the packet is sent to the controller for decision on the action.

3. **Group Table:** contains group entries where each entry has a list of actions. These actions are applied to packets which are sent to the group entries.

3. **Meter Table:** consists of meter entries which define per-flow meters. Various simple QoS can be enabled in OpenFlow by utilizing Per-flow meters, for example rate-limiting, and can be combined with per-port queues to implement complex QoS frameworks, such as DiffServ.

Communication between the controller and switch when a new flow arrives is illustrated in Figure 2.6.

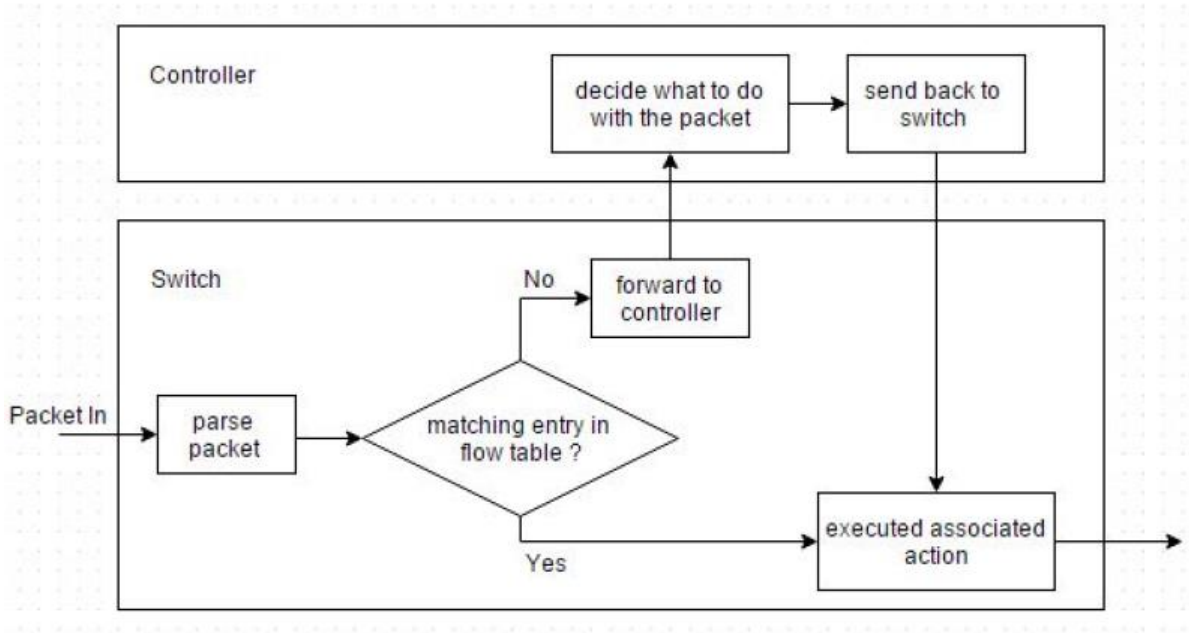


Figure 2.6 OpenFlow Controller and Switch.

2.5 SDN in datacenters

SDN is considered a mature technology in datacenters. A datacenter is a centralized repository, either physical or virtual, and employs many host servers and networking devices that process requests and interconnect to other hosts in the network or to the public network Internet. The requests made to a data center range from web content serving, email, distributed computation to many cloud-based applications. The hierarchical topology of a data center network can be seen in Figure.

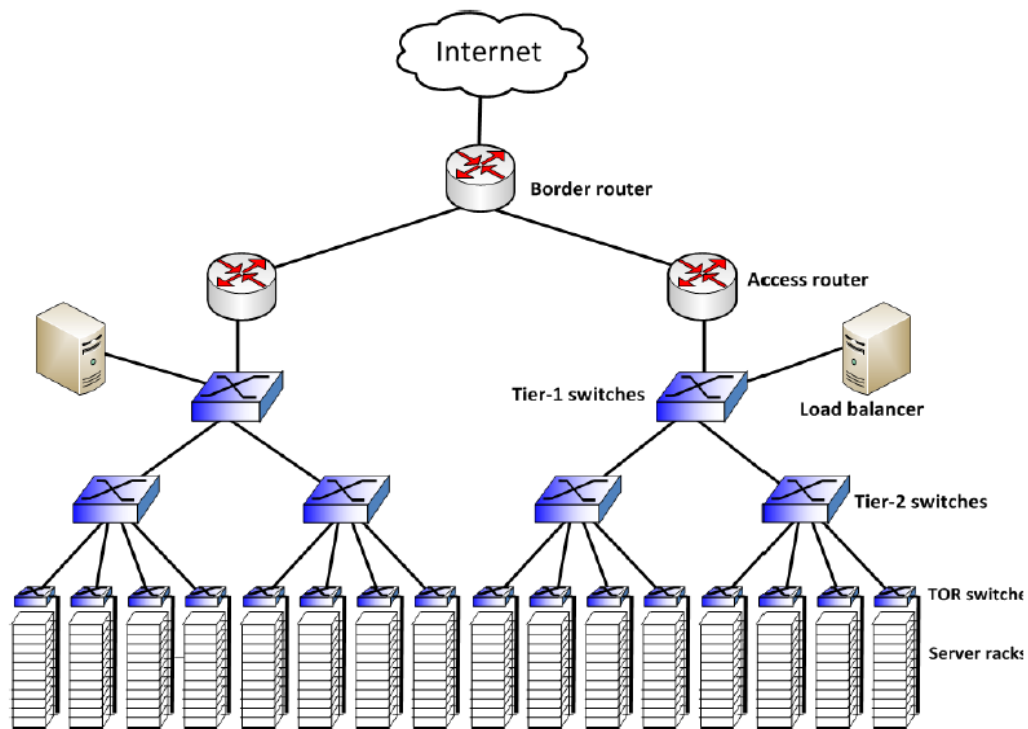


Figure 2.7 The hierarchical topology of a datacenter network.

A datacenter provides many applications simultaneously that are associated with a publicly visible IP address. A load balancer acts as a relay, and performs functions like NAT and firewall preventing direct interactions. So, all external requests are first directed to a load balancer that distributes responses to the associated host server(s), and balances the load across the host servers in the network.

SDN has attracted many datacenter operators. Google has deployed the SDN approach into one of its backbone WANs known as an internal (G-scale) network that carries traffic between datacenters. The deployed SDN network has been in operation at Google, and has offered benefits including higher resource utilization, faster failure handling and faster upgrade functionality. However, its challenges include fault tolerant controllers, flow programming and slicing of network elements for a distributed control. Similarly, Nippon Express Co., Ltd. NEC has also successfully deployed SDN approaches in the datacenter and backbone network at its own software factory, Nippon Express Co., Ltd. As well as Kanazawa University Hospital in Japan [7].

2.6 SDN Migration

Open SDN has been well accepted by the networking industry as the way to transform enterprise, datacenter, 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.

As more network operators adopt open SDN, there is a need for best practices to facilitate the migration of existing networks and services to SDN.

The key steps involved in an SDN migration:

- **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 SDN.
- **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 complete, the target network must be validated against a documented set of requirements or expectations. Network migration steps are shown in Figure 2.8.

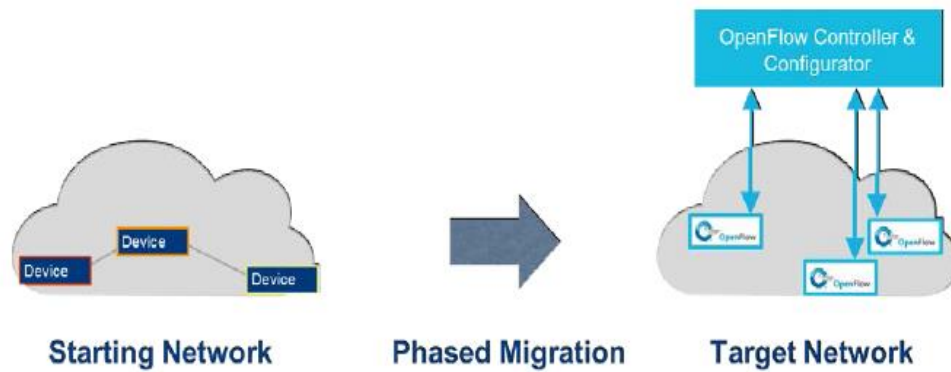


Figure 2.8 SDN Migration steps.

2.6.1 Google legacy-to-hybrid migration

Google’s OpenFlow Wide Area Network WAN is organized as two distinct backbones, one carrying Internet-facing user traffic and another carrying internal traffic between Google’s global datacenters. The breadth of user requirements and the broad scale of the project made Google’s SDN OpenFlow migration a unique, challenging use case demonstrating OpenFlow’s flexibility.

The objective of Google’s WAN migration was to improve scalability, flexibility, and agility in managing the Internet-facing WAN fabric to enhance Google’s user-based services, including Google+, Gmail, YouTube, Google Maps, and others.

Both of Google’s WANs support thousands of individual applications, tremendous traffic volumes, and latency sensitivities—all governed by different overall priorities. Google’s internal network that connects multiple datacenters today is an OpenFlow-based network and a well-known SDN use case. This inter-data-center network was built in a three-layer architecture: switch hardware layer, site controller layer, and global control layer.

Google's SDN migration path moved in stages from a fully distributed monolithic control and data plane hardware architecture to a physically decentralized

(though logically centralized) control plane architecture. The hybrid migration for the Google B4 network proceeded in three general stages:

1. **Starting network** (Figure 2.9). In the initial stage, the network connected datacenters through legacy nodes using External/Internal Border Getaway Protocol (E/IBGP) and Intermediate System Intermediate System (ISIS) routing. Cluster border routers interfaced the datacenters to the network.

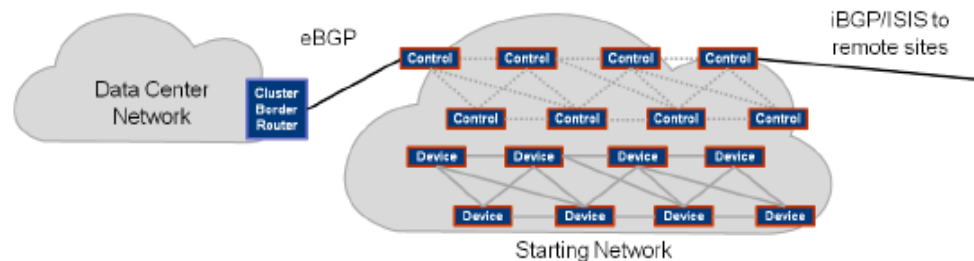


Figure 2.9 Google B4 Starting network.

2. **Phased deployment** (Figure 2.10). In this mixed-network phase, a subset of the nodes in the network were OpenFlow-enabled and controlled by the logically centralized controller utilizing Paxos, an OpenFlow controller, and Quagga an open source routing stack that Google adapted to its requirements.

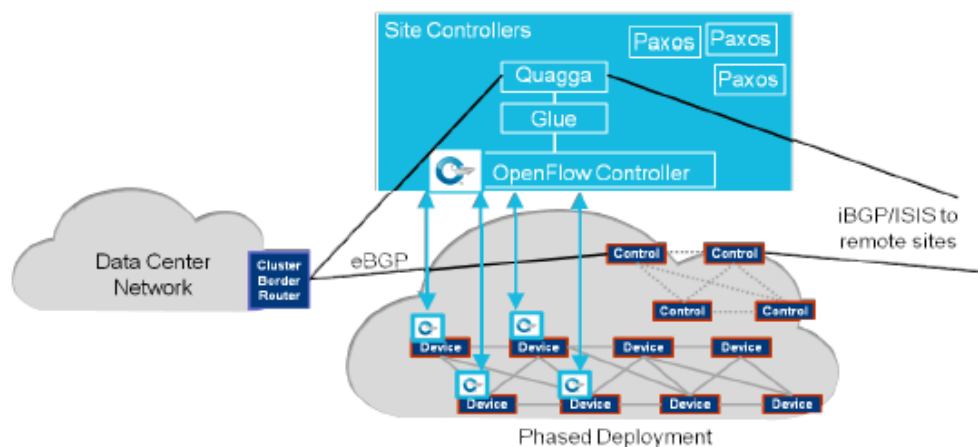


Figure 2.10 Google B4 Phased deployment mixed network.

3. **Target network** (Figure 2.11). In this final phase, all nodes were OpenFlow-enabled. In the target network, the controller controls the entire network. There is no direct correspondence between the datacenter and the network. The controller also has a TE server that guides the traffic engineering in the network.

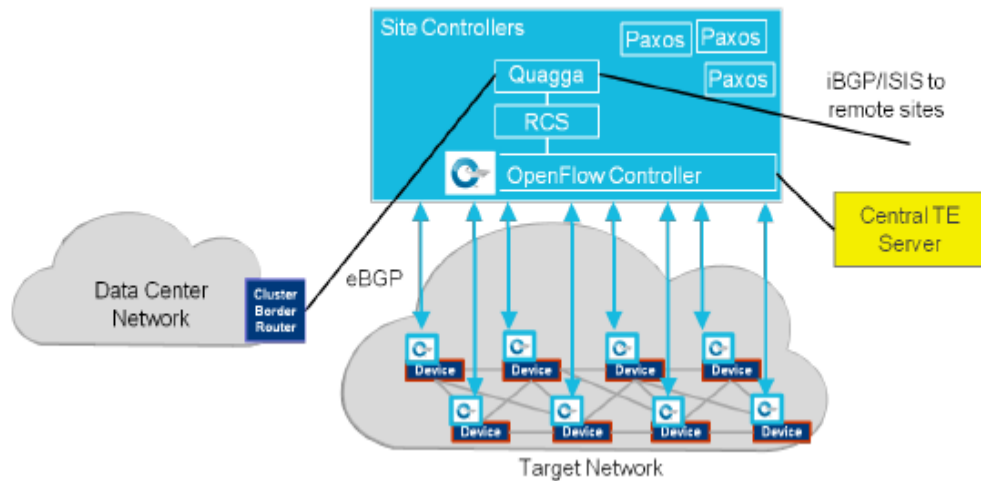


Figure 2.11 Google B4 Target network.

2.7 Network virtualization

Network virtualization is the representation of one or more logical networks on the same infrastructure. There are many different instantiations of network virtualization, some of them emerged since 1990 like VLANs.

Network virtualization provides many benefits such as sharing and thus cost reduction. However this requires resource isolation in CPU, memory, bandwidth, forwarding tables and all other resources. It also provides customizability for different users because they can see their network as a separate logical network and thus being able to customize it in terms of running different routing and forwarding software on that particular slice of the virtual network. The legacy of network virtualization for SDN is quite rich. For example, the notion that a single physical infrastructure can

expose different logical network topologies has its roots in network virtualization. Moreover, the idea of separating service providers from infrastructure providers, which is very common in software defined networks today, has started in network virtualization. In addition, the idea of using multiple controllers to control a single switch and implementing multiple logical switches on a single physical switch has also started with network virtualization.

2.8 Summary

In this chapter, a detailed survey of SDN has been introduced. The next chapter overviews literature review, tools and techniques

Chapter 3 LITERATURE REVIEW, TOOLS AND TECHNIQUES

3.1 Introduction

3.2 Literature Review

3.3 Tools and Techniques

3.1 Introduction

This chapter reviews related works to SDN, It also explains the techniques and software products that are used in this project.

3.2 Literature Review

This section survey previous studies on SDN. A comparison with our proposed system is provided in **Table 3.1**

3.2.1 Event-Driven Network Control Using Software-Defined Networking

This project explores SDN as an emerging paradigm and tests its implementation in dynamic network environments .It also highlights the problem 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. It also covers some of the QoS and load balancing mechanisms and provides a validation for those modules.

Technologies that have been used in this project were Mininet to emulate the network topology and POX as a controller for an OpenFlow protocol, they also used a programming language called Kinetic, which is a Python-embedded domain-specific programming language for writing SDN control programs.

The results for considered parameters concluded that SDN is a very attractive technology for network operators in the market [8].

3.2.2 Software Defined Networking based Data-Center Services

This PhD dissertation deals with the question of how datacenter networks can benefit from the integration of Software-Defined Networking. The thesis covered network primitives, load balancing, QoS overlays and forwarding, and firewalls. It also introduced new and innovative concepts to realize the usual ingress datacenter network service chain.

The scope was to collect isolated performance results and verify if this approach in general can lead to an enhancement of today's predominated data-center network designs. The tools they have used were OFELIA Control Framework and TUB Island.

They evaluated the general opportunity of using SDN technology to deploy these virtual network appliances such as firewall, load balancer, and QoS as network services in data-center networks [9].

3.2.3 Programmable and Scalable Software-Defined Networking Controllers

The work in this PhD dissertation demonstrates that it is feasible to arguably use the simplest possible programming model for centralized SDN policies, in which the programmer specifies the forwarding behavior of a network by defining a packet-processing function as an ordinary algorithm in a general-purpose language.

The contribution of this dissertation was the development of a novel SDN programming model, called algorithmic policies, that provides SDN programmers with a convenient, high-level programming model for expressing network policies. SDN programmers define their network policy by writing a function in a general-purpose language that will be executed on every packet entering the network. This dissertation introduces Maple, an SDN programming framework consisting of a domain specific

language that can be embedded into any programming language with appropriate support.

It demonstrated that Maple can be used to implement several realistic network control algorithms, including L2 shortest path routing, declarative access control policies, traffic monitoring, and policies that include run-time modifiable administrative parameters [10].

- The Limitations in this work are:
 - Maple applies a fully dynamic approach; *i.e.* it does not perform any static analysis or static compilation. Instead, Maple must execute the user's packet processing function on packets to generate flow tables.
 - Any algorithm which changes system state on every packet will prevent Maple from caching any decisions in flow tables.

3.2.4 Implementation of Remote Configuration Using SDN Approach

The purpose of this research was to take advantages of SDN and implement remote configuration of virtual local area network (VLAN) from a single controlling point. The case study that they had adopted is to balance the load between multiple VLANs when one of these VLANs is facing more load than it can handle.

They used GNS3 to emulate the topology and Visual Basic.NET programming language [11].

3.2.5 Comparing a Commercial and an SDN-Based Load Balancer in Campus Network

This project discussed the need for an SDN transport-level load balancing. Their solution over UDP packets. The problem is that the existing load balancer does not divide the load equally among the servers because it uses the source IP address to

divide the load, their solution was to load balance the data coming from the firewall among servers using Software-Defined Networking, They developed three different load balancing policies: Round-Robin, Random and Load-Based.

The tools they used were XenServer for server virtualization and to install and manage all the needed virtual machines (VMs) on the physical machine, Mininet network emulator and Open Virtual Switch (OVS) were used to provide switching for hardware virtualizing platforms in a multilayer virtual switch. They also used Splunk and Syslog to monitor and analyze machine-generated big data through its web interface

The results of these experiments suggest that these SDN load balancing policies are reliable enough to be used in a real production network with a high input data rate. Their work proves that the SDN-based load balancer is competitive with the commercial load balancer. Replacing the software OpenFlow switch with a hardware switch is likely to further improve the results [12].

3.2.6 Implementation of SDN in a Campus NAC Use Case

This project introduces and demonstrates how to implement software defined networking concepts within a campus network taking Sudan University of Science and Technology network as a case study. They use Mininet emulation environment, and OpenDayLight as a controller to control this environment.

The work was to set and run a few network access control polices (NAC) in different locations in 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, and FTP) [13].

Table 3.1 Comparison of previous work and proposed system.

Related Study	Emulator	Tools	Use case	Results
SDN for Datacenter	Mininet	POX Controller	SUST datacenter network	Implement virtual network appliances including Hub, Switch and Load Balancer.
SDN in NAC use case	Mininet	OpenDayLight controller	SUST campus network	Set and Run a few network access control polices.
Comparing a commercial SDN load balancer	Mininet	1-XenServer for server virtualization 2- Splunk and Syslog for monitor	Campus network	SDN load balancing policies are reliable enough to be used in a real production network.
Implementation of Remote Configuration Using SDN Approach	GNS3	—	Remote configuration of virtual local area network	Adopted is to balance the load between multiple VLANs when one of these VLANs is facing more load than it can handle.
Software Defined Networking based Data-Center Services	—	OFELIA Control Framework and TUB Island.	Datacenter network	Deploy virtual network appliances such as firewall, load balancer, and QoS.
Event-Driven Network Control Using Software-Defined Networking	Mininet	POX controller	Dynamic Network and Fat tree topology	Explores some QoS and load balancing mechanisms and provides a validation for those modules.

3.3 Tools and Techniques

3.3.1 Oracle VM VirtualBox

VirtualBox is a cross-platform virtualization application. For one thing, it is installed 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. VirtualBox is deceptively simple yet very powerful. It can run everywhere from small embedded systems or desktop class machines all the way up to datacenter deployments and even Cloud environments.

3.3.2 Linux Operating System

Linux is an open source operating system (OS) for personal computers, servers and many other hardware platforms that is based on the UNIX operating system. Linux was originally created by Linus Torvalds as a free alternative operating system to more expensive UNIX systems. Linux has grown since its creation due in part to its open source roots. Open source software is freely licensed and users may copy and even change the code. It is popular among technology savvy computer users because of its efficiency and reliability. It is cross-platform compatible, working on desktops, laptops, servers, mobile devices, and even game consoles, tablets and supercomputers. Because of this, the Linux OS is found in a great variety of hardware platforms. However, very few desktop and laptop PCs are installed with Linux as an operating system. The most popular operating systems for personal computers are Microsoft Windows and Apple's iOS.

There are several operating systems that use the Linux kernel. These include Ubuntu, Debian, RedHat, and Fedora.

3.3.3 Mininet

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run on standard Linux network software and can be used as a flexible network testbed for developing OpenFlow applications. 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. Mininet uses process-based virtualization to run many (up to 4096) hosts and switches on a single Operating System kernel. Mininet can create kernel or user-space OpenFlow switches, controllers to control the switches, and hosts to communicate over the simulated network. Mininet connects switches and hosts using virtual ethernet (veth) pairs. It can be downloaded as a VM which comes along with an Ubuntu 13.04 image. The major Mininet advantage is that it has developed its own controller, but it also allows the emulated network. One feature of Mininet is Miniedit which allows editing and configuring the topology through graphical user interface (GUI).

3.3.4 POX Controller

POX is a networking software platform written in Python. It started life as an OpenFlow controller, but can now also function as an OpenFlow switch, and can be useful for writing networking software in general. POX currently communicates with OpenFlow 1.0 switches and includes special support for the Open vSwitch/Nicira extensions. Pox.py boots up POX. It takes a list of module names on the command line, locates the modules, calls their launch () function (if it exists), and then transitions to the "up" state.

3.3.5 Python

Python is an interpreter, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

3.3.6 Wireshark Network Analyzer

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

3.3.7 USB to Ethernet Adapter



Figure 3.1 USB to Ethernet adapter.

Because the laptop that contains the network does not have integrated Ethernet ports, as an alternative, we turned to USB Ethernet adapters. These network adapters connect an Ethernet port via the device's USB port to connect more than one device to the network.

3.4 Summary

This chapter has explained the literature reviews, tools and techniques used in the project. The following chapter illustrates the design and methodology of the system.

Chapter 4 DESIGN AND METHODOLOGY

- 4.1** Introduction
- 4.2** The Network Topology
- 4.3** Operating System
- 4.4** The Emulator
- 4.5** The Controller
- 4.6** Load Balancing
- 4.7** Software Configuration

4.1 Introduction

This chapter demonstrates how SDN is implemented using Mininet network emulator and POX controller. It describes the network topology, operating system, load balancing and software configuration.

4.2 The Network Topology

Sudan University of Science and Technology (SUST) datacenter network topology was selected to be implemented in this project. A clearance was needed in order to collect information about the datacenter. Once this clearance was acquired, several visits have been scheduled to the university's main network department and datacenter. The result was acquiring SUST's main datacenter network topology, as shown in Figure 4.1.

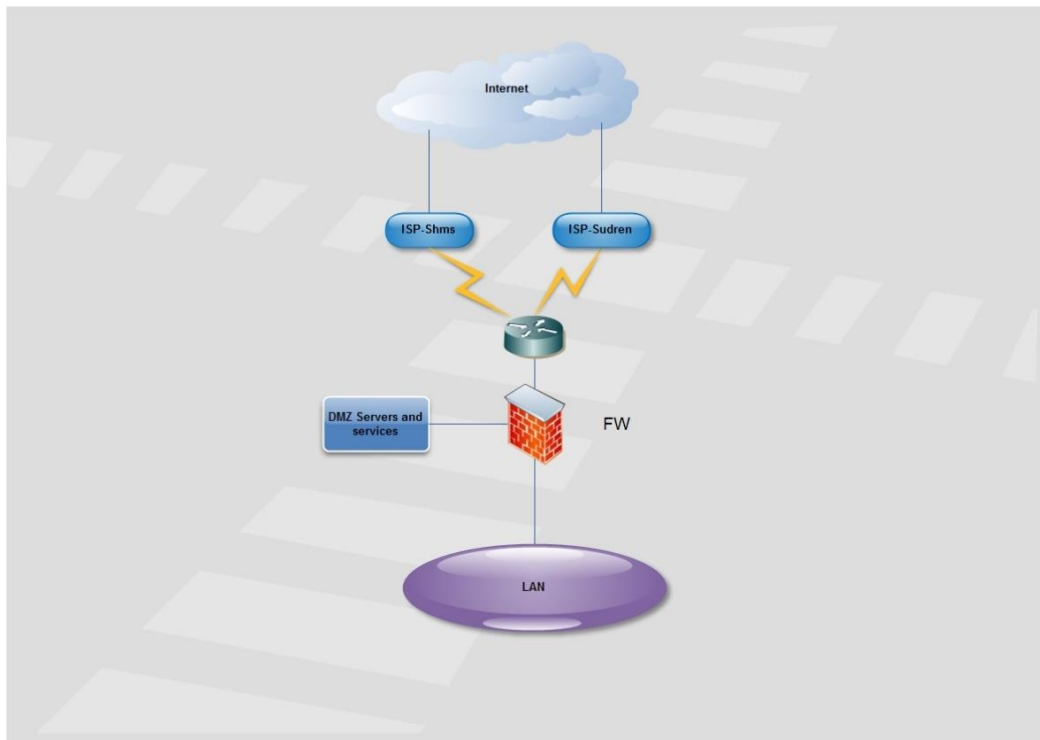


Figure 4.1 SUST Datacenter Network Topology.

The university network contains one main core switch connecting the LAN network with the Demilitarize Zone (DMZ) area. Inside the DMZ there is one server with multiple services provided to the network as shown in Figure 4.2.

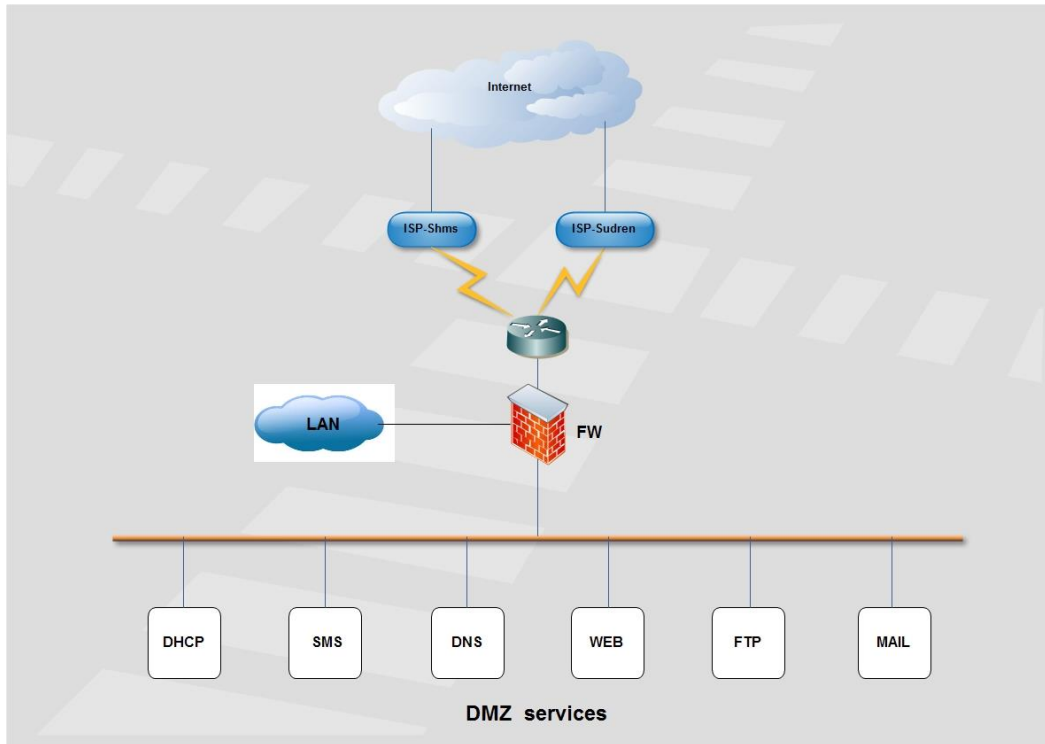


Figure 4.2 Services inside the DMZ.

This topology was edited in order to give a clear implementation of SDN concepts and to meet load balancing main expectations which is distributing the traffic to more than one server. The edited topology contains three basic legs: The DMZ area, LAN, and the main switch.

4.3 Operating System

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

4.4 The Emulator

Mininet was chosen for this project because it combines many of the best features of emulators, hardware testbeds, and simulators. Compared to full system virtualization based approaches, Mininet:

- Boots faster: seconds instead of minutes
- Scales larger: hundreds of hosts and switches vs. single digits
- Provides more bandwidth: typically 2Gbps total bandwidth on modest hardware
- Installed easily: a prepackaged Virtual Machine (VM) is available that runs on VMware or VirtualBox for Mac/Win/Linux with OpenFlow v1.0 tools already installed.

Compared to hardware testbeds, Mininet:

- Is inexpensive and always available
- Is quickly reconfigurable and restartable

Compared to simulators, Mininet:

- Runs real, unmodified code including application code, OS kernel code, and control plane code (both OpenFlow controller code and Open vSwitch code)
- Easily connects to real networks
- Offers interactive interface.

4.5 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, and 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 POX, as the SDN controller in this project.

In our topology we have one switch to direct the traffic to the servers. So the controller will manage the switch to work as desired. Three scenarios will be implemented and tested on the Open vSwitch in our network.

1. Hub:

A hub is a common connection point for devices in a network, it contains multiple ports. When a packet arrives at one port, it is copied to the other ports so that all segments of the LAN can see all packets.

2. Switch:

A switch manages the flow of data across a network by transmitting a received network packet only to the one or more devices for which the packet is intended. Each network device connected to a switch can be identified by its network address, allowing the switch to regulate the flow of traffic

3. Load Balancer:

A load balancer is a device that acts as a reverse proxy and distributes network or application traffic across a number of servers. Load balancers are used to increase capacity (concurrent users) and reliability of applications. The following section provides more information on load balancers.

4.6 Load Balancing

Load balancing is an important networking feature generally performed by multiple dedicated hardware devices known as Application Delivery Controllers (ADCs). These load balancing devices distribute large amounts of inbound traffic among a group of back-end servers hosting the same application content.

A load balancer device sits between the client and the server farm accepting incoming network and application traffic and distributing the traffic across multiple backend servers using various methods.

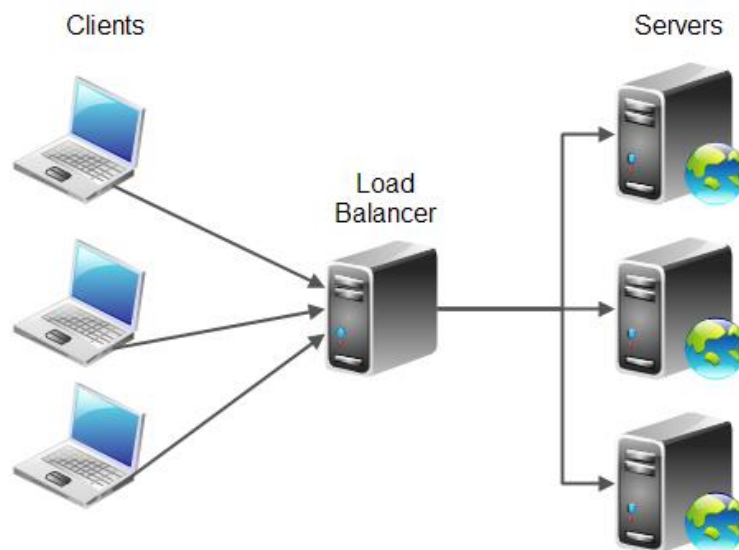


Figure 4.3 Load Balancer.

By balancing application requests across multiple servers, a load balancer reduces individual server load and prevents any one application server from becoming a single point of failure, thus improving overall application availability and responsiveness. When one application server becomes unavailable, the load balancer directs all new application requests to other available servers in the pool.

4.6.1 Load balancing algorithms and methods

Load balancing uses various algorithms, called load balancing methods, to define the criteria that the appliance uses to select the service to which to redirect each client request. Different load balancing algorithms use different criteria. The most famous load balancing algorithms are:

1. The Least Connection Method
2. The Round Robin Method
3. Weighted Round Robin
4. Fastest Response

In this project, we implement the round robin algorithm, the following subsection briefly explains it.

4.6.2 Round-Robin algorithm:

Round-Robin one of the simplest methods for distributing client requests across a group of servers. Going down the list of services in the group, the round-robin load balancer forwards a client request to each server in turn. When it reaches the end of the list, the load balancer loops back and goes down the list again (sends the next request to the first listed server, the one after that to the second server, and so on).

The main benefit of round-robin load balancer is that it is extremely simple to implement. The algorithm flowchart of the Round-Robin algorithm is shown in Figure 4.4.

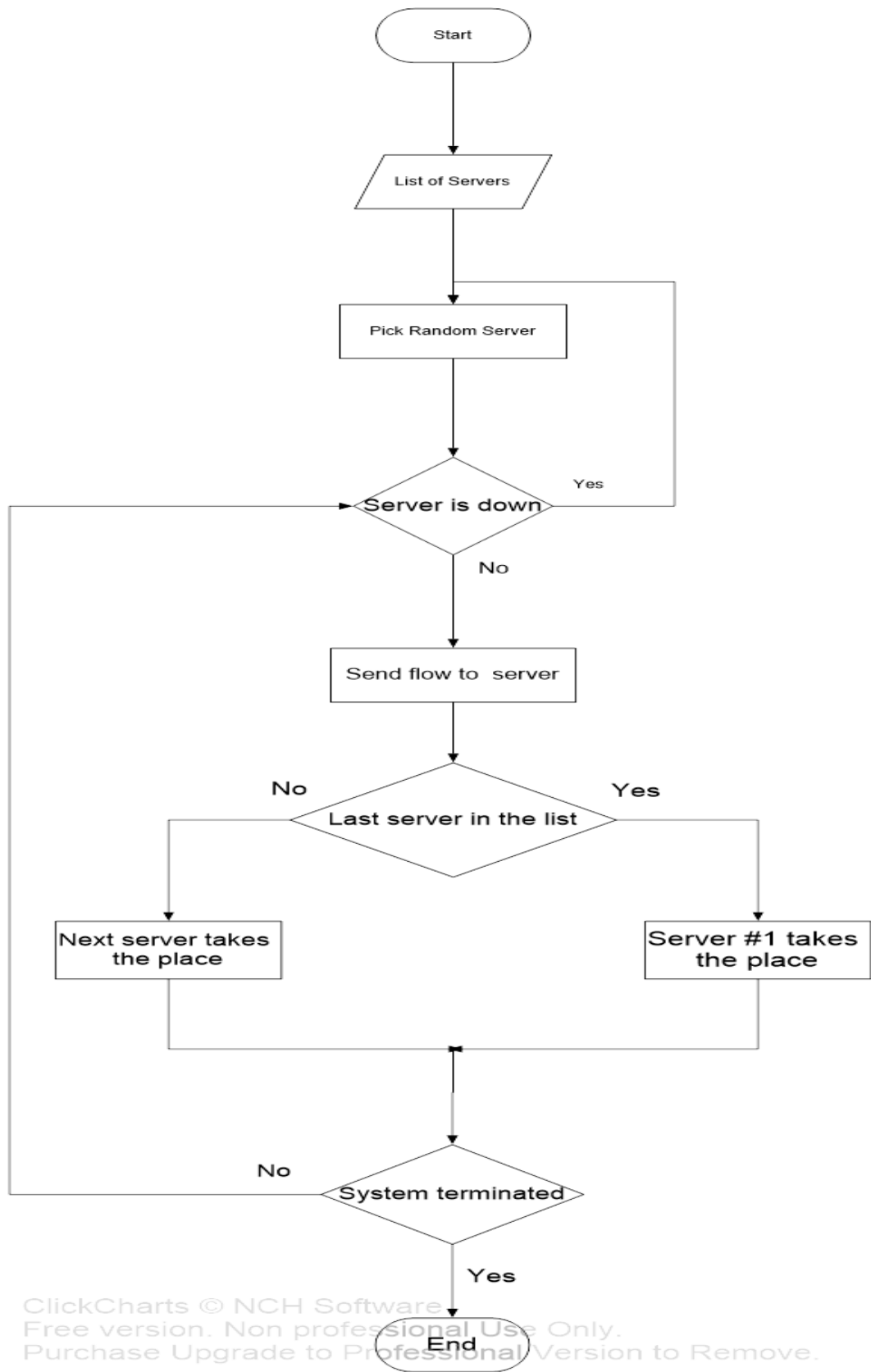
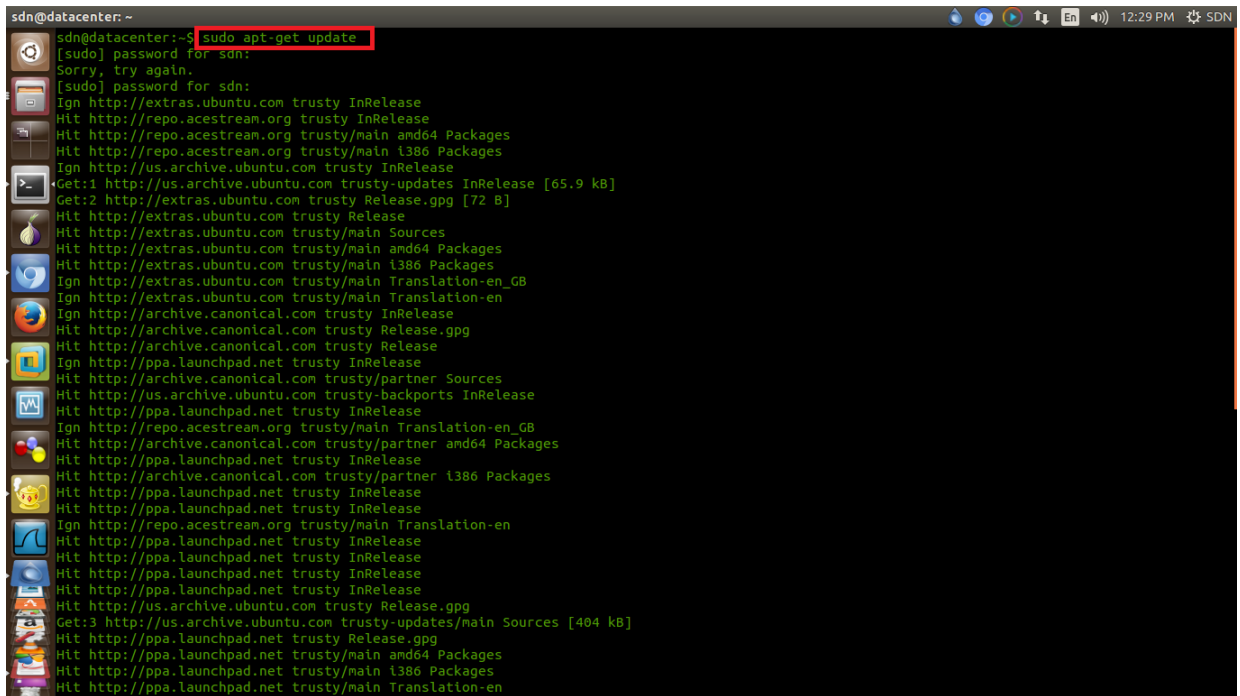


Figure 4.4 Round-Robin Algorithm flowchart.

4.7 Software Configuration

In this part all steps to build and integrate the system are explained.

4.7.1 Mininet Installation



```
sdn@datacenter: ~  
sdn@datacenter:~$ sudo apt-get update  
[sudo] password for sdn:  
Sorry, try again.  
[sudo] password for sdn:  
Ign http://extras.ubuntu.com trusty InRelease  
Hit http://repo.acestream.org trusty InRelease  
Hit http://repo.acestream.org trusty/main amd64 Packages  
Hit http://repo.acestream.org trusty/main i386 Packages  
Ign http://us.archive.ubuntu.com trusty InRelease  
Get:1 http://us.archive.ubuntu.com trusty-updates InRelease [65.9 kB]  
Get:2 http://extras.ubuntu.com trusty Release.gpg [72 B]  
Hit http://extras.ubuntu.com trusty Release  
Hit http://extras.ubuntu.com trusty/main Sources  
Hit http://extras.ubuntu.com trusty/main amd64 Packages  
Hit http://extras.ubuntu.com trusty/main i386 Packages  
Ign http://extras.ubuntu.com trusty/main Translation-en_GB  
Ign http://extras.ubuntu.com trusty/main Translation-en  
Ign http://archive.canonical.com trusty InRelease  
Hit http://archive.canonical.com trusty Release.gpg  
Hit http://archive.canonical.com trusty Release  
Ign http://ppa.launchpad.net trusty InRelease  
Hit http://archive.canonical.com trusty/partner Sources  
Hit http://us.archive.ubuntu.com trusty-backports InRelease  
Hit http://ppa.launchpad.net trusty InRelease  
Ign http://repo.acestream.org trusty/main Translation-en_GB  
Hit http://archive.canonical.com trusty/partner amd64 Packages  
Hit http://ppa.launchpad.net trusty InRelease  
Hit http://archive.canonical.com trusty/partner i386 Packages  
Hit http://ppa.launchpad.net trusty InRelease  
Hit http://ppa.launchpad.net trusty InRelease  
Ign http://repo.acestream.org trusty/main Translation-en  
Hit http://ppa.launchpad.net trusty InRelease  
Hit http://ppa.launchpad.net trusty InRelease  
Hit http://ppa.launchpad.net trusty InRelease  
Hit http://ppa.launchpad.net trusty InRelease  
Hit http://ppa.launchpad.net trusty InRelease  
Hit http://us.archive.ubuntu.com trusty Release.gpg  
Get:3 http://us.archive.ubuntu.com trusty-updates/main Sources [404 kB]  
Hit http://ppa.launchpad.net trusty Release.gpg  
Hit http://ppa.launchpad.net trusty/main amd64 Packages  
Hit http://ppa.launchpad.net trusty/main i386 Packages  
Hit http://ppa.launchpad.net trusty/main Translation-en
```

Figure 4.5 Updating packages on Ubuntu.

The highlighted command in Figure downloads the package lists from the “repositories” and updates them to get information on the newest versions of packages and their dependencies. After this command downloads all the dependences, the command shown in Figure 4.6 is executed:

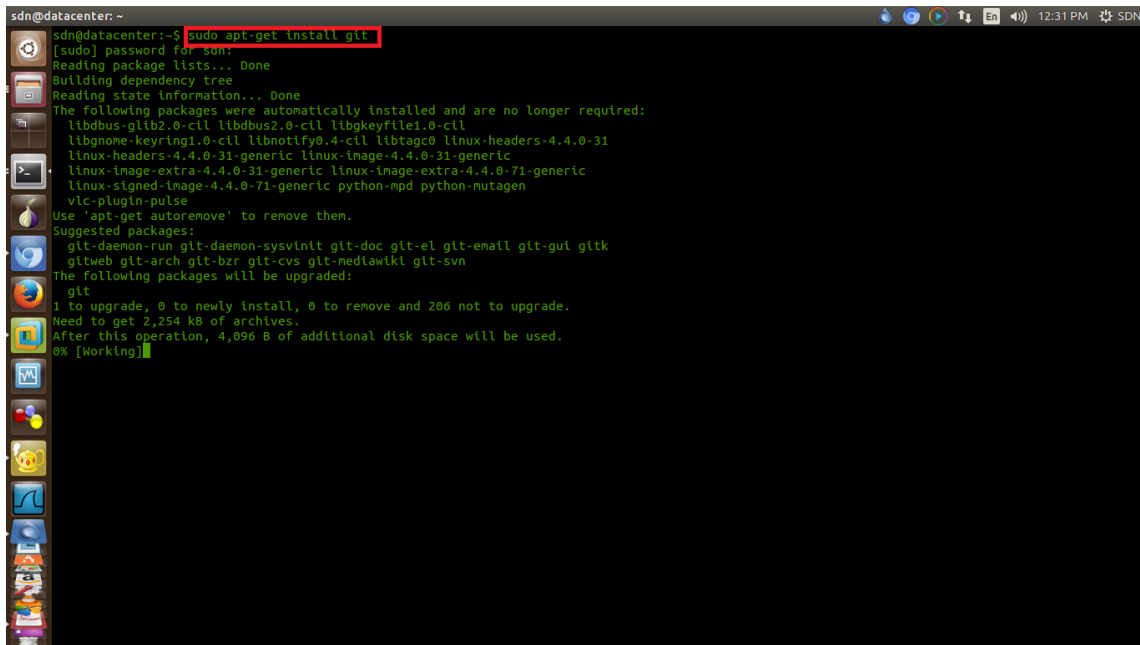


Figure 4.6 Installing git.

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 Mininet emulator.

```
#git clone http://github.com/mininet/mininet
```

After this command all files of Mininet are stored in /home/mininet. Once the files of Mininet are cloned, we execute the installation command is executed as shown in Figure.

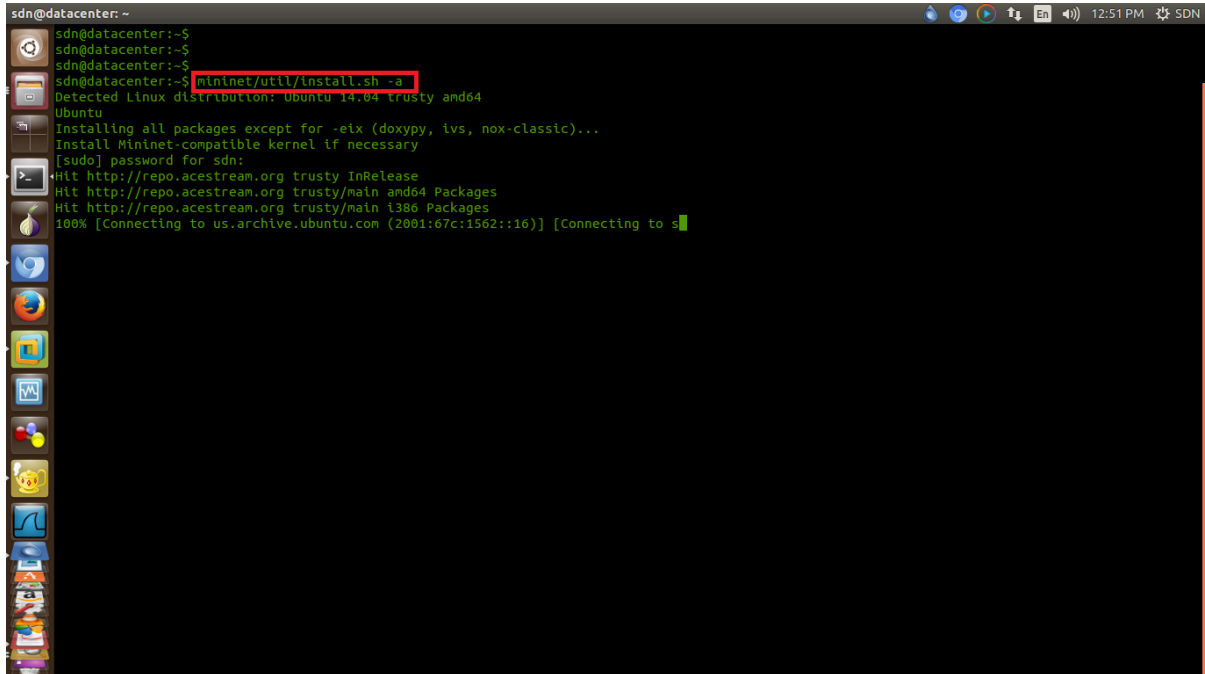


Figure 4.7 Installing mininet.

At this stage, the topologies are ready to be imported.

The SDN topology implemented using Miniedit is shown in Figure 4.8.

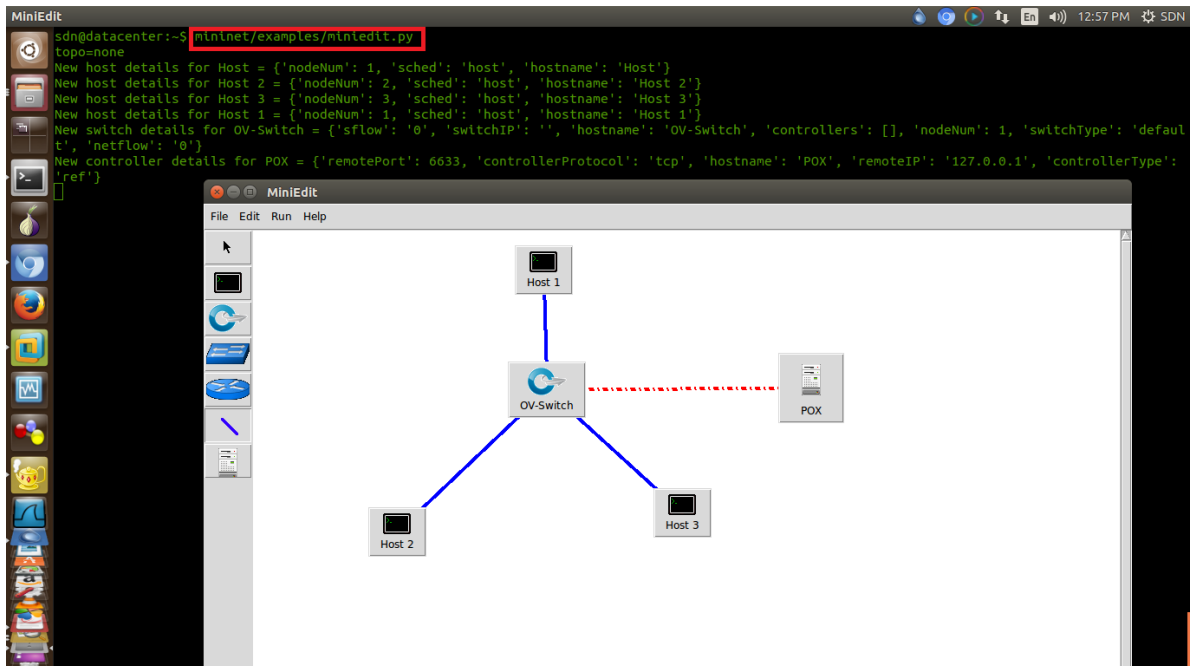


Figure 4.8 Miniedit.

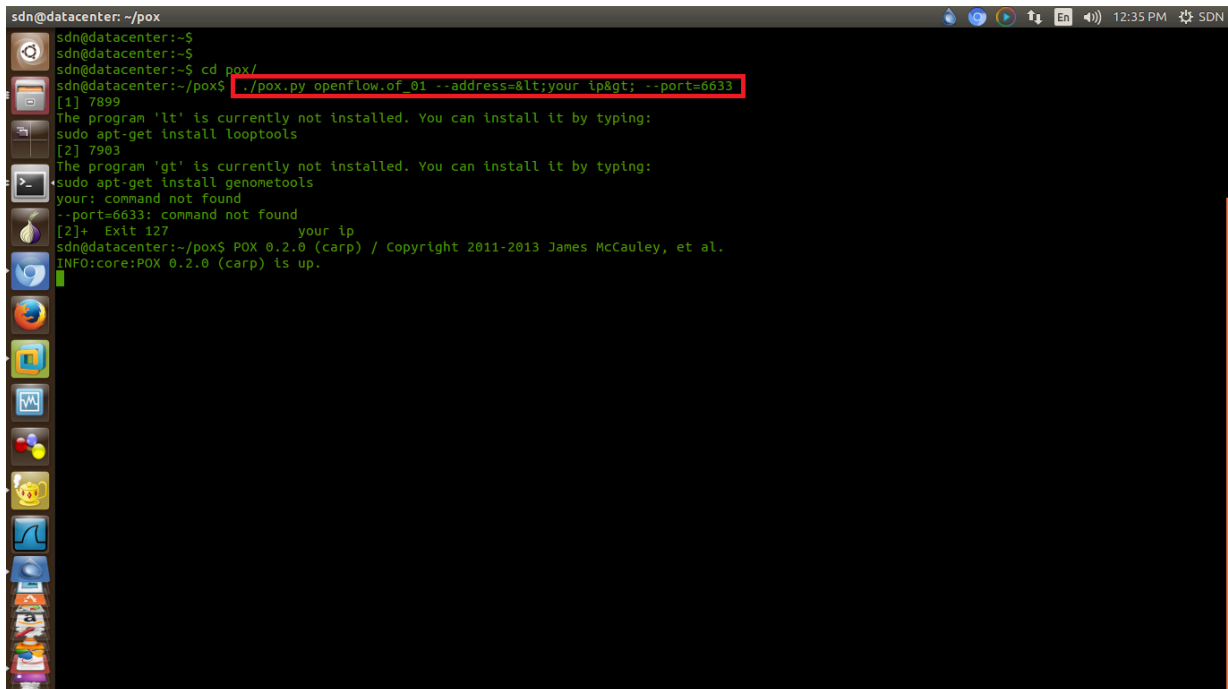
4.7.2 POX Controller Installation

POX is a lightweight OpenFlow controller that is written completely in Python and is targeted for developers to spin up their own controllers. Python is installed by default in Ubuntu and when executing “sudo apt-get update” it will get the newest version of python.

POX is also found in git repositories and is implemented using the following command:

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

After this all configuration files of POX will be found in /home/POX and we can test to find out if it is working or not by typing the command shown in Figure 4.9.



```
sdn@datacenter: ~/pox
sdn@datacenter:~$
sdn@datacenter:~$
sdn@datacenter:~$ cd pox/
sdn@datacenter:~/pox$ ./pox.py openflow.of_01 --address=<your ip> --port=6633
[1] 7899
The program 'lt' is currently not installed. You can install it by typing:
sudo apt-get install looptools
[2] 7903
The program 'gt' is currently not installed. You can install it by typing:
sudo apt-get install genometools
your: command not found
--port=6633: command not found
[2]+  Exit 127      your ip
sdn@datacenter:~/pox$ POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
```

Figure 4.9 Running POX controller.

4.7.3 The Servers

Ubuntu OS was chosen to be the server and it had two services installed, HTTP and FTP services.

The HTTP services run on apache server and it contains the webpage of SUST as a default webpage (<http://www.sust.com>) as shown in Figure. The website design and configuration is taken as an example from the original sustech.edu page. So it is for demonstration purposes only.

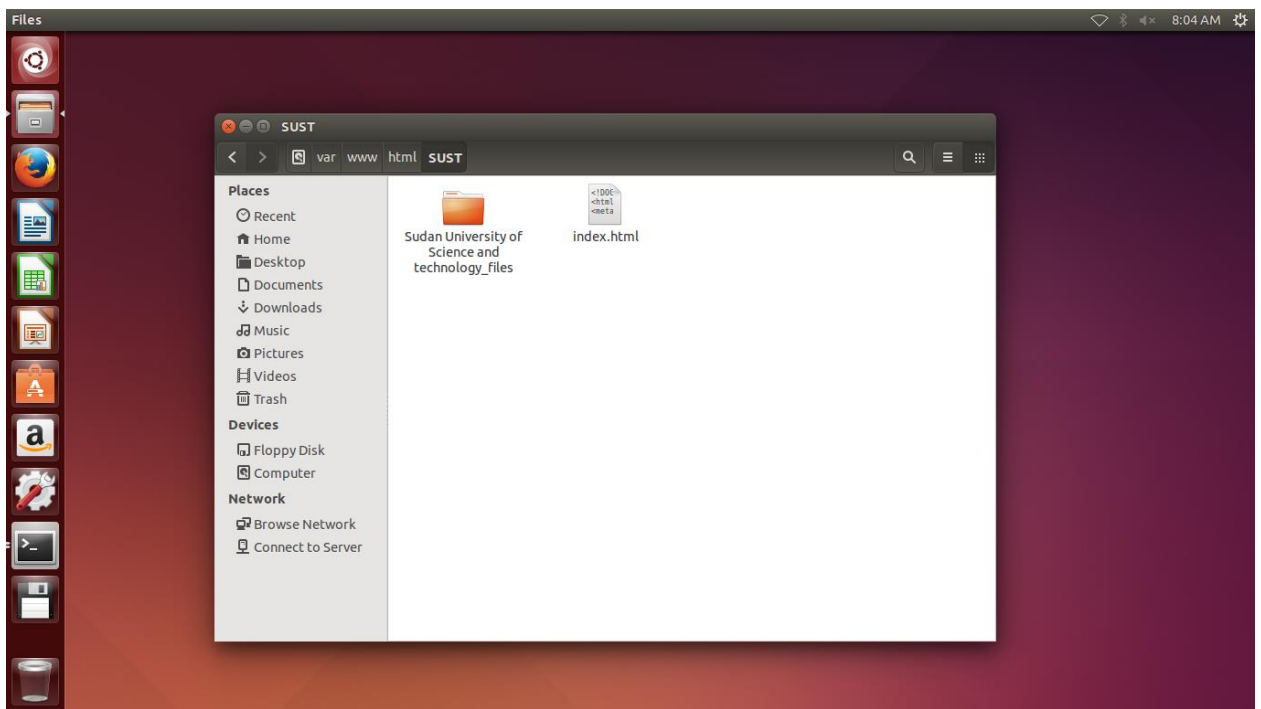


Figure 4.10 Document root for SUST's page.

As for the FTP service, a default FTP site was made including several folders. Configuration of the FTP role included configuring the IP address of the server and selecting the default FTP.

4.8 Summary

This chapter has explained the network topology, operating system, emulator, controller, load balancing and software configuration used in the design of the network. The next chapter demonstrates the implementation and testing of the SDN.

Chapter 5 Implementation and Results

5.1 Introduction

5.2 Implementation

5.3 Simulation Results

5.4 External Devices Results

5.1 Introduction

This chapter demonstrates the implementation, testing and verification of the SDN network. It shows the hub, switch and load balancer scenarios implemented using the mininet simulator and external devices. A HTTP and FTP connectivity test is also verified.

5.2 Implementation

The edited Topology of SUST's datacenter is implemented in mininet containing two clients and three servers in the DMZ area as shown in Figure 5.1.

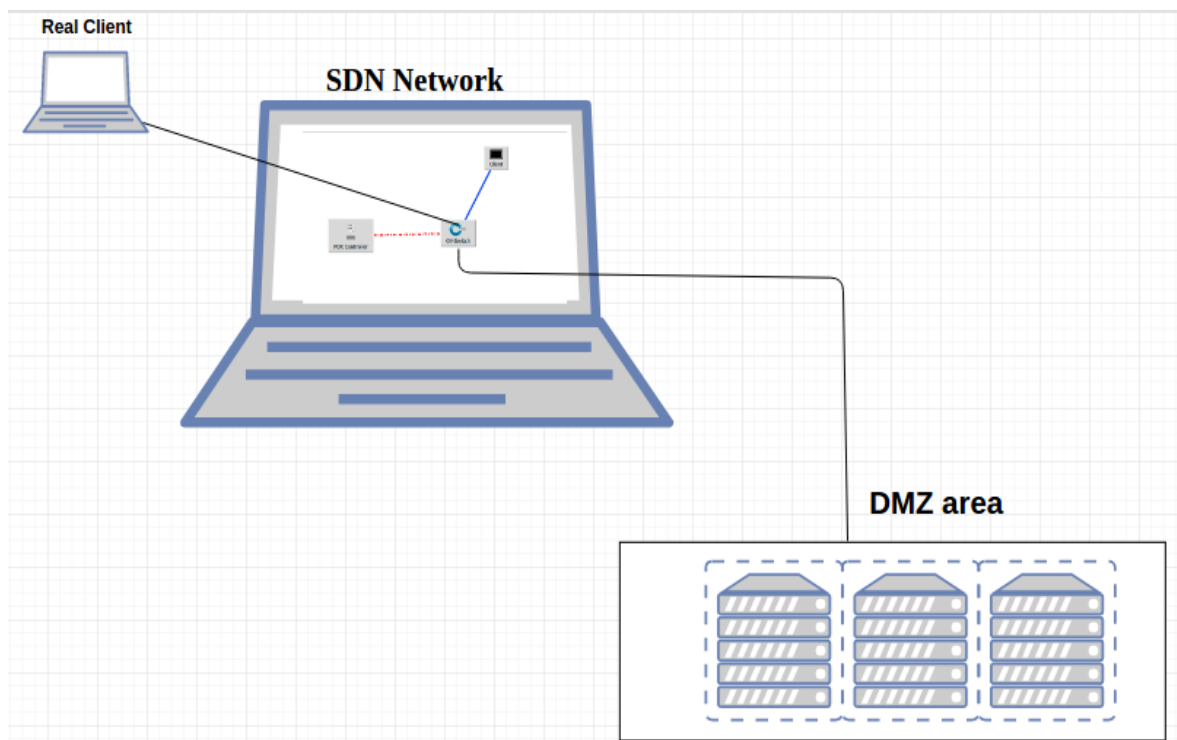


Figure 5.1 Network topology with external devices.

The servers and the physical client are connected to the main laptop that contains the network using an external interface.

POX controller is managed by the application window shown in Figure 5.2 to perform three different scenarios on the Open-vSwitch, the hub, switch and load balancer scenarios.

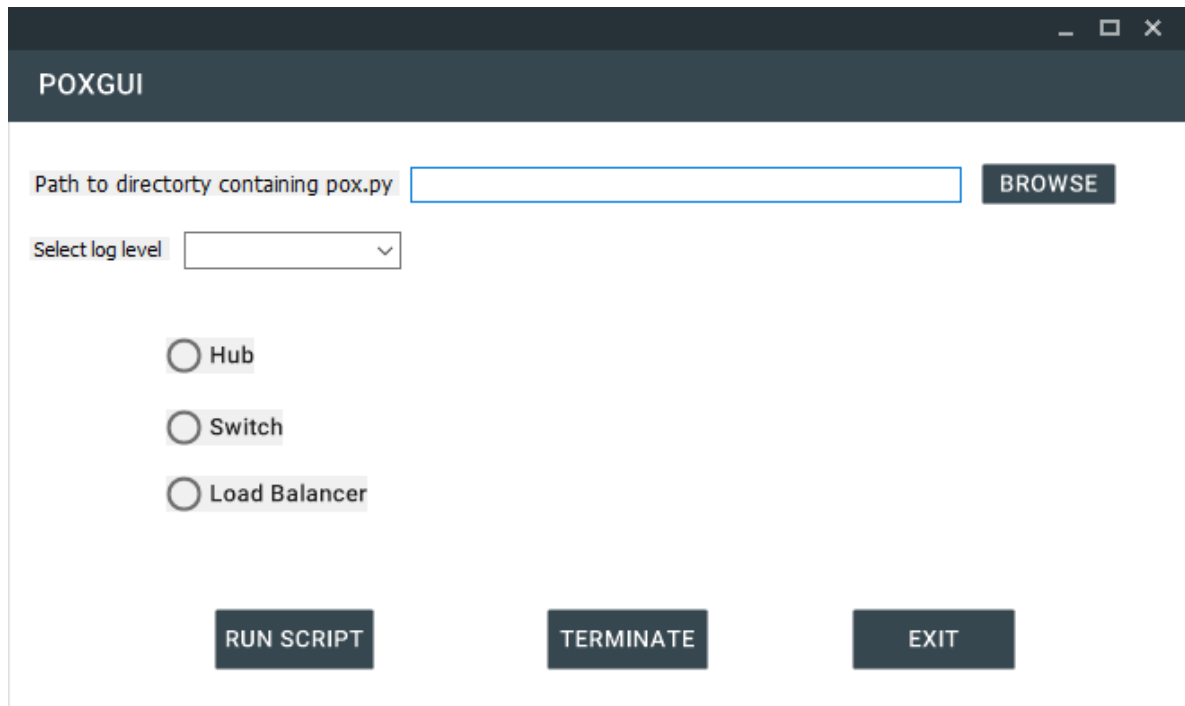


Figure 5.2 POX Controller GUI.

5.3 Simulation Results

This section demonstrates the results of the hub, switch and load balancer scenarios, obtained from the Mininet simulation.

5.3.1 The Hub Scenario

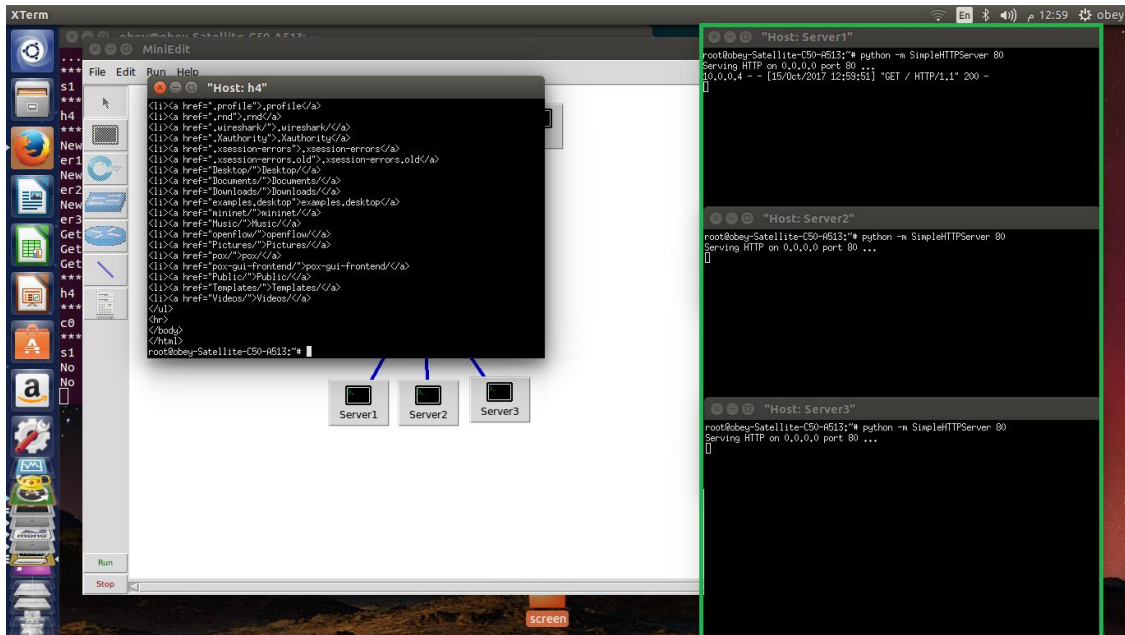


Figure 5.3 Requesting the service in hub scenarios.

The box in Figure 5.3 contains the three servers that will serve the network and h4 is the client who requests the service from server 1.

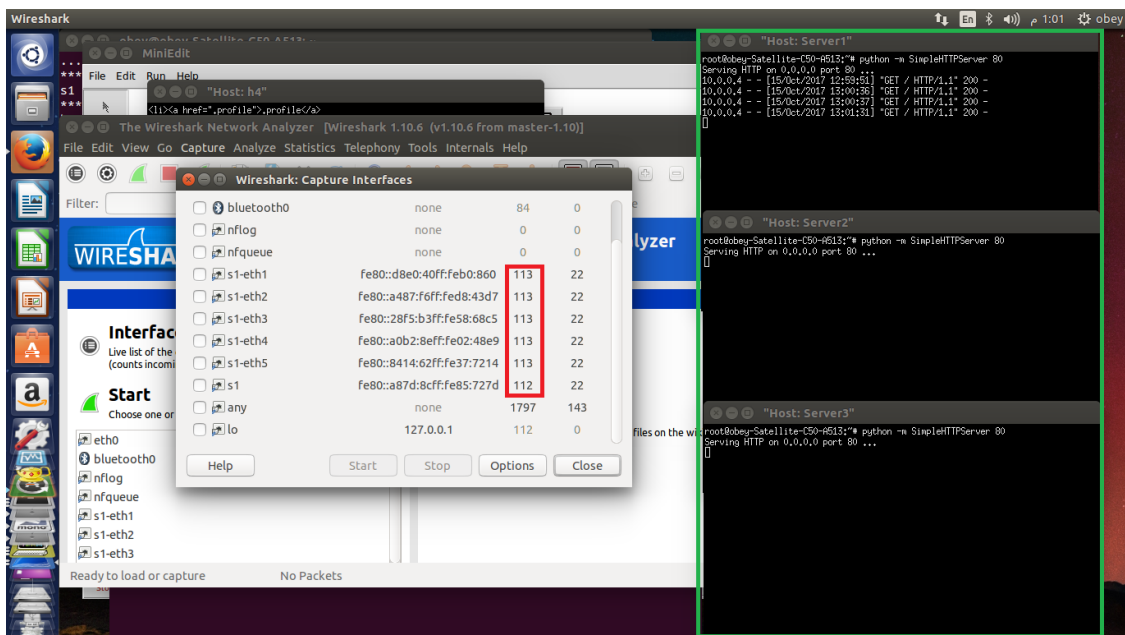


Figure 5.4 Broadcasting of packets in the hub scenarios.

Figure 5.4 show the results obtained from Wireshark packet analyzer, and as shown, the request from h4 to server 1 is broadcasted to all nodes in the datacenter.

5.3.2 The Switch Scenario

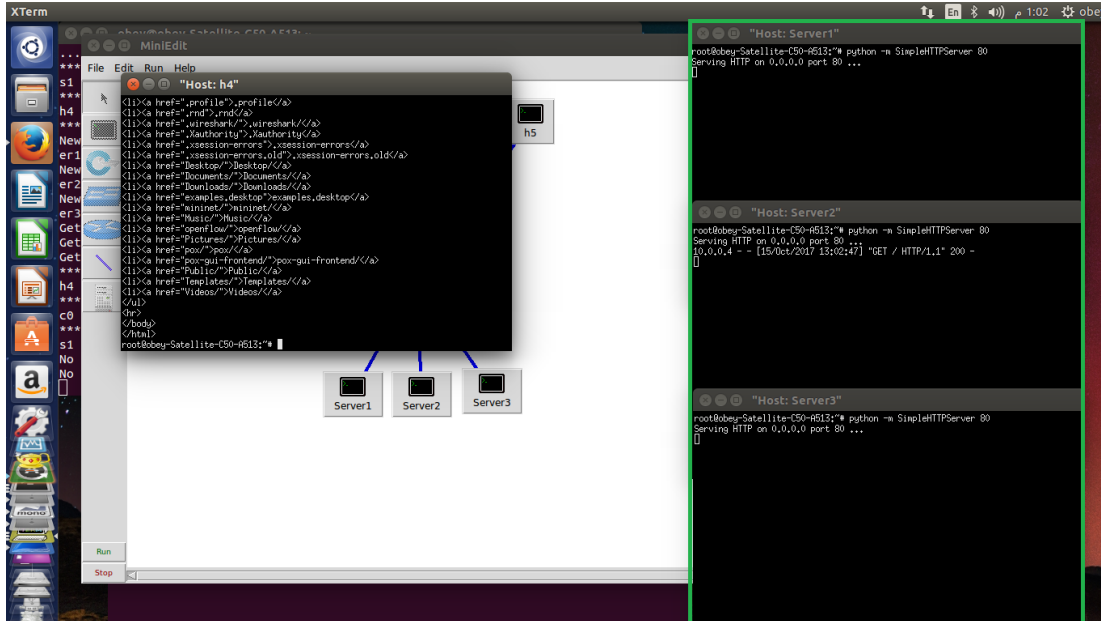


Figure 5.5 Requesting the service in the switch scenarios.

In the switch scenario the request is made from h4 to server 2 as shown in Figure 5.5.

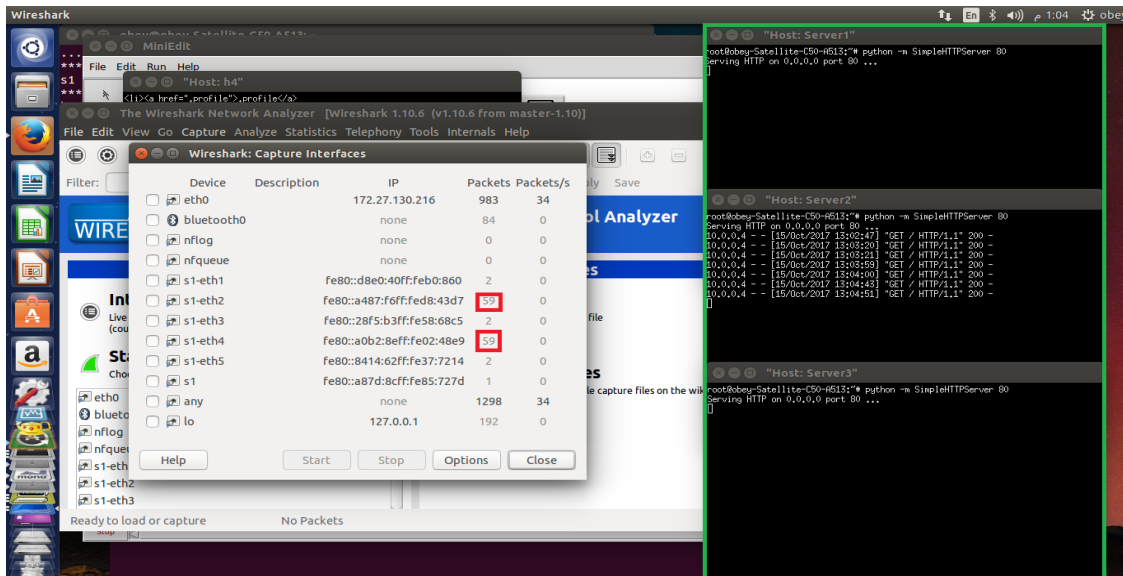


Figure 5.6 Exchange of packets in the switch scenario.

Figure 5.6 shown the results obtained from Wireshark, and as we can see the packets are exchanged between the source h4 and the destination server 2 only.

5.3.3 Load Balancer Scenario

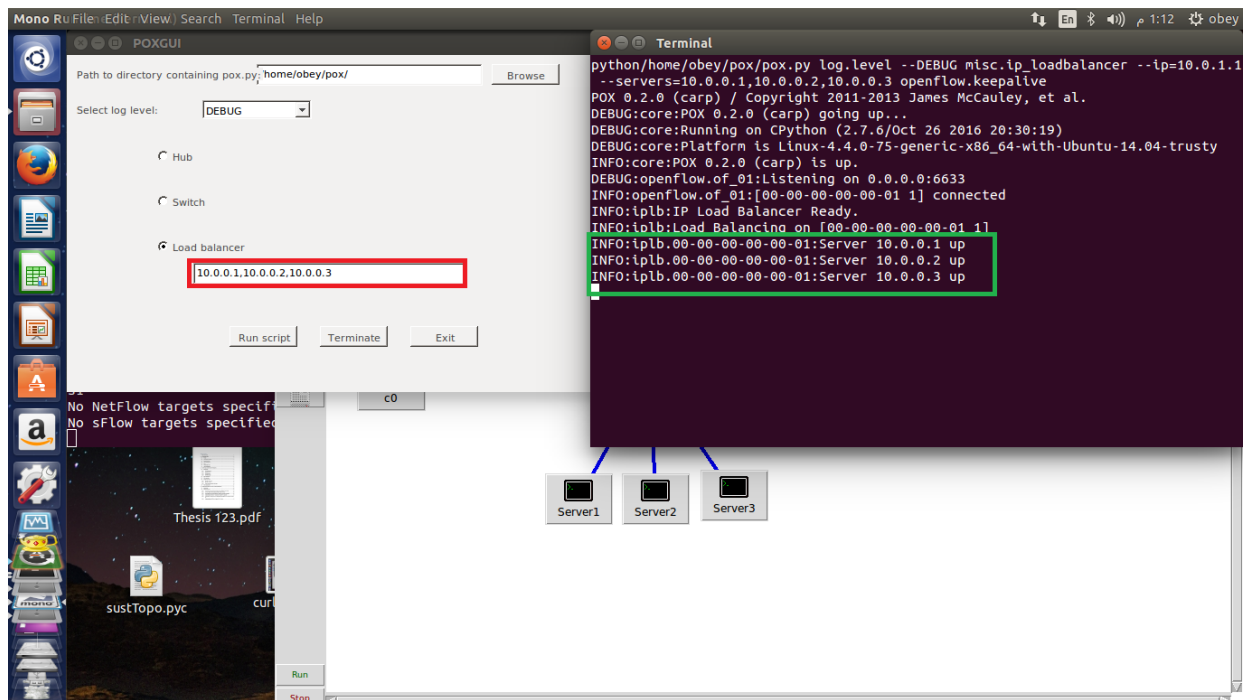


Figure 5.7 Run Load Balancer.

In the load balancer scenario the operator of the network will pass a list of IP servers, and then a debug message notifies if all servers are up or if there is a problem.

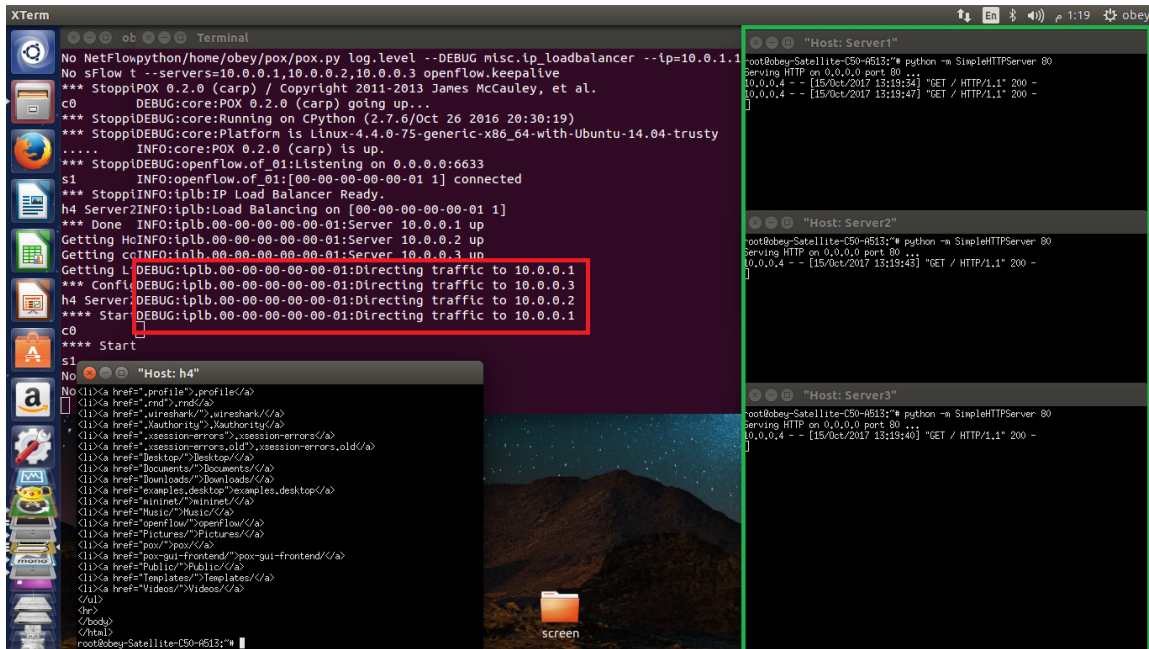


Figure 5.8 Distributing of traffic among servers.

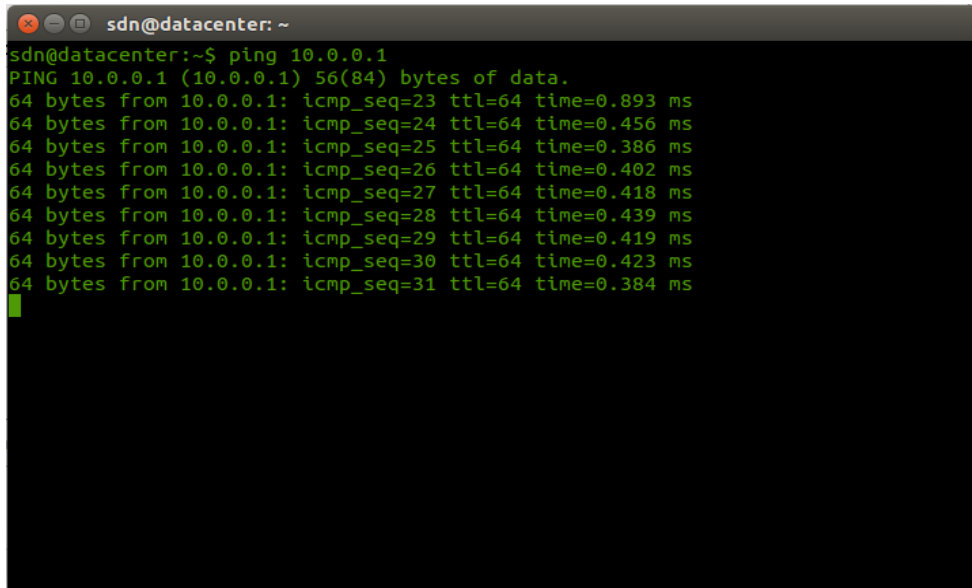
When h4 requests the service in the load balancer scenario, the traffic is randomly directed to one of the servers. If there is another request from the same client or another client, the traffic is directed to the next randomly selected server in the list, as shown in Figure 5.8.

5.4 External Devices Results

In addition to implementing the SDN in a simulation environment, we connect the network to external devices and test the connectivity of the implementation. Three actual servers are connected to the emulator through external interfaces, and the connectivity of HTTP and FTP services are verified using Wireshark.

5.4.1 Hub Connectivity

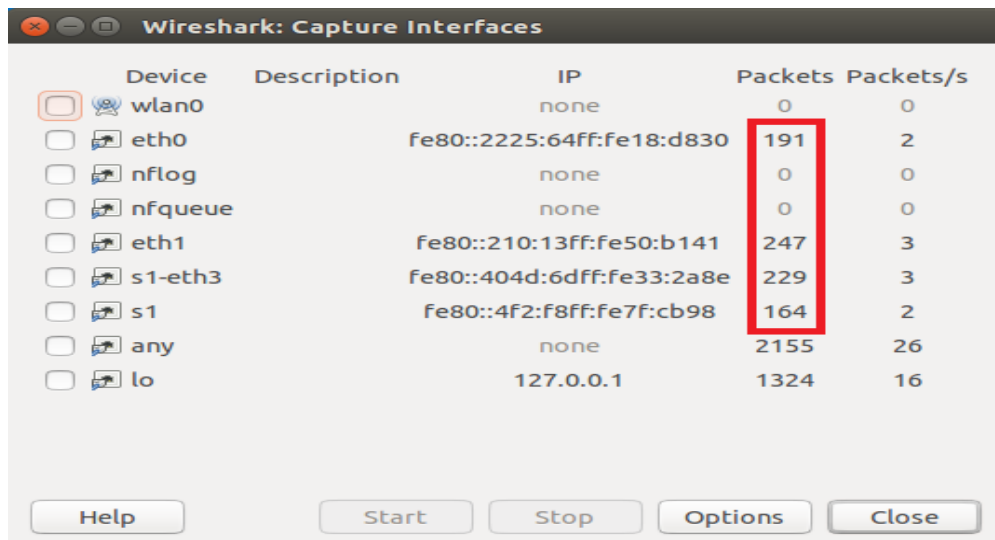
To verify the connectivity of the topology using a Programmable Hub, a successful Internal Control Message Protocol (ICMP) echo and echo reply messages (ICMP ping) is sent between the nodes as shown in Figure 5.9.



```
sdn@datacenter: ~  
sdn@datacenter:~$ ping 10.0.0.1  
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.  
64 bytes from 10.0.0.1: icmp_seq=23 ttl=64 time=0.893 ms  
64 bytes from 10.0.0.1: icmp_seq=24 ttl=64 time=0.456 ms  
64 bytes from 10.0.0.1: icmp_seq=25 ttl=64 time=0.386 ms  
64 bytes from 10.0.0.1: icmp_seq=26 ttl=64 time=0.402 ms  
64 bytes from 10.0.0.1: icmp_seq=27 ttl=64 time=0.418 ms  
64 bytes from 10.0.0.1: icmp_seq=28 ttl=64 time=0.439 ms  
64 bytes from 10.0.0.1: icmp_seq=29 ttl=64 time=0.419 ms  
64 bytes from 10.0.0.1: icmp_seq=30 ttl=64 time=0.423 ms  
64 bytes from 10.0.0.1: icmp_seq=31 ttl=64 time=0.384 ms
```

Figure 5.9 ICMP ping by the Hub.

To verify the hub mechanism is working, we can see in Wireshark that there is a broadcast of ICMP packets inside the topology as shown in Figure 5.10.

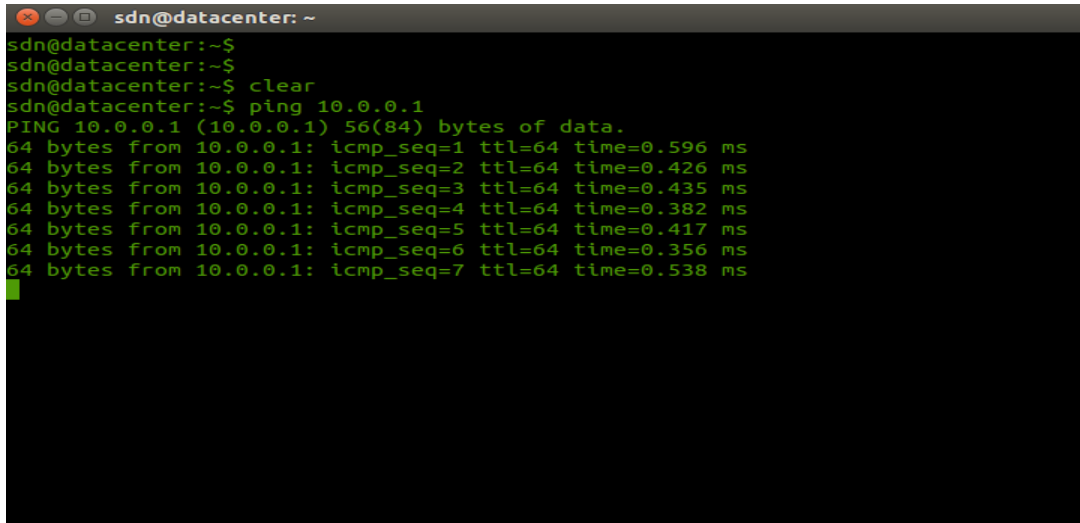


Device	Description	IP	Packets	Packets/s
<input checked="" type="checkbox"/> wlan0		none	0	0
<input type="checkbox"/> eth0	fe80::2225:64ff:fe18:d830		191	2
<input type="checkbox"/> nflog		none	0	0
<input type="checkbox"/> nfqueue		none	0	0
<input type="checkbox"/> eth1	fe80::210:13ff:fe50:b141		247	3
<input type="checkbox"/> s1-eth3	fe80::404d:6dff:fe33:2a8e		229	3
<input type="checkbox"/> s1	fe80::4f2:f8ff:fe7f:cb98		164	2
<input type="checkbox"/> any		none	2155	26
<input type="checkbox"/> lo		127.0.0.1	1324	16

Figure 5.10 Broadcasting of packets in the hub scenario.

5.4.2 Switch Connectivity

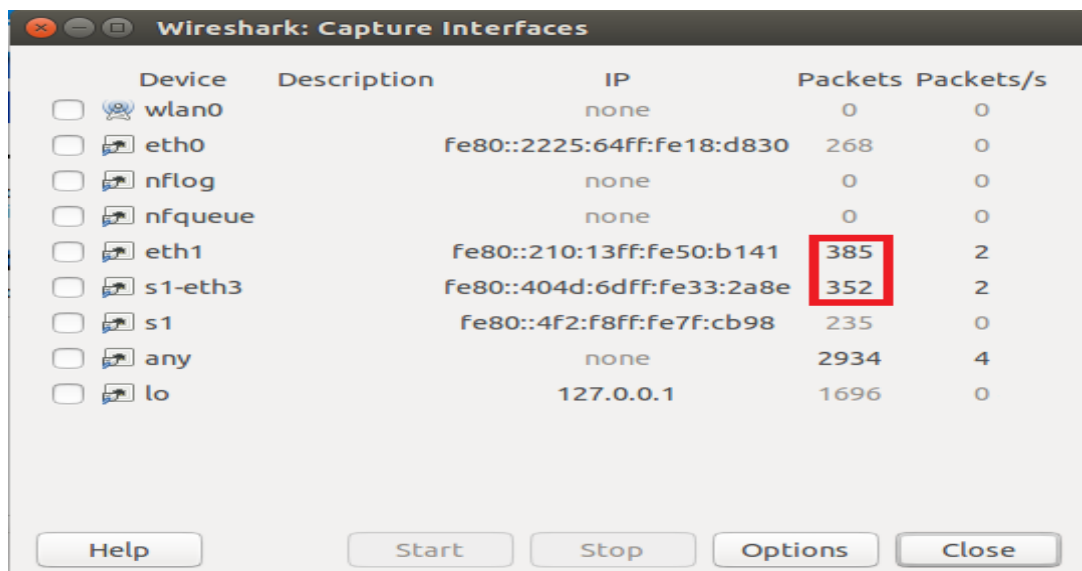
To verify that the connectivity of the topology using a Programmable Switch, a successful ICMP echo and echo reply messages (ICMP ping) is sent between the nodes as shown in Figure 5.11.



```
sdn@datacenter: ~
sdn@datacenter:~$
sdn@datacenter:~$ clear
sdn@datacenter:~$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data:
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.596 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.426 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.435 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.382 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.417 ms
64 bytes from 10.0.0.1: icmp_seq=6 ttl=64 time=0.356 ms
64 bytes from 10.0.0.1: icmp_seq=7 ttl=64 time=0.538 ms
```

Figure 5.11 ICMP ping by switch.

As can be seen in Figure 5.12 only the source and the destination of the ICMP request exchange packets between each other. This validates the success of the switch mechanism.



Device	Description	IP	Packets	Packets/s
<input type="checkbox"/> wlan0		none	0	0
<input type="checkbox"/> eth0		fe80::2225:64ff:fe18:d830	268	0
<input type="checkbox"/> nflog		none	0	0
<input type="checkbox"/> nfqueue		none	0	0
<input type="checkbox"/> eth1		fe80::210:13ff:fe50:b141	385	2
<input type="checkbox"/> s1-eth3		fe80::404d:6dff:fe33:2a8e	352	2
<input type="checkbox"/> s1		fe80::4f2:f8ff:fe7f:cb98	235	0
<input type="checkbox"/> any		none	2934	4
<input type="checkbox"/> lo		127.0.0.1	1696	0

Figure 5.12 Source and destination exchange packets in the switch scenarios.

5.4.3 HTTP Connectivity

Simple test is made from the physical host to the server, by opening a web browser in the host and typing <http://www.sust.com>.

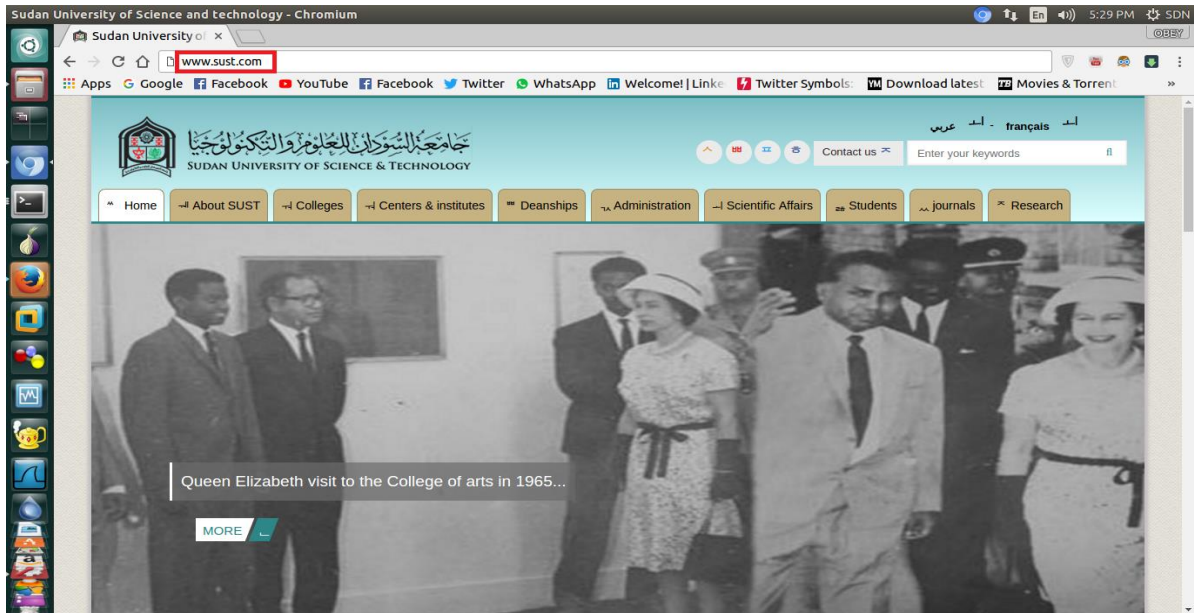


Figure 5.13 HTTP request.

The HTTP protocol can also be verified from Wireshark as in Figure 5.14.

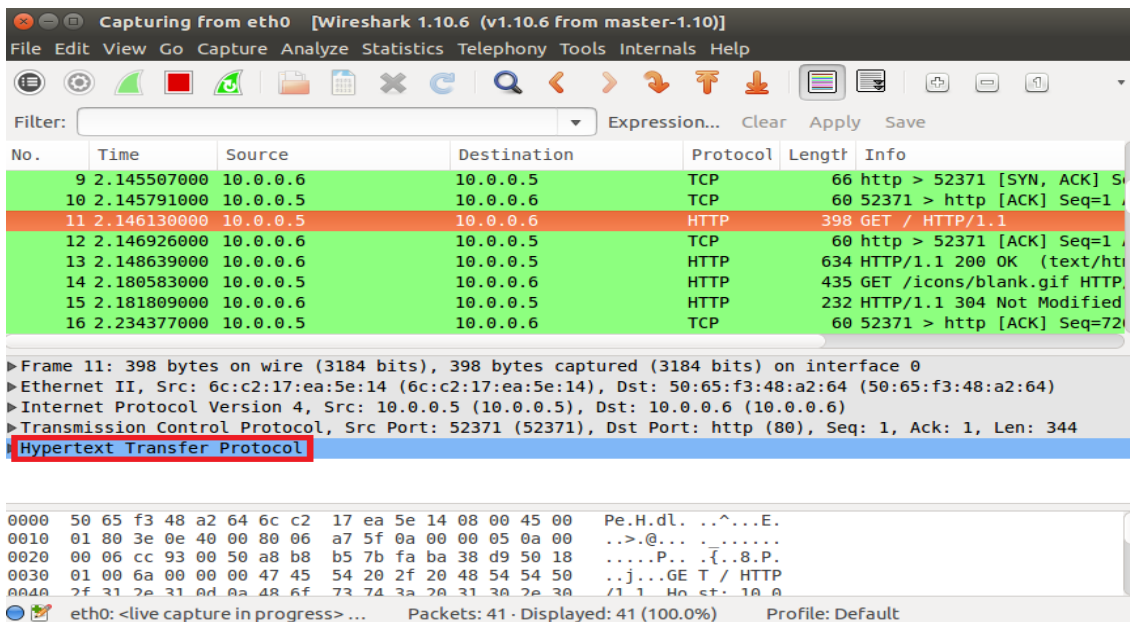


Figure 5.14 Exchange of HTTP packets.

5.4.4 FTP Connectivity

The File Transfer Protocol (FTP) can also be verified from the web browser of the physical host, Figure 5.15 shows a successful FTP access from host to server.

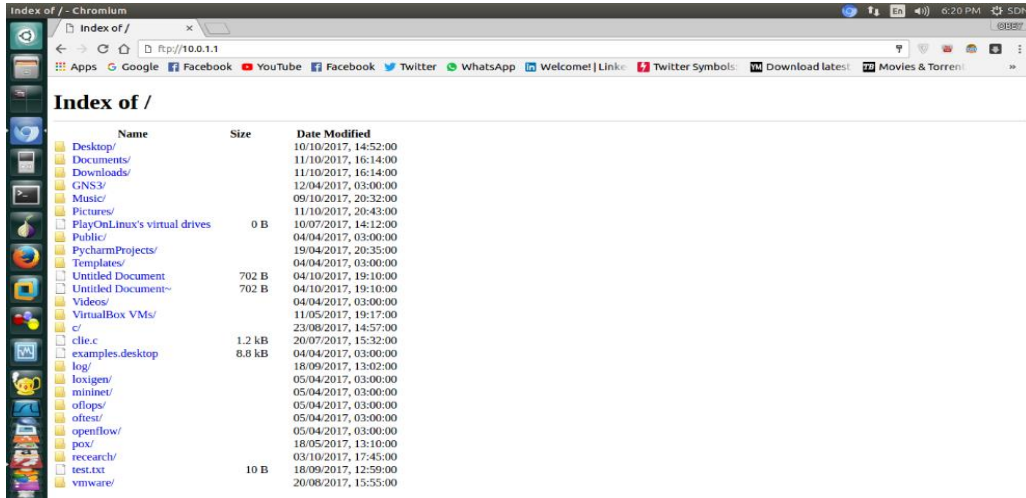


Figure 5.15 FTP request.

Wireshark can provide a better resolution that shows the source IP and the destination IP. Figure 5.16 shows a snapshot captured from Wireshark.

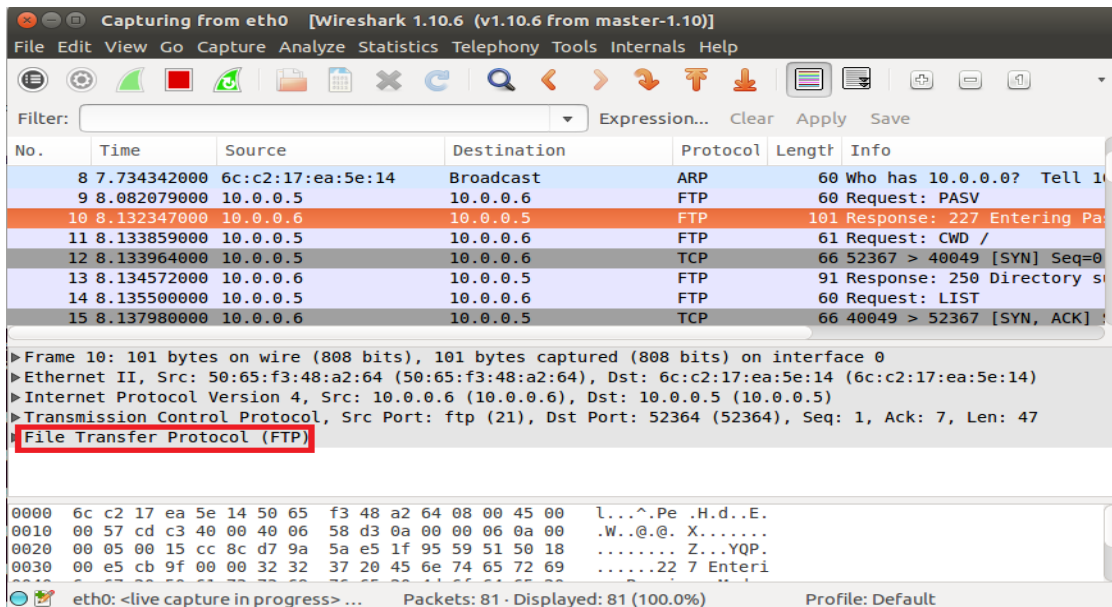


Figure 5.16 Exchange of FTP packets.

5.4.5 Load Balancer

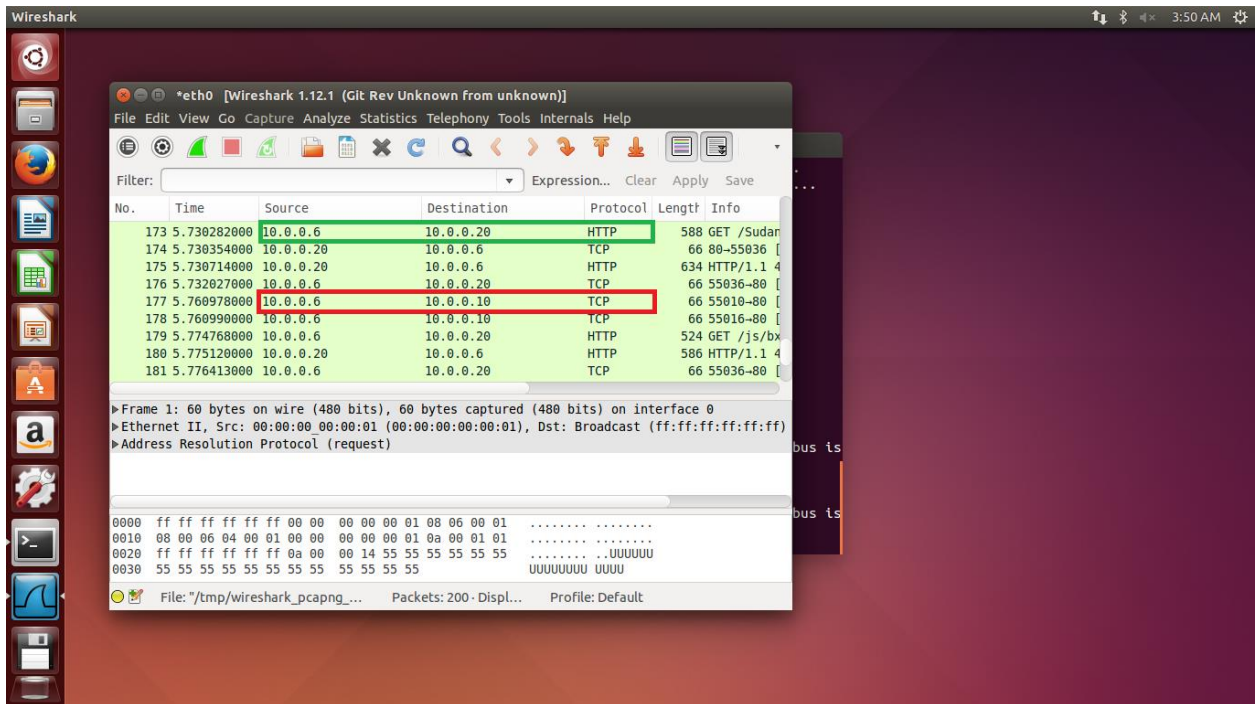


Figure 5.17 Distributing of traffic among servers.

In the load balancer scenario, packets exchanged won't be noticed because the three servers are connected to the network through one external interface, so the distributed traffic among the servers can't be verified by Wireshark interfaces as in previous scenarios. But, if we install Wireshark in one of the servers the received traffic can be seen. Figure 5.17 shows a snapshot from one of the server, where client 10.0.0.6 requested HTTP service. The response in the first time came from server 10.0.0.20 and in the second time came from server 10.0.0.10.

5.5 Summary

This chapter has demonstrated the implementation and results of the SDN including all scenarios. The following chapter is conclusion and future work.

Chapter 6 CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

6.2 Future work

6.1 Conclusions

This thesis explores the implementation of SDN and the migration from a traditional to SDN based datacenter network. The process starts with gathering information about the traditional datacenter network, its design, implementation and configuration. From that information, a sample from the network is taken to transform it to an SDN based network. The sample consists of a main core switch connecting the LAN network with the DMZ area, inside the DMZ there is one server with multiple services. The topology is deployed in a Mininet emulator. The SDN network is deployed by programming the POX controller.

The presented work also highlights the problem of datacenter 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 contributions was to demonstrate that SDN presents a smooth solution for controlling and programming datacenter networks.

The simulation environment is set running three different scenarios Hub, Switch and Load Balancer. The round robin load balancer was successfully implemented in a POX controller, and the software-based nature of a load balancer helped reduce the cost of implementation for users. This provides flexibility in configuration and deployment by allowing SUST to install the software on any white-box or OpenFlow supported device. Thus, significantly reducing the period of time required to deploy new services when compared to a traditional hardware-based approach. The connectivity of these scenarios was performed to demonstrate an aspect of network flexibility. These scenarios are verified through several tests that testify reachability. The reachability test is done on different protocols including HTTP and FTP.

6.2 Future work

This thesis has provided a base for migration from traditional to SDN based network. We recommend that future work should include using different types of controllers which have advantages over the POX to compare which type is better for datacenter networks. Also we believe 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 datacenter.

Another important improvement 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, adding a redundant controller or even several controllers is one of the important issues that should be addressed in future work. In addition, the program can be modified to implement different load balancing algorithms like weighted round robin or IP-based Hashing.

References:

- [1] Nick Feamster, Jennifer Rexford and Ellen Zegura, "The Road to SDN: An Intellectual History of Programmable Networks", acmqueue, 2013.
- [2] SakirSezer, 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] Sridhar K. N. Rao, "SDN AND ITS USE-CASES- NV AND NFV", NEC Technologies India Limited, A State-of-the-Art Survey, White Paper, 2014.
- [4] Open Networking Foundation, "Software-defined networking: The New Norm for Networks" ONF White Paper, Apr. 2012.
- [5] Wenfeng Xia, Yonggang Wen, ChuanHengFoh, DusitNiyato, and HaiyongXie, "A Survey on Software-Defined Networking", IEEE Communication Surveys & Tutorial, Vol. 17, No. 1, First Quarter 2015
- [6] OpenFlow version 1.3 tutorial, [Online], available at <http://sdnhub.org/tutorials/openflow-1-3/> , date accessed: 15/5/2017.
- [7] KHATRI VIKRAMAJEET, "Analysis of OpenFlow Protocol in Local Area Networks", Master of Science Thesis, 62 pages, 4 Appendix pages, TAMPERE University of Technology, August 2013.
- [8] Rufaida Ahmed Mahjoub, "Event-Driven Network Control Using Software-Defined Networking", University Of Khartoum, August 2015.

- [9] Marc F. Körner, "Software Defined Networking based Data-Center Services", Ph.D. dissertation, Univ. of BerlinzurErlangung, July 2015.
- [10] Andreas Voellmy, "Programmable and Scalable Software-Defined Networking Controllers" Ph.D. dissertation, Univ. of Yale University, May 2014.
- [11] Hind Amir Mohammed Salih, WeaamKamilAlbalola Ahmed, Yahia Mohammed Elamin Ahmed, "Implementation of Remote Configuration Using SDN Approach", Sudan University of Science and Technology, September 2014.
- [12] AshkanGhaarinejad, "Comparing a Commercial and an SDN-Based Load Balancer in a Campus Network", Arizona State University, May 2015.
- [13] Mohammed AdilAbdelwahab Mohammed, Mohammed Omar Mohammed AL-Hassan Akoud, MugahedIzzeldin Osman HajAhmed, Mustafa Khalid Mustafa Abdelrahim, "Implementation of SDN in a Campus NAC Use Case", Sudan University of Science and Technology October 2016.