Sudan University of Science and Technology
College of Postgraduate Studies

INVESTIGATION OF SOFTWARE DEFINED NETWORK FOR VIRTUAL PRIVATE NETWORKS

تحقيق الشبكات المعرفة بالرمجة للشبكات الخاصة الافتراضية

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

By:

Asmaa Mubarak Altayeb Awad Alkareem

Supervisor:

Dr. Ibrahim khider

March, 2019

(وَمَا أُوتِيتُم مِّنَ الْعِلْمِ إلا قليلاً)

الإسراء (85)

DEDICATION

To:
The spirit of my hero, Altayeb M.Ibrahim,
My teacher Wayser S.koko,
My Extended family,
My small family,
My friends,
My little Angel…

ACKNOWLEDGMENTS

I wish to express my appreciation and gratitude to Dr. Ibrahim Khider who, through his ideas, suggestions and advice improved this project. Thanks to him, not only for his help in general but also for his trust and guidance during the revision process. My deepest thanks to all the staff in electronic department at Sudan University of Science and Technology, who, in many ways contributed in making this project a memorable and an enriching experience. Finally, I thank my family, Co-workers, friends for their patience and understanding during the days of writing and revising this project.

ABSTRACT

The traditional networks face so many problems including: time-consuming, Multi-vendor environments require a high level of expertise and complicate network segmentation, inconvenience and difficulty of learning to manage such a huge systems and devices and more. Software-defined network continues to be one of the most hyped technology evolutions in information and communication technology that provide a centralized management of the network controlled by one controller and it promise to offer an easy to manage, scalable and high performance networks.

In this study, Mininet simulator and python programming language are used to emulate a software defined network for an Internet service provider in two different models and implement virtual private local area network service networks for three customers-two sites per customer, then begin to scale the network by double the number of sites per every customer using different scenarios in every model in order to evaluate the connectivity and performance of the software defined network controller by observing the number of flows in the flow table that the controller used in data transfer and we observed that: as we scale the network by double the flow table number be scale approximately by three times as in the first model or one and half time as in the next model. Which means that the scalability problem takes place also in SDN networks and it needs more studies.

المستلخص

إن الشبكات التقليدية تواجه العديد من المشكلات منها :أنها تستغرق وقت ا طويلا ، كما أن عليها أن تعمل في بيئات مختلفة؛ مما يتطلب مستوى عالي من الخبرة كذلك صعوبة إدارة الشبكة لأنها تكون مجزئة وصعوبة التعلم لإدارة كل هذه النظم والأجهزة الضخمة وأكثر من ذلك.

تعتبر الشبكات المعرفة بالبرمجيات واحدة من أكثر التطورات التكنولوجية تأثير ا في تكنولوجيا المعلومات والاتصالات والتي توفر إدارة مركزية للشبكة حيث سيكون التحكم فيها عبر متحكم واحد وقد وعدت بذلك بتوفير شبكات سهلة الإدارة قابلة للتوسع وبأداء عالي.

في هذه الدراسة تم استخدام برنامج المحاكاة(ميني نت)ولغة البرمجة(بايثون)لعمل شبكة معرفة بالبرمجة لمزود خدمة انترنت في نموذجين مختلفين وتطبيق خدمة الشبكة المحلية الافتراضية لثلاثة مشتركين بموقعين لكل مشترك، ثم بدأنا بتوسيع الشبكة عبر مضاعفة عدد المواقع لكل مشترك باستخدام سناريوهات مختلفة في كل نموذج بغرض تقييم توصيل وأدائية المتحكم في الشبكات المعرفة بالبرمجة، عن طريق ملاحظة عدد مدخلات جدول التدفق الذي يجب علي المتحكم الرجوع إليه لتوصيل البيانات، وقد لاحظنا أنه كلما توسعت الشبكة بمقدار الضعف توسع جدول التدفق بثلاثة أضعاف تقريب ا كما في النموذج الأول، أو مرة ونصف كما في النموذج الثاني، مما يدل أن مشكلة توسع الشبكة قائمة في الشبكات المعرفة بالرمجة أيض ا وتحتاج إلى دراسات كثيرة.

LIST OF TABLES

LIST OF FIGURES

GLOSSARY

Abbreviation	Definition:
SDN	Software Defined Networks
API	Application Programmer Interface
WAN	Wide Area Network
VPN	Virtual Private Network
IP	Internet Protocol
MPLS	Multi-Protocol Label Switching
VPLS	Virtual Private Local Area Network Service
ISP	Internet Service Providers
ONOS	Open Network Operating System
CLI	Command Line Interface
VPDN	Virtual Private Dial-up Network
LAN	Local Area Network
ID	Identity
IPsec	Internet Protocol Security
L2TP	Layer Two Tunneling Protocol
PPTP	Point-to-Point Tunneling Protocol
SSL	Secure Sockets Layer
TLS	Transport Layer Security
SSH	Secure Shell
SNMP	Simple Network Management Protocol
L2	Layer Two
L2VPN	Layer Two Virtual Private Network
L3VPN	Layer Three Virtual Private Network
CE	Customer Edge
PE	Provider Edge
LSP	Label Switched Path
AS	Autonomous Systems
VE	Virtual Private Local Area Network Service Edge
STP	Spanning Tree Protocol
IETF	Internet Engineering Task Force
RSVP	Resource Reservation Protocol
OSPF	Open Shortest Path First
ATM	Asynchronous Transfer Mode
LSR	Label Switch Router
P	Provider
BGP	Border Gateway Protocol
ONF	Open Network Foundation
QoS	Quality of Service
NAT	Network Address Translation
OSGi	Open Services Gateway Initiative

SAL	**Service Abstraction Layer**
I/O	**Input/Output**
ASIO	**Asynchronous Input/Output**
OS	**Operating System**
OPENSIG	**Open Signaling Working Group**
GSMP	**General Switch Management Protocol**
ForCES	**Forwarding and Control Element Separation**
TCP	**Transmission Control Protocol**
VM	**Virtual Machine**

TABLE OF CONTENTS

1. Introduction

1.1 Preface:

Computer networks are typically built from a large number of network devices such as routers, switches and firewalls with many complex protocols implemented on them. Network operators are responsible for configuring policies to respond to a wide range of network events and applications. They have to manually transform these high-level policies into low-level configuration commands while adapting to changing network conditions. The fact that network devices are usually vertically integrated so network operators and administrators have to make this task so many time and that the challenge they are always facing.

Other challenge, the Internet practitioners and researchers facing is that the Internet infrastructure have to evolve current and emerging services and applications as well as its protocols and performance. Traditional networks have various problems, it is very complex, difficult to manage and control; adding new features means new protocols and new hardware (vender dependent) so innovation costs a lot.

The limitations of traditional networking technologies make it harder to determine where security devices such as firewalls should be deployed in the network.

Programmable networks are proposed to resolve traditional network approaches problems and adding new features. One of the programmable networks is **S**oftware **D**efined **N**etworks (**SDN**). SDN is currently attracting significant attention from both academic and industrial field. SDN is a new networking paradigm in which the forwarding hardware is decoupled from control decisions it decouples the data plane from control plane and connect them throw a protocol like open flow in **A**pplication **P**rogrammer **I**nterface (**API**).

SDN can overcome the classic problem by implementing a central firewall in the network, and thereby network administrators can route all traffic through a central firewall. With the help of SDN, a vendor independent control from a single logical point can be obtained.

In SDN, the network intelligence is logically centralized in software-based controllers (the control plane), and network devices become simple packet forwarding devices (the data plane) that can be programmed via an open interface. It promises to dramatically simplify network management and enable innovation and evolution. A group of network operators, service providers, and vendors have recently created the Open Network. Centralized the management in a controller make it easy and flexible to manage but at the same time it have to approve that it can add scalability to applied on the **W**ide **A**rea **N**etwork (**WAN**) [1].

Virtual **P**rivate **N**etwork (**VPN**) services are considered to be widely used in **I**nternet **P**rotocol/**M**ulti-**P**rotocol **L**abel **S**witching (**IP/MPLS**) networks for connecting customers' remote sites. However, service providers struggle with many challenges to provide these services. Management complexity, equipment costs, and last but not least, scalability issues emerging as the customers increase in number, are just some of these problems. SDN is an emerging paradigm that can solve aforementioned issues using a logically centralized controller for network devices [4].

1.2 Problem Statement:

Traditional networks, MPLS VPN and **V**irtual **P**rivate **L**ocal Area Network **S**ervice (**VPLS**) services impose some serious issues: The distributed architecture of control plane causes complexity in configuration of the network done in advice by device manner. For **I**nternet **S**ervice **P**roviders (**ISP**) high-performance routers are needed to meet the demands of vertically integrated, customers' numerous IP prefixes which costs a lot.

1.3 Objectives:

The objectives of this thesis are to:

1. Verify the efficiency of SDN in scalability issue.

2. Implement different SDN VPLS scenarios simulating to show challenges face SDN networks in the term of scalability when it is used at WAN.

1.4 Methodology:

In this study we will implement two different models using Mininet simulator and SDN **O**pen **N**etwork **O**perating **S**ystem (**ONOS**) controllers configured by **C**ommand **L**ine **I**nterface (**CLI**).

1.5 Thesis Outlines:

This thesis contains a total of six chapters, the brief outline of these chapters is as follows:

Chapter 1: presents the introduction and contains the problem which carries out this thesis and the project goals. At the end of this chapter the methodology used to work in this thesis is illustrated.

Chapter 2: gives a background about the virtual private networks.

Chapter 3: presents the modeling of the software defined network for a virtual private network.

Chapter 4: explains the simulating software defined network for the virtual private network.

Chapter 5: presents the obtained results and a brief discussion.

Chapter 6: includes the thesis conclusion and recommendations.

2. Background

2.1 Introduction:

Simply, a Virtual Private Network, or VPN, is a group of computers (or discrete networks) networked together over a public network—namely, the internet VPN is a network that allow you to create a secure connection to another network over the Internet. It provides inter-connectivity to exchange information among various entities that belong to the VPN. It is private, so that it has all the characteristics of a private network.

VPN is a generic term used to describe a communication network that uses any combination of technologies to secure a connection tunneled through an otherwise unsecured or untrusted network. Instead of using a dedicated connection, such as leased line, a "virtual" connection is made between geographically dispersed users and networks over a shared or public network, like the Internet. Data is transmitted as if it were passing through private connections [2].

There are a number of systems that enable you to create networks using the Internet as the medium for transporting data. A VPN secures the private network, using encryption and other security mechanisms to ensure that only authorized users can access the network and that the data cannot be intercepted. The primary reason for deploying VPN is cost savings. VPN technology provides a way of protecting information being transmitted over the Internet, by allowing users to establish a virtual private “tunnel” to securely enter an internal network, accessing resources, data and communications via an insecure network such as the Internet. There is an increasing demand nowadays to connect to internal networks from distant locations. Employees often need to connect to internal private networks over the Internet (which is by nature insecure) from home, hotels, airports or from other external networks. Security becomes a major consideration when staff or business partners have constant access to internal networks from insecure external locations [4].

2.2 VPN Characteristics:

1) Supports a closed community of authorized users, allowing them to access various network related services and resources.
2) The traffic originating and terminating within a private network traverses only those nodes that belong to the private network.
3) The traffic corresponding to this private network does not affect nor is it affected by other traffic extraneous to the private network.
4) Virtual topology is built on an existing, shared physical network infrastructure. However, the virtual topology and the physical network are usually administered by different administrative bodies.

2.3 VPN Types:

There are two common types of VPNs:

2.3.1 Remote-Access also called a Virtual Private Dial-up Network (VPDN):

This is a user-to-**L**ocal **A**rea **N**etwork (**LAN**) connection used by a company that has employees who need to connect to the private network from various remote locations. Typically, a corporation that wishes to set up a large remote-access VPN provides some form of Internet dial-up account to their users using an ISP.

The telecommuters can then dial a 1−800 number to reach the Internet and use their VPN client software to access the corporate network. A good example of a company that needs a remote-access VPN would be a large firm with hundreds of sales people in the field. Remote-access VPNs permit secure, encrypted connections between a company's private network and remote users through a third party service provider.

2.3.2 Site-to-Site:

Through the use of dedicated equipment and large scale encryption, a company can connect multiple fixed sites over a public network such as the

Internet. Each site needs only a local connection to the same public network, thereby saving money on long private leased-lines.

Site-to-site VPNs can be further categorized into intranets or extranets. A site-to-site VPN built between offices of the same company is said to be an intranet VPN, while a VPN built to connect the company to its partner or customer is referred to as an extranet VPN [3].

2.4 VPN Design:

Features are needed in a well-designed VPN:

2.4.1 Security:

When using public network security techniques are needed, to be effective, a VPN must address the following basic requirements: Data integrity, to verify that the contents of a datagram were not changed in transit, either deliberately or due to random errors. Data confidentiality, to conceal the clear text of a message by using encryption. Replay protection, to ensure that an attacker cannot intercept a datagram (containing, for example, an encrypted user **Id**entity **ID** and password) and play it back at some other time. Key management to ensure that your VPN policy can be implemented throughout the extended network with little or no manual configuration. Interoperability, to ensure that VPN uses standard-based technologies to maintain interoperability with other VPN vendors. Figure 2.1 illustrates a VPN connection between a server and a client [8].

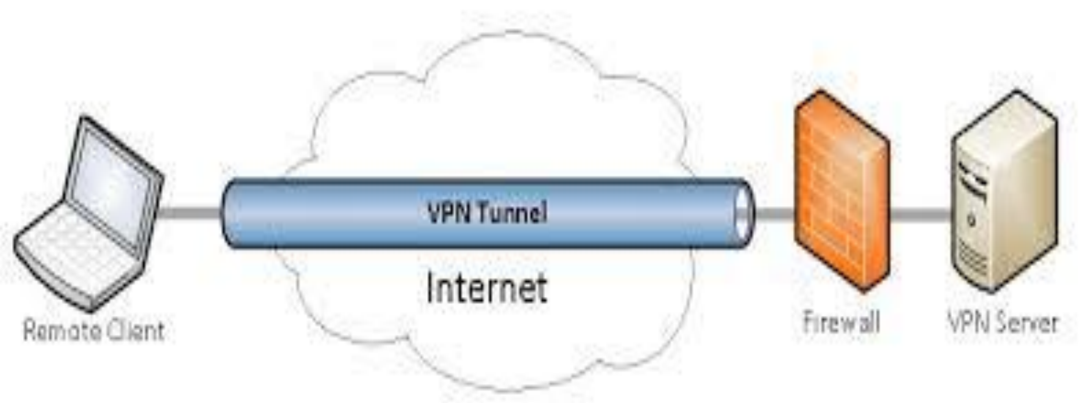


Figure 2.1: VPN Connection between Server and Client

VPN technology is based on the idea of tunneling. VPN tunneling involves establishing and maintaining a logical network connection. On this connection, packets constructed in a specific VPN protocol format are encapsulated within some other base or carrier protocol, then transmitted between VPN client and server, and finally de-encapsulated on the receiving side. There are six VPN tunneling protocols:

1) **I**nternet **P**rotocol **Sec**urity (**IPsec**)
2) **L**ayer **Two T**unneling **P**rotocol (**L2TP**)
3) **P**oint-to-**P**oint **T**unneling **P**rotocol (**PPTP**)
4) **S**ecure **S**ockets **L**ayer (**SSL**) and **T**ransport **L**ayer **S**ecurity (**TLS**)
5) OpenVPN.
6) **S**ecure **Sh**ell (**SSH**)

2.4.2 VPN Reliability:

Reliability is a function of time. The way in which time is specified varies considerably depending on the nature of the system under consideration. For example, if a system is expected to complete its mission in a certain period of time, like in case of a spacecraft, time is likely to be defined as a calendar time or as a number of hours. For software, the time interval is often specified in so called natural or time units. A natural unit is a unit related to the amount of processing performed by a software-based product, such as pages of output, transactions, telephone calls, jobs or queries [5].

2.4.3 VPN Scalability:

There are several different tunneling protocols that can be used to create VPN connections: Point to Point Tunneling Protocol (PPTP), Layer 2 Tunneling Protocol (L2TP), Internet Protocol Security (IPsec) and Secure Sockets Layer (SSL). Some VPN solutions support more than one of these protocols others are more limited. Scalability needs will affect which tunneling protocols are most appropriate. Remote access users must have the proper client software to support the protocols you choose. For site-to-site VPNs, the VPN gateways at each end must support a common protocol [9].

2.4.4 Network Management:

Network management as monitoring, testing, configuring, and troubleshooting network components to meet a set of requirements defined by an organization. These requirements include the smooth, efficient operation of the network that provides the predefined quality of service for users. To accomplish this task, a network management system uses hardware, software, and humans. In this chapter, first we briefly discuss the functions of a network management system. Then we concentrate on the most common management system, the **S**imple **N**etwork **M**anagement **P**rotocol (**SNMP**).

The functions performed by a network management system can be divided into five broad categories: configuration management, fault management, performance management, security management, and accounting management [7].

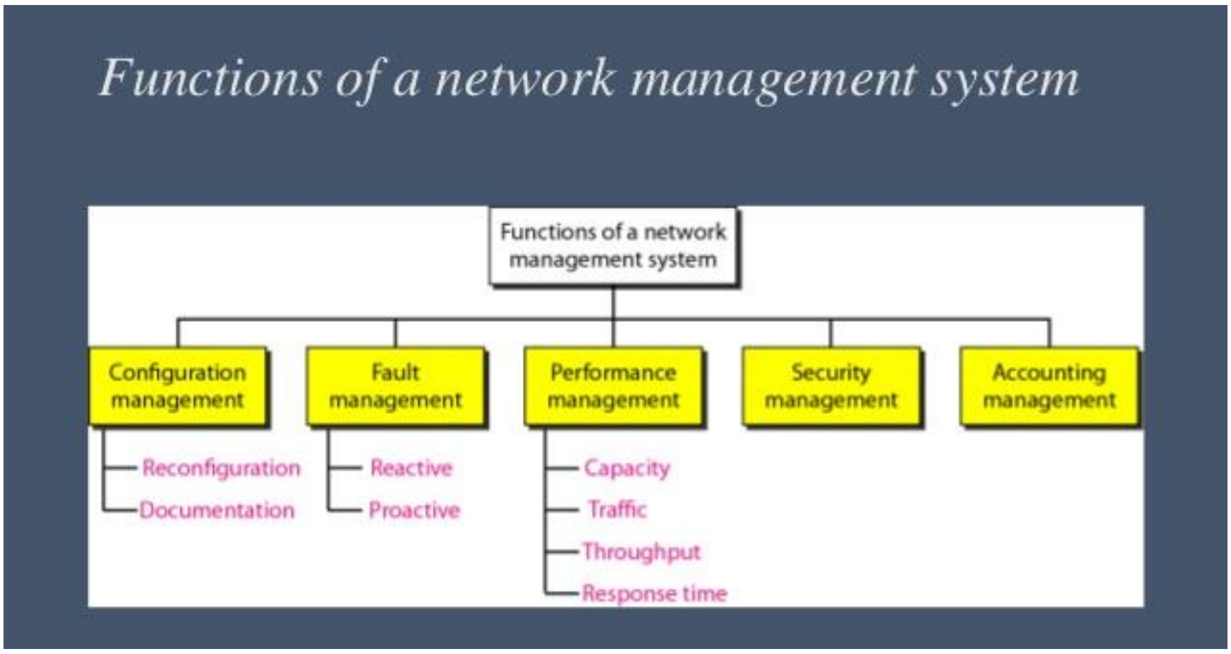


Figure 2.2: Functions of Network Management System

2.4.5 Policy Management:

The management of network infrastructure in an enterprise is a complex and daunting affair. In an era of increasing technical complexity, it is becoming difficult to find trained personnel who can manage the new features introduced into the various servers, routers, and switches. Policy-based network management provides a means by which the administration process can be simplified and largely automated. In this article we look at a general policy-based architecture that can be used to simplify several new technologies emerging in the context of IP networks.

2.4.6 Virtual Private LAN Service (VPLS):

Virtual Private LAN Service (VPLS) is an SDN application that enables operators to create **L**ayer **Two** (**L2**) broadcast overlay networks on-demand, on top of Open Flow infrastructures. The application connects into overlay broadcast networks hosts connected to the Open Flow data plane.

VPLS, in its implementation and configuration, has much in common with a **L**ayer **Two** **V**irtual **P**rivate **N**etwork (**L2VPN**). In VPLS, a packet originating within a service provider customer’s network is sent first to a **C**ustomer **E**dge (**CE**) device (for example, a router or Ethernet switch). It is then sent to a **P**rovider **E**dge (**PE**) router within the service provider network. The packet traverses the service provider network over an MPLS **L**abel **S**witched **P**ath (**LSP**). It arrives at the egress PE router, which then forwards the traffic to the CE device at the destination customer site. The difference is that, for VPLS, packets can traverse the service provider networks in point-to-multipoint fashion, meaning that a packet originating from a CE device can be broadcast to all the PE routers participating in a VPLS routing instance.

In contrast, a L2VPN forwards packets in point-to-point fashion. The paths carrying VPLS traffic between each PE router participating in a routing instance are called pseudo wires. VPLS multi-homing enables you to connect a customer site to multiple PE routers to provide redundant connectivity while preventing the formation of L2 loops in the service provider network. A VPLS site that is multihomed to two or more PE routers provides redundant connectivity in the event of a PE router-to-CE device link failure or the failure of a PE router. When multi-homing a VPLS site (potentially in different **A**utonomous **S**ystems [**AS**s]), the PE routers connected to the same site can either be configured with the same **V**PLS **E**dge (**VE**) device identifier or with different VE device identifiers. If you are using different VE device identifiers, you must run the **S**panning **T**ree **P**rotocol (**STP**) on the CE device, and possibly on the PE routers, to construct a loop-free VPLS topology [9].

2.5 Multi-Protocol Label Switching MPLS:

Software Defined Network SDN is a new network architecture that can be implements at both WAN and LAN networks. In our research we are focusing on WAN network so before we study SDN one must understand the current WAN technology which will be work concurrently with SDN before complete replacement.

MPLS is an **I**nternet **E**ngineering **T**ask **F**orce (**IETF**)–specified framework that provides for the efficient designation, routing, forwarding, and switching of traffic flows through the network. MPLS performs the following functions:

- Specifies mechanisms to manage traffic flows of various granularities, such as flows between different hardware, machines, or even flows between different applications.
- Remains independent of the Layer-2 and Layer-3 protocols.
- Provides a means to map IP addresses to simple, fixed-length labels used by different packet-forwarding and packet-switching technologies.
- Interfaces to existing routing protocols such as Resource Reservation Protocol (RSVP) and **O**pen **S**hortest **P**ath **F**irst (**OSPF**)
- Supports the IP, **A**synchronous **T**ransfer **M**ode (**ATM**), and frame-relay Layer-2 protocols.

2.5.1 MPLS Architecture:

- Provider Edge (PE) router –also known as an Ingress/Egress **L**abel **S**witch **R**outer (**LSR**), is a router between one network service provider's area and areas administered by other network providers/customers.
- **P**rovider (**P**) router –is a Label Switch Router (LSR) that functions as a transit router of the core network. The P Router is typically connected to one or more PE Routers.
- Customer Edge (CE) router–The customer edge (CE) is the router at the customer premises that is connected to the provider edge of a service provider IP/MPLS network. CE peers with the Provider Edge (PE) and exchanges routes with the corresponding VRF inside the PE. The routing protocol used could be static or dynamic (an interior gateway protocol like OSPF or an exterior gateway protocol like **B**order **G**ateway **P**rotocol-**BGP**).

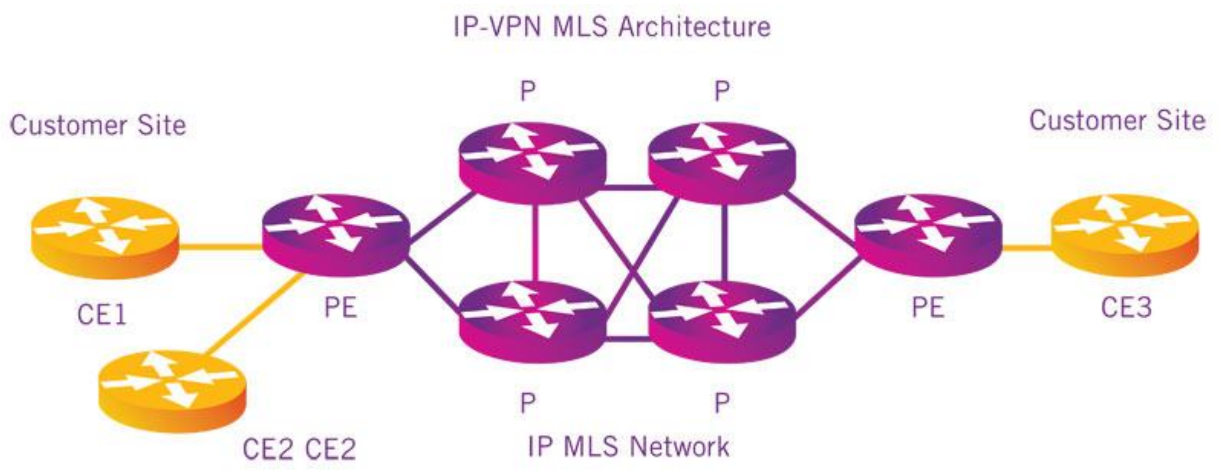


Figure 2.3: MPLS Architecture

2.5.2 Labels and Label Bindings:

A label, in its simplest form, identifies the path a packet should traverse. A label is carried or encapsulated in a Layer-2 header along with the packet. The receiving router examines the packet for its label content to determine the next hop. Once a packet has been labeled, the rest of the journey of the packet through the backbone is based on label switching. The label values are of local significance only, meaning that they pertain only to hops between LSRs.

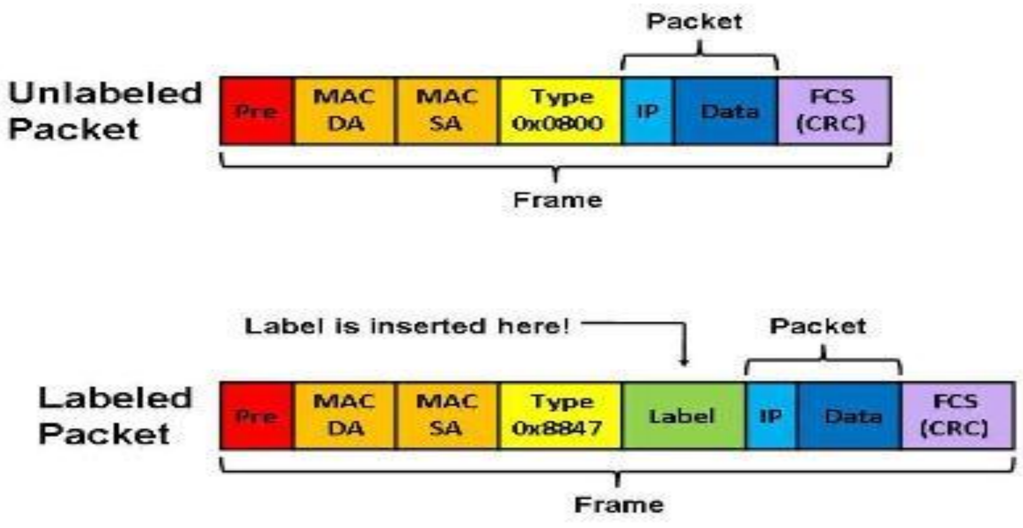


Figure 2.4: MPLS Labeling

2.5.3 The MPLS Process:

There are four scenarios detailing how LSRs forward packets:

1) An unlabeled IP packet is received, and is routed unlabeled to the next hop.
2) An unlabeled IP packet is received, a label is inserted in the header, and is switched to the next hop.
3) A labeled IP packet is received, the label is swapped, and is switched to the next hop.
4) A labeled IP packet is received, the label is stripped off, and is routed to the next hop or destination.

Frame-mode MPLS performs as follows:

1) An edge LSR receives a packet.
2) The edge LSR performs a routing table lookup to determine the next hop (or exit interface).
3) If destined for the MPLS network, the edge router inserts the label between the Layer-2 and Layer-3 headers.
4) The edge LSR forwards the labeled packet to the core LSR.
5) Core LSRs will route solely based on the label, and will not perform a routing table lookup [10].

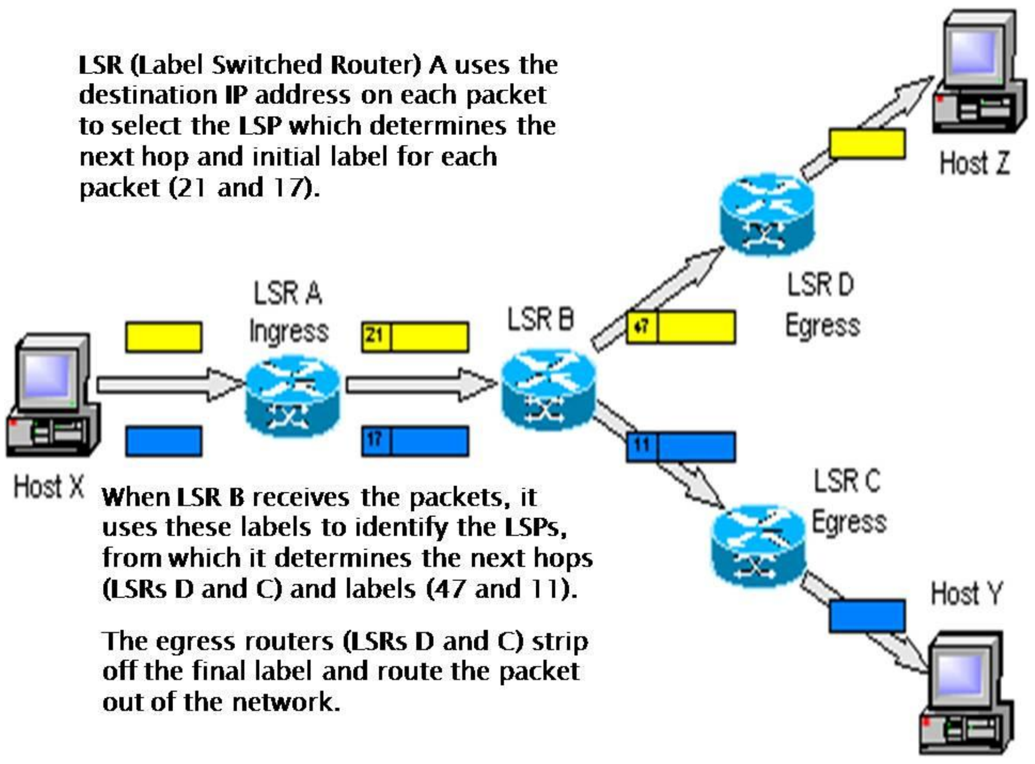

LSR (Label Switched Router) A uses the destination IP address on each packet to select the LSP which determines the next hop and initial label for each packet (21 and 17).
Host Z
LSR A Ingress
21
LSR B
47
LSR D Egress
17
11
Host X
When LSR B receives the packets, it uses these labels to identify the LSPs, from which it determines the next hops (LSRs D and C) and labels (47 and 11).
LSR C Egress
Host Y
The egress routers (LSRs D and C) strip off the final label and route the packet out of the network.

Figure 2.5: MPLS Process

2.5.4 MPLS L2VPN VS L3VPN:

- L3VPN uses IP routing, L2VPN uses circuit switching approach.
- In L3 MPLS VPN the customer traffic consist of IP frames, in L2 MPLS VPN the customer traffic is tagged or untagged Ethernet frames.
- In L3 MPLS VPN the inner label is a label containing the final virtual routing and forwarding table, in L2 MPLS VPN the inner label is virtual circuit tag
- L3 MPLS VPN uses standard routing protocols such as BGP to create route maps, L2 MPLS VPN resamples a virtual circuit type service.

2.6 SDN Network:

Software-defined networking (SDN) is a new networking architecture that comes after MPLS Technology is proposed as a facilitating technology for network evolution and network virtualization. It has attracted significant attention from both academic researchers and industry.

One of the main organizations that contribute to the development of SDN is the **O**pen **N**etwork **F**oundation (**ONF**) which is a non-profit industry consortium of network operators, service providers and vendors that promotes the SDN architecture and drives the standardization process of its major elements.

ONF defines SDN as a technology where "network control is decoupled from forwarding and is directly programmable". It concentrates the network intelligence in software-based central controllers, which aims to bring better and more efficient control, customizability and adaptability.

The main benefits that the SDN technology might offer are: Centralized unified control of network devices from different vendors, better automation and control, as an abstraction of the real network is created simplified and quicker implementation of innovations, as the network control is centralized and there is no need every individual device to be reconfigured. Improved network reliability and security, because of fewer configuration errors and unified policy enforcement, provided by the automated management and the centralized control Ability to easily adapt the network operation to changing user needs, as centralized network state information is available and can be exploited.

2.6.1 SDN Architecture:

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

Let us first try and peel back the layers of SDN and how it impacts networking. Networking typically involves a collection of switches and routers that work in harmony to achieve end to end communication. The key functions of these network elements can be segmented into layers of management, data plane and control plane. The traditional way of making these nodes work with each other is by implementing protocols running at each of these nodes to exchange information. This creates a distributed architecture, where every node across the network needs to be at a similar state to get the desired end result. In addition, these protocols are very rigid in what they can and cannot do. The result is a very static network architecture that is not adaptive to change as presented in Figure 2.6

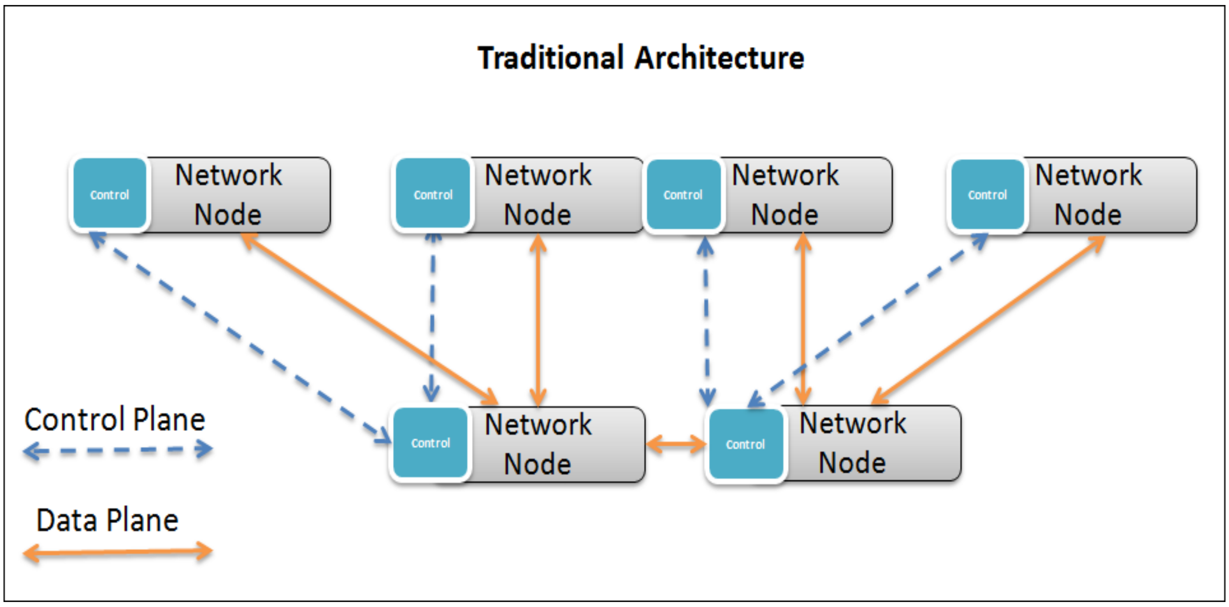


Figure 2.6: Traditional Architecture

Now consider what would happen if we remove the protocols and instead open up a standard set of APIs. Then, build a centralized control plane that uses these APIs to program the network elements. This control plane will have a global view of the network and can make smart decisions. For example, how can one carve out a dedicated path between 2 servers? If we had switches opening up APIs indicating the flow to the output port mapping it is a matter of programming all the elements with that information. Imagine trying to do that with the spanning tree protocol instead! This is just a very high level concept, but the fundamental idea is that network elements need to be programmable and cannot be static within a fluid environment like the Cloud, where provisioning needs to happen on demand and elasticity is a key requirements presented in Figure 2.7

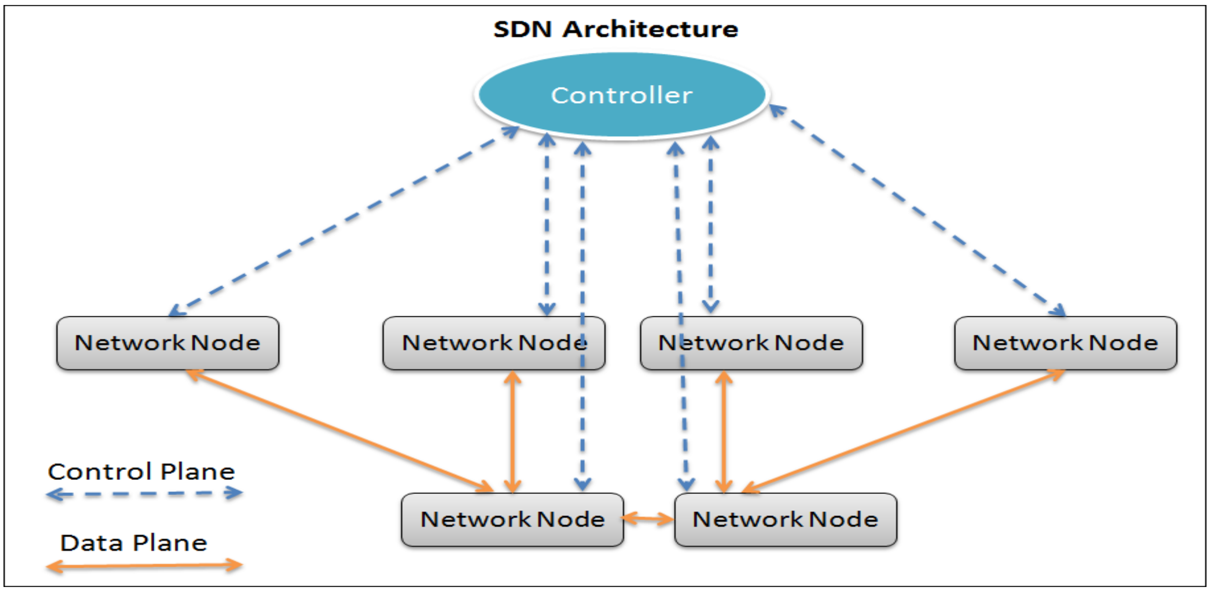


Figure 2.7: SDN Architecture

Moving the same concept into enterprise networking, Firewalls, VPN, WAN optimization solutions and **Q**uality **of S**ervice (**QoS**) are some of the aspects of WAN technologies built on a foundation of L3 routing. L3 routing is destination based and is not flow aware. It does have significant benefits over L2 networks, like support for multi-pathing, VPNs but is built on protocols running in a distributed manner and lacking programmability.

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

2.6.2 SDN Applications:

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

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

2.6.3 SDN Controller:

The controller is the core of an SDN network. It lies between network devices at one end and applications at the other end. Any communications between applications and devices have to go through the controller [11]. SDN controllers are based on protocols, such as Open Flow to configure network devices and choose the optimal network path for application traffic and to allow servers to tell switches where to send packets as presented in Figure 2.8.

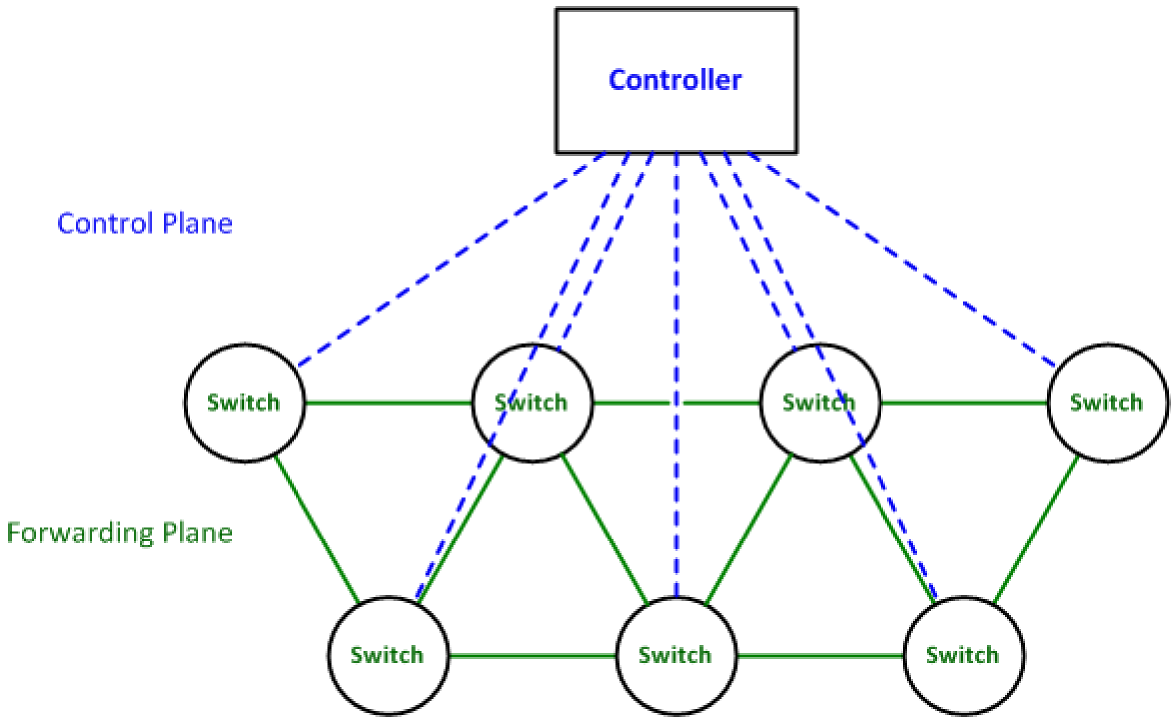


Figure 2.8: SDN Controller

2.6.3.1 Open Daylight:

Open Daylight SDN Controller has several layers. The top layer consists of business and network logic applications, the middle layer is the framework layer and the bottom layer consists of physical and virtual devices. The middle layer is the framework in which the SDN abstractions can manifest. This layer hosts north-bound and southbound APIs. The controller exposes open northbound APIs which are used by applications. Open Daylight supports the **O**pen **S**ervices **G**ateway **I**nitiative (**OSGi**) framework and bidirectional REST for the northbound API. The business logic resides in the applications above the middle layer. These applications use the controller to gather network intelligence, run algorithms to perform analytics, and then use the controller to orchestrate the new rules, if any throughout the network. Open daylight is created with an objective of reducing vendor, locking therefore it supports protocols other than Open Flow. The southbound interface is capable of supporting multiple protocols such as Open Flow and BGP-Link State as separate plugins. The **S**ervice **A**bstraction **L**ayer (**SAL**) determines how to fulfill the requested service irrespective of the underlying protocol used between the controller and the network devices [12].

2.6.3.2 NOX-MT:

NOX, whose measured performance motivated several recent proposals on improving control plane efficiency has a very low flow setup throughput and large flow setup latency. Fortunately, this is not an intrinsic limitation of the SDN control plane: NOX is not optimized for performance and is single-threaded. We present NOX-MT, a slightly modified multi-threaded successor of NOX, to show that with simple tweaks we were able to significantly improve NOX's throughput and response time. The techniques we used to optimize NOX are quite well-known including: **I**nput/**O**utput (**I/O**) batching to minimize the overhead of I/O, porting the I/O handling harness to Boost **As**ynchronous **I/O** (**ASIO**) library (which simplifies multi-threaded operation), and using a fast multiprocessor-aware malloc implementation that scales well in a multi-core machine. Despite these modifications, NOX-MT is far from perfect. It does not address many of NOX's performance deficiencies, including but not limited to: heavy use of dynamic memory allocation and redundant memory copies on a per request basis, and using locking were robust wait-free alternatives exist. Addressing these issues would significantly improve NOX's performance. However, they require fundamental changes to the NOX code base and we leave them to future work. To the best of our knowledge, NOX-MT was the first effort in enhancing controller performance and motivated other controllers to improve. [14]

2.6.3.3 Floodlight controller:

Floodlight is not just an Open Flow controller. Floodlight is an Open Flow controller (the "Floodlight Controller") and a collection of applications built on top the Floodlight Controller.

The Floodlight Controller realizes a set of common functionalities to control and inquire an Open Flow network, while applications on top of it realize different features to solve different user needs over the network. The figures below show the relationship among the Floodlight Controller, the applications built as Java modules compiled with Floodlight, and the applications built over the Floodlight REST API. When you run Floodlight, the controller and the set of Java module applications (those loaded in the floodlight properties file, see Module Applications for examples) start running. The REST APIs exposed by all running modules are available via the specified REST port (8080 by default). Any REST applications, written in any language, can now retrieve information and invoke services by sending http REST commands to the controller REST port 8080 [13].

2.6.3.4 Open Network Operating System Controller:

Open Network Operating System (ONOS) controller, is an SDN **O**perating **S**ystem (**OS**) proposed with many features: performance, scalability, and availability requirements of large operator networks.

The evaluation of the first ONOS controller indicated that needs to design a more efficient data model, reduce the number of expensive data store operations, and provide fast notifications and messaging across nodes. Additionally, the API needed to be simplified to represent the network view abstraction more clearly without exposing unnecessary implementation details.

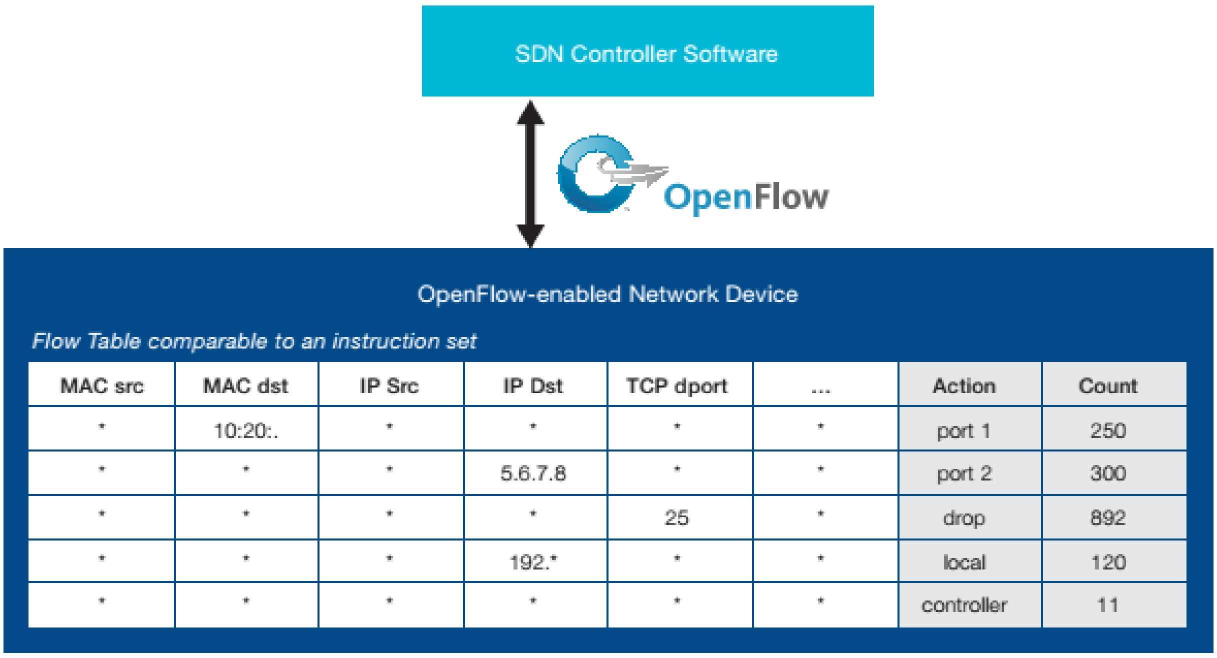
2.6.4 Open Flow Protocol:

Open Flow is currently the only open standard for implementing SDN and it is a standardized protocol that defines the communication between the control and the data forwarding plane in the SDN architecture. It moves the control out of the networking devices (routers, switches, etc.) into the centralized controller.

The protocol uses the concept of flows that use match rules to determine how the packets will be handled. The protocol is configured on both sides – the device and the controller. The forwarding. Device in an Open Flow scenario is an Open Flow switch that contains one or more flow tables and an abstraction layer that communicates with the controller. The flow tables are filled with flow entries which define how the packet will be forwarded, depending on the particular flow they are part of.

In order to understand the role of Open Flow and its building blocks, and how it can be used for Open Flow-based network application development, it is important to provide a brief introduction of Open Flow and how it works. This chapter shapes the required knowledge prior to the actual setup of SDN/Open Flow-enabled experimental and development environment. Open Flow can be considered as one of the early implementations of the SDN concept. Therefore, before going through Open Flow, it is worth giving a brief introduction to the SDN and the related activities around it.

The flow entries have the following fields: Match fields, might contain information from the packet headers, ingress port or metadata and matches the packets to a certain flow. Counters collect statistic about the particular flow. Actions, define how the incoming packets to be handle. An example of the Open Flow instruction set is presented on Figure 2.12


SDN Controller Software
OpenFlow
OpenFlow-enabled Network Device
Flow Table comparable to an instruction set

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

Figure 2.12: Open Flow instruction set

SDN is possible without using the Open Flow standard, but proprietary alternatives would lock an operator into vendor-defined solutions, capabilities and pricing. This would greatly reduce the value of SDN as it would result in the loss of both device interoperability and multi-network Interoperability. An Open Flow switch essentially receives data packets, extracts the packet header and matches the value to the entries in the flow table. If the value is found the packet is forwarded according to the instructions in the actions fields. In case the value does not match any of the entries, the packet is handled according to the instructions defined in the table-miss entry.

The packet can be either dropped, forwarded to the next flow table or send to the Open Flow controller via the control channel. Another possibility, employed in switches that have both Open Flow and non-Open Flow ports, is to forward the packet using standard IP-forwarding schemes. The Open Flow switch communicates with the controller over a secure channel. The controller adds, removes or updates the entries in the flow table [6].

2.6.4.1 Activities around SDN/Open Flow:

While Open Flow has received a considerable amount of industry attention, it is worth mentioning that the idea of programmable networks and decoupled control plane (control logic) from data plane has been around for many years. The Open Signaling Working Group (OPENSIG) initiated a series of workshops in 1995 to make ATM, Internet, and mobile networks more open, extensible, and programmable. Motivated by these ideas, an Internet Engineering Task Force (IETF) working group came up with **G**eneral **S**witch **M**anagement **P**rotocol (**GSMP**), to control a label switch. This group is officially concluded and GSMPv3 was published in June, 2002. The Active Network initiative proposed the idea of a network infrastructure that would be programmable for customized services. However, Active Network never gathered critical mass, mainly due to practical security and performance concerns. Starting in 2004, the 4D project advocated a clean slate design that emphasized separation between the routing decision logic and the protocols governing the interaction between network elements. The ideas in the 4D project provided direct inspiration for later works such as NOX, which proposed an operating system for networks in the context of an Open Flow-enabled network. Later on in 2006, the IETF Network Configuration Protocol working group proposed NETCONF as a management protocol for modifying the configuration of network devices. The working group is currently active and the latest proposed standard was published in June, 2011. The IETF **For**warding and **C**ontrol **E**lement **S**eparation (**ForCES**) working group is leading a parallel approach to SDN. SDN and Open Networking Foundation share some common goals with ForCES. With ForCES, the internal network device architecture is redefined as the control element is separated from the forwarding element, but the combined entity is still represented as a single network element to the outside world. The immediate predecessor to Open Flow was the Stanford's SANE/Ethane project which, in 2006, defined a new network architecture for enterprise networks. Ethane's focus was on using a centralized controller to manage policy and security in a network.

2.6.4.2 Open Flow Messages:

The communication between the controller and switch happens using the Open Flow protocol, where a set of defined messages can be exchanged between these entities over a secure channel. The secure channel is the interface that connects each Open Flow switch to a controller. The Transport Layer Security (TLS) connection to the user-defined (otherwise fixed) controller is initiated by the switch on its power on. The controller's default TCP port is 6633. The switch and controller mutually authenticate by exchanging certificates signed by a site-specific private key. Each switch must be user-configurable with one certificate for authenticating the controller (controller certificate) and the other for authenticating to the controller (switch certificate).

Traffic to and from the secure channel is not checked against the flow table and therefore the switch must identify incoming traffic as local before checking it against the flow table. In the case that a switch loses contact with the controller, as a result of an echo request timeout, TLS session timeout, or other disconnection, it should attempt to contact one or more backup controllers. If some number of attempts to contact a controller (zero or more) fail, the switch must enter emergency mode and immediately reset the current TCP connection. Then the matching process is dictated by the emergency flow table entries (marked with the emergency bit set). Emergency flow modify messages must have timeout value set to zero. Otherwise, the switch must refuse the addition and respond with an error message. All normal entries are deleted when entering emergency mode. Upon connecting to a controller again, the emergency flow entries remain. The controller then has the option of deleting all the flow entries, if desired.

The controller configures and manages the switch, receives events from the switch, and sends packets out to the switch through this interface. Using the Open Flow protocol, a remote controller can add, update, or delete flow entries from the switch's flow table. That can happen reactively (in response to a packet arrival) or proactively. The Open Flow protocol can be viewed as one possible implementation of controller switch interactions (southbound interface), as it defines the communication between the switching hardware and a network controller. For security, Open Flow 1.3.x provides optional support for encrypted

TLS communication and a certificate exchange between the switches/controller(s); however, the exact implementation and certificate format is not currently specified. Also, fine-grained security options regarding scenarios with multiple controllers are outside the scope of the current specification, as there is no specific method to only grant partial access permissions to an authorized controller [6].

2.7 Traditional Networking VS SDN:

Table 2.1: Difference between Traditional and Software Defined Networking Types [12]

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

3. Modelling Virtual Private Network

3.1 Overview:

Mininet simulator and SDN ONOS1.13.2 controller are used in the thesis network models experiments. We will implement two different scenarios of SDN VPLS, in the first one we will use one Open Flow switch connected to the controller, configure 3 VPLS customers beginning with small number of sites per customer (two sites) then begin to add new sites for every VPLS customer and reed the total number of flow entries the controller use to forward data. In the next model we will repeat the experiment we did at the first model but using two Open Flow switches connecting to the controller.

In scenario one we use Mininet CLI to simulate the networks and use ONOS CLI to configure VPLS. In scenario two we will use python scripts to implement networks and use ONOS controller CLI to make VPLS configuration.

To implement the two network topologies scenarios and make the VPLS configuration and test them we follow the next steps:

1) We download and install Oracle **V**irtual **M**achine (**VM**) VirtualBox.
2) We install SDN Hub tutorial VM 64-bit with Docker as a virtual machine (Ubuntu 16.04 with Mininet and a default controller installed on it).
3) Download ONOS 1.13.2, use CLI to run it.
4) We implement the network topologies using CLI or python scripts and connect it to ONOS remote controller.
5) Test the connectivity between Mininet and the controller by using PING test (ONOS controller choose IP 127.0.0.1 and 6633 port number if it run in other port number it should change because Mininet is only connect to in this port, also the open flow feature must be activated on the controller).
6) Activate VPLS application on ONOS controller.
7) Use CLI to create VPLS, add interfaces to ONOS controller, and associate interfaces to it is VPLS.
8) Use ONOS CLI to check the configuration.
9) Use Mininet PING tests to make sure the network is work probably.

3.2 Models' Scenarios:

Models and their scenarios will be as flow:

3.2.1 Model 1 Scenario 1:

In this scenario we will use ONOS controller connecting to an Open Flow switch, add three VPLS customer everyone has two sites one host per site as shown in figure 3.1, make VPLS configuration and read the number of flows in the controller.

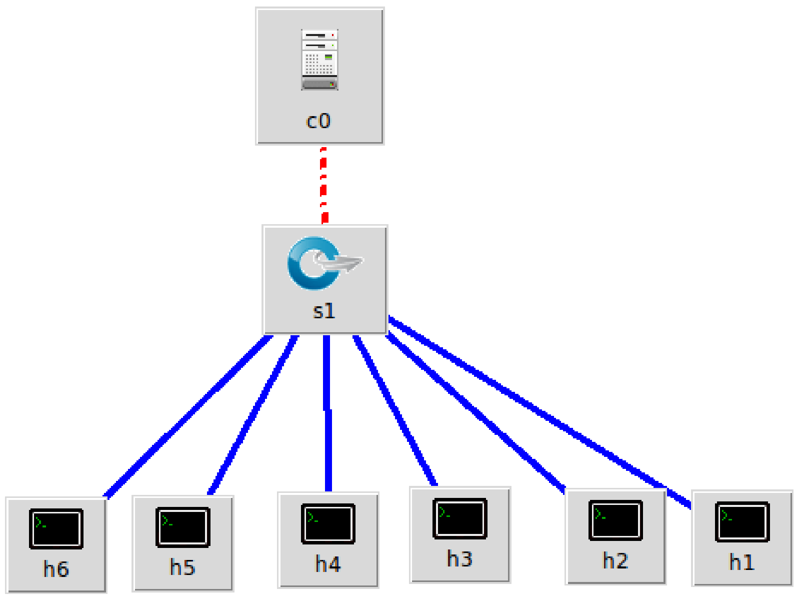


Figure 3.1: Model 1 Scenario 1

3.2.2 Model 1 Scenario 2:

In this scenario we will use ONOS controller connecting to an Open Flow switch, add three VPLS customer everyone has four sites one host per site as shown in figure 3.2, make VPLS configuration and read the number of flows in the controller.

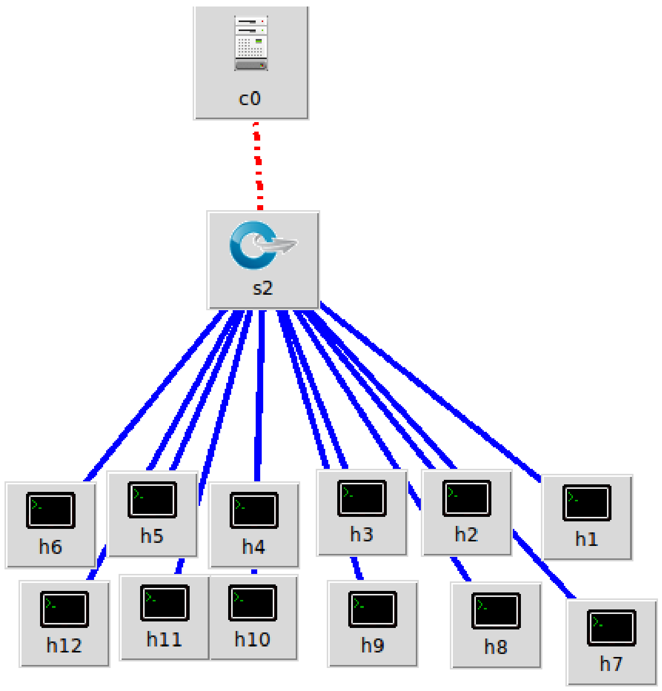


Figure 3.2: Model 1 Scenario 2

3.2.3 Model 1 Scenario 3:

In this scenario we will use ONOS controller connecting to an Open Flow switch, add three VPLS customer everyone has eight sites one host per site as shown in figure 3.3, make VPLS configuration and read the number of flows in the controller.

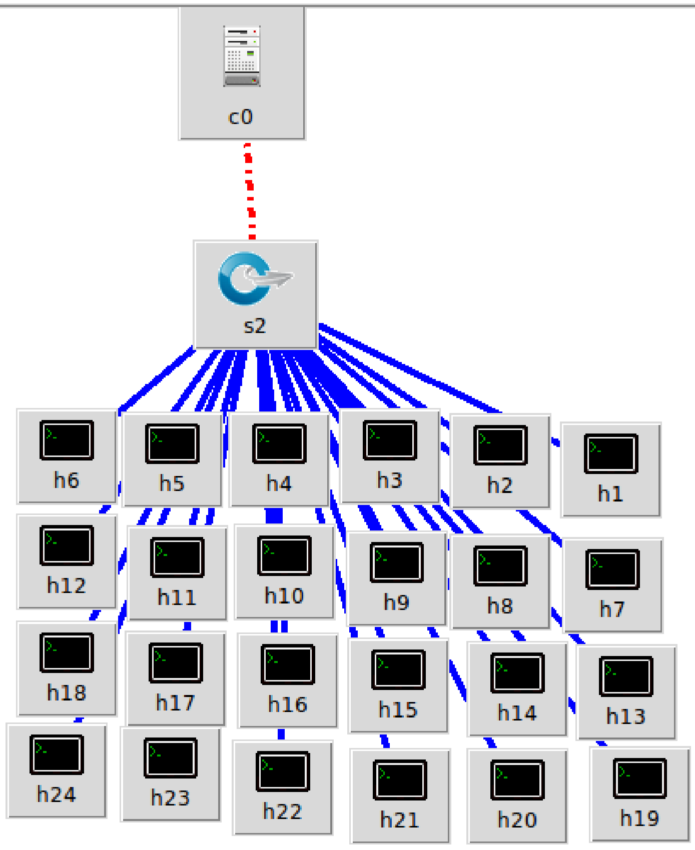


Figure 3.3: Model 1 Scenario 3

3.2.4 Model 2 Scenario 1:

In this scenario we will use ONOS controller connecting to an Open Flow switch, add three VPLS customer everyone has two sites one host per site as shown in figure 3.4, make VPLS configuration and read the number of flows in the controller.

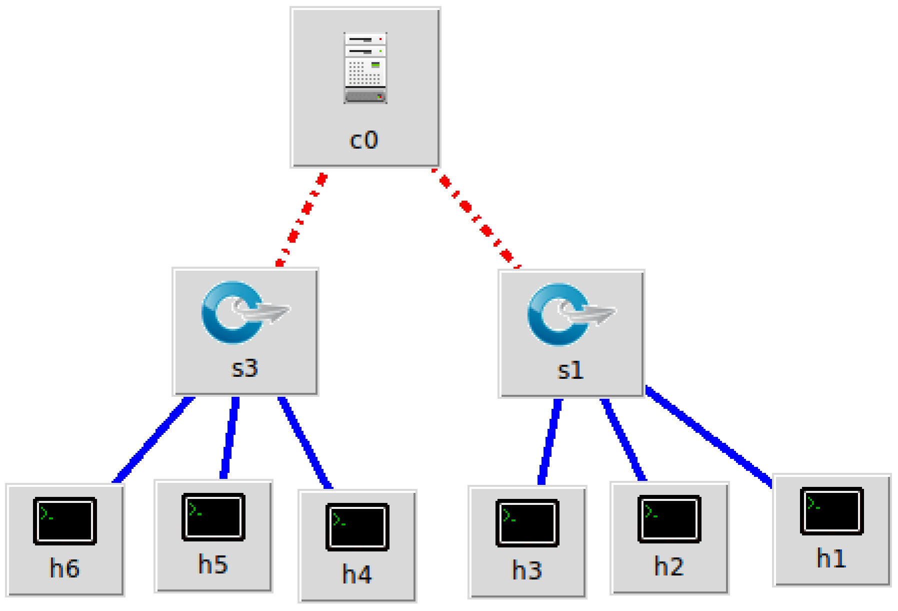


Figure 3.4: Model 2 Scenario 1

3.2.5 Model 2 Scenario 2:

In this scenario we will use ONOS controller connecting to an Open Flow switch, add three VPLS customer everyone has four sites one host per site as shown in figure 3.5, make VPLS configuration and read the number of flows in the controller.

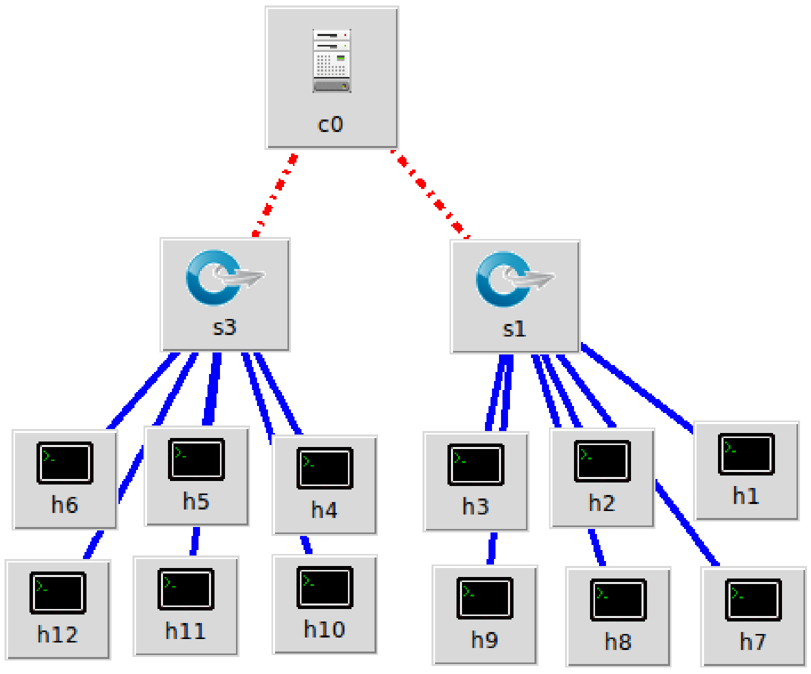


Figure 3.5: Model 2 Scenario 2

3.2.6 Model 2 Scenario 3:

In this scenario we will use ONOS controller connecting to an Open Flow switch, add three VPLS customer everyone has eight sites one host per site as shown in figure 3.6, make VPLS configuration and read the number of flows in the controller.

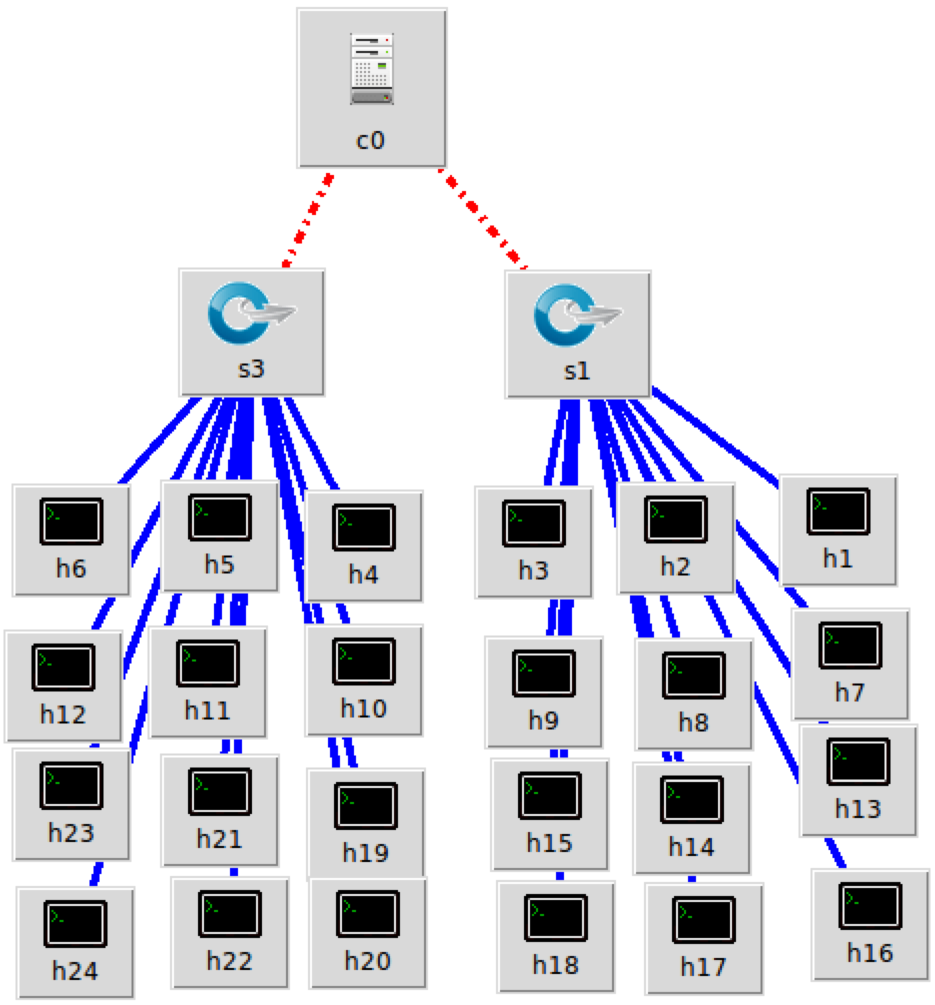


Figure 3.6: Model 2 Scenario 3

After implementing the three scenarios of each model the results will be retested on the table below:

Table 3.1: Simulation Parameters

Model/Scenario	**Number of switches**	**Number of VPNs**	**Number of sites/VPN**	**Flow table size**
1		3	2	
2		3	4	
3		3	8	

4. Simulation & Results

4.1 Simulation Description:

In order to VPLS establish connectivity two or more hosts, different things need to happen:

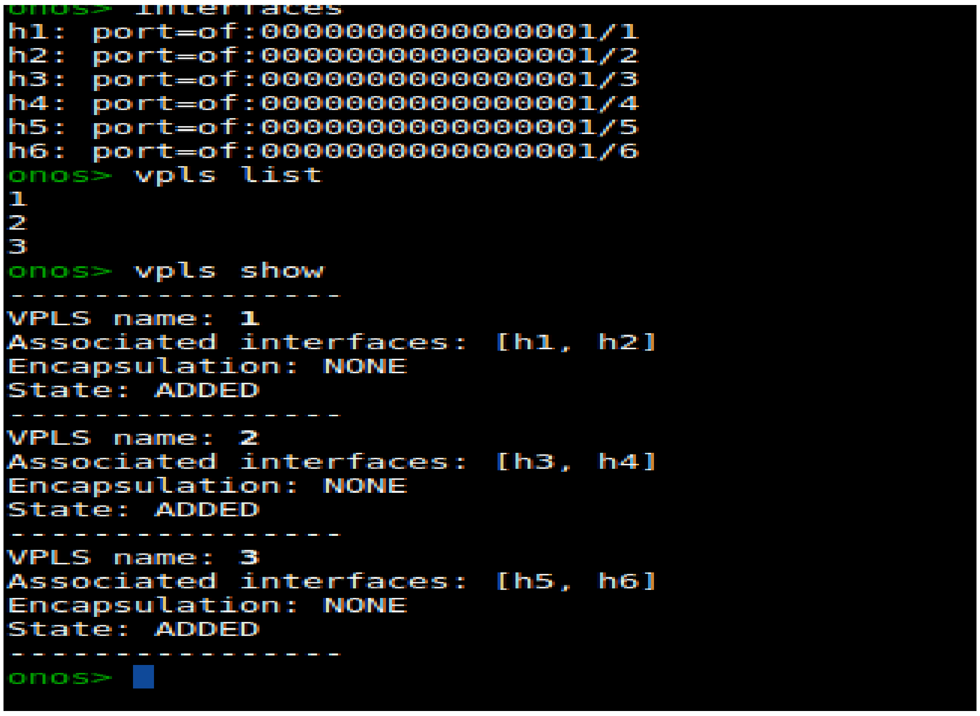
1) A VPLS needs to be defined.
2) At least two interfaces need to be configured in the ONOS interfaces configuration.
3) At least two interfaces need to be associated to the same VPLS.
4) At least two hosts need to be connected to the Open Flow network Hosts participating to the same VPLS can send in packets tagged with the same VLAN Ids, different VLAN IDs, and no VLAN Ids at all.

When conditions 1, 2 and 3 are satisfied, hosts attached to the VPLS will be able to send and receive broadcast traffic. This is needed to make sure that all hosts get discovered properly, before establishing unicast communication. When 4 gets satisfied-meaning that ONOS discovers at least two hosts of the same VPLS-unicast communication is established.

After implementing, configuring, testing the connectivity (by using PING tests) of the VPLS different models, the results will registered on a table to be analyzed and observed.

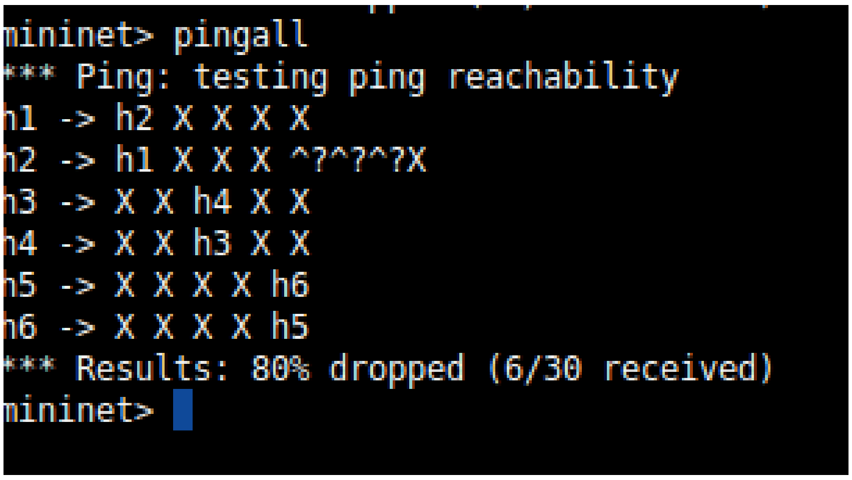
4.2 Results:

As we explained in the previous chapter we make our experiments on two scenarios with three models. Now we will showing the implementation, configuration, and the connectivity tests that we made for each model and finally showing the results.



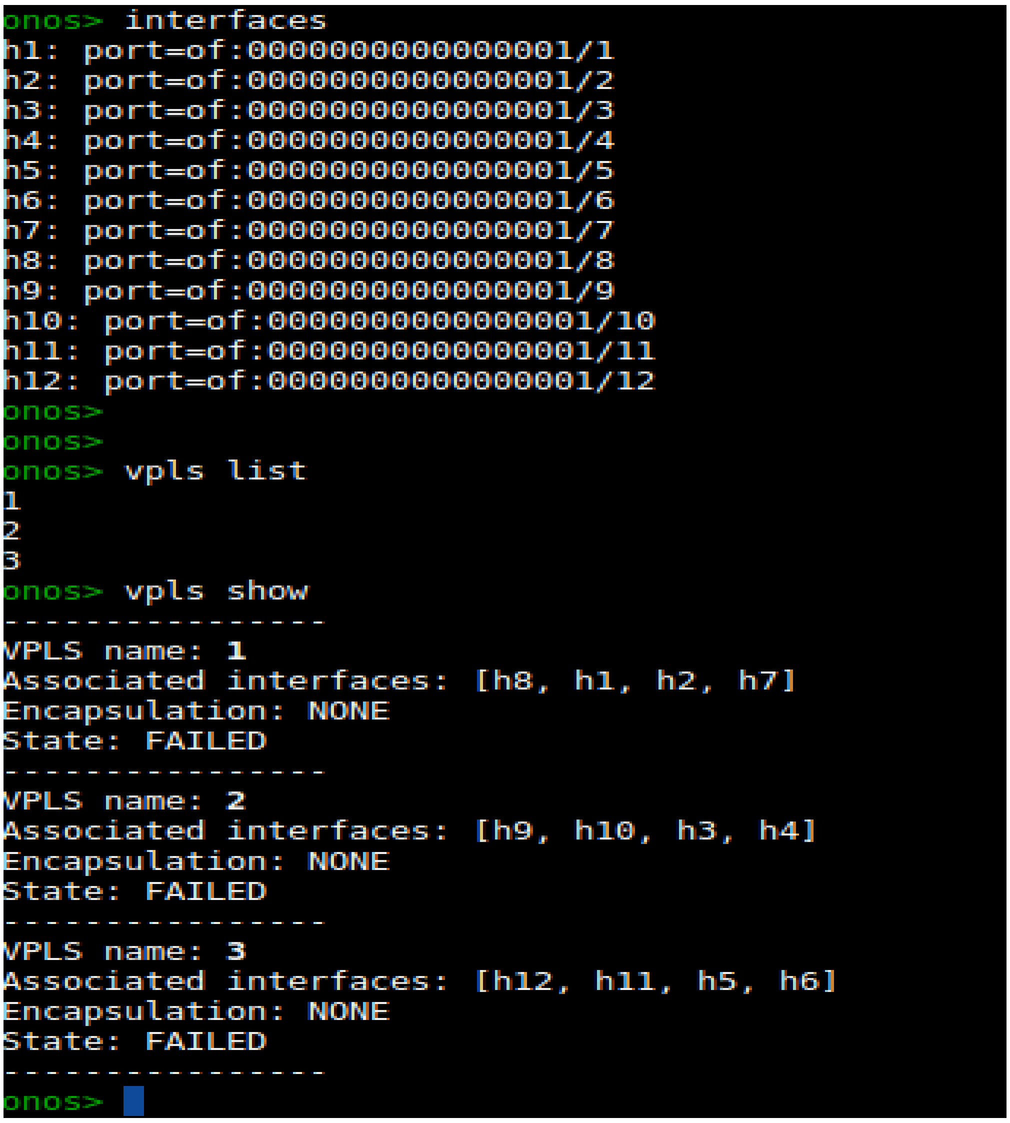
```
onos> interfaces
h1: port=of:0000000000000001/1
h2: port=of:0000000000000001/2
h3: port=of:0000000000000001/3
h4: port=of:0000000000000001/4
h5: port=of:0000000000000001/5
h6: port=of:0000000000000001/6
onos> vpls list
1
2
3
onos> vpls show
----------------
VPLS name: 1
Associated interfaces: [h1, h2]
Encapsulation: NONE
State: ADDED
----------------
VPLS name: 2
Associated interfaces: [h3, h4]
Encapsulation: NONE
State: ADDED
----------------
VPLS name: 3
Associated interfaces: [h5, h6]
Encapsulation: NONE
State: ADDED
----------------
onos>
```

Figure 4.1: Model 1 Scenario 1 Implementation



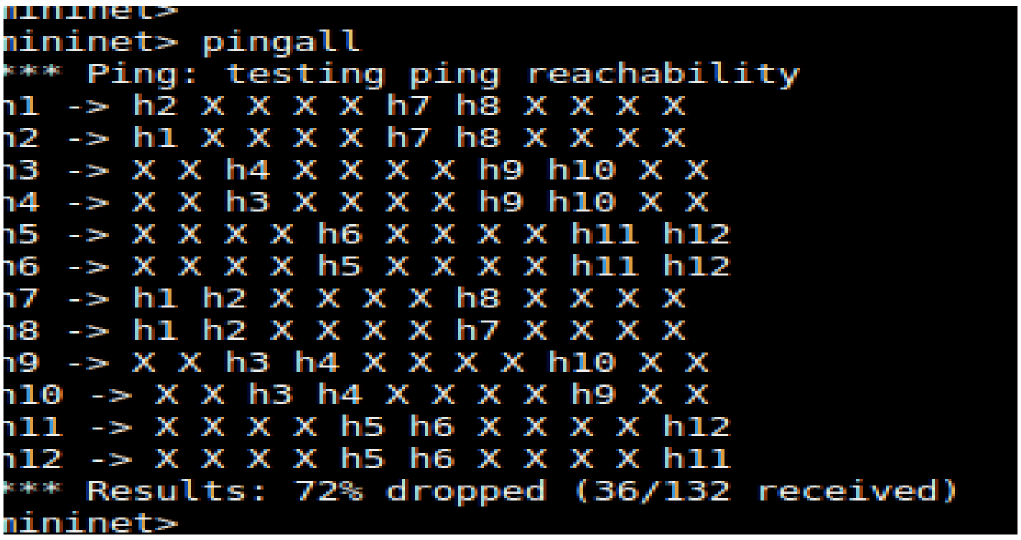
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X X X
h2 -> h1 X X X ^?^?^?X
h3 -> X X h4 X X
h4 -> X X h3 X X
h5 -> X X X X h6
h6 -> X X X X h5
*** Results: 80% dropped (6/30 received)
mininet>
```

Figure 4.2: Model 1 Scenario 1 PING Test



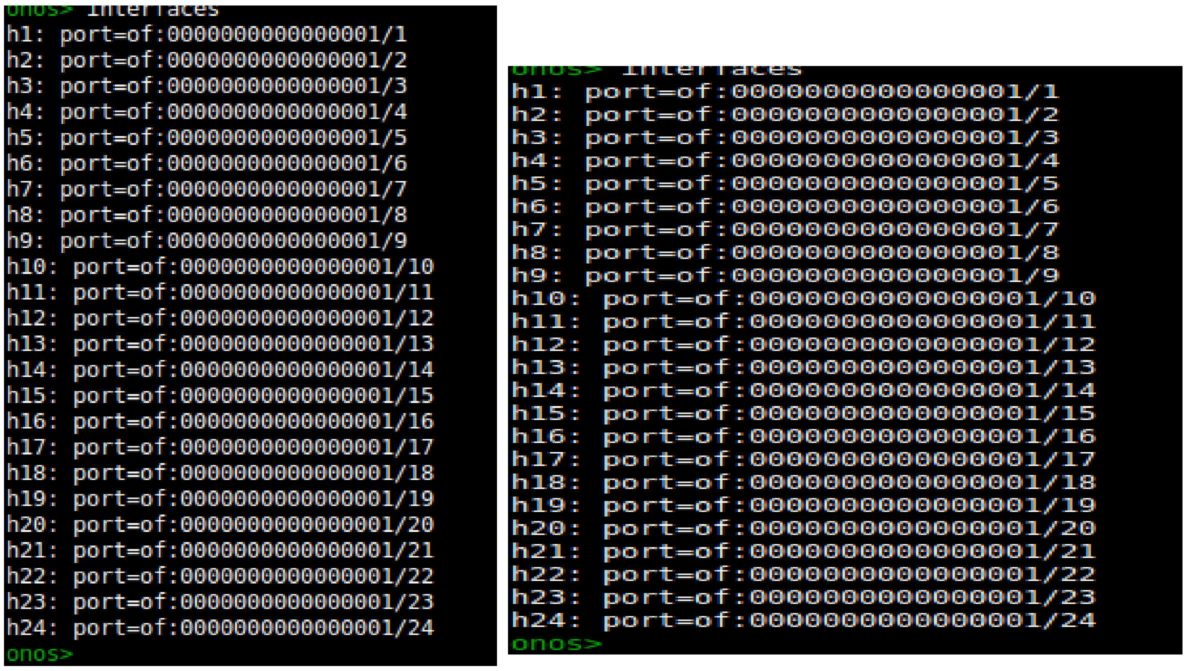
```
onos> interfaces
h1: port=of:0000000000000001/1
h2: port=of:0000000000000001/2
h3: port=of:0000000000000001/3
h4: port=of:0000000000000001/4
h5: port=of:0000000000000001/5
h6: port=of:0000000000000001/6
h7: port=of:0000000000000001/7
h8: port=of:0000000000000001/8
h9: port=of:0000000000000001/9
h10: port=of:0000000000000001/10
h11: port=of:0000000000000001/11
h12: port=of:0000000000000001/12
onos>
onos>
onos> vpls list
1
2
3
onos> vpls show
----------------
VPLS name: 1
Associated interfaces: [h8, h1, h2, h7]
Encapsulation: NONE
State: FAILED
----------------
VPLS name: 2
Associated interfaces: [h9, h10, h3, h4]
Encapsulation: NONE
State: FAILED
----------------
VPLS name: 3
Associated interfaces: [h12, h11, h5, h6]
Encapsulation: NONE
State: FAILED
----------------
onos>
```

Figure 4.3: Model 1 Scenario 2 Implementation



```
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X X X h7 h8 X X X X
h2 -> h1 X X X X h7 h8 X X X X
h3 -> X X h4 X X X X h9 h10 X X
h4 -> X X h3 X X X X h9 h10 X X
h5 -> X X X X h6 X X X X h11 h12
h6 -> X X X X h5 X X X X h11 h12
h7 -> h1 h2 X X X X h8 X X X X
h8 -> h1 h2 X X X X h7 X X X X
h9 -> X X h3 h4 X X X X h10 X X
h10 -> X X h3 h4 X X X X h9 X X
h11 -> X X X X h5 h6 X X X X h12
h12 -> X X X X h5 h6 X X X X h11
*** Results: 72% dropped (36/132 received)
mininet>
```

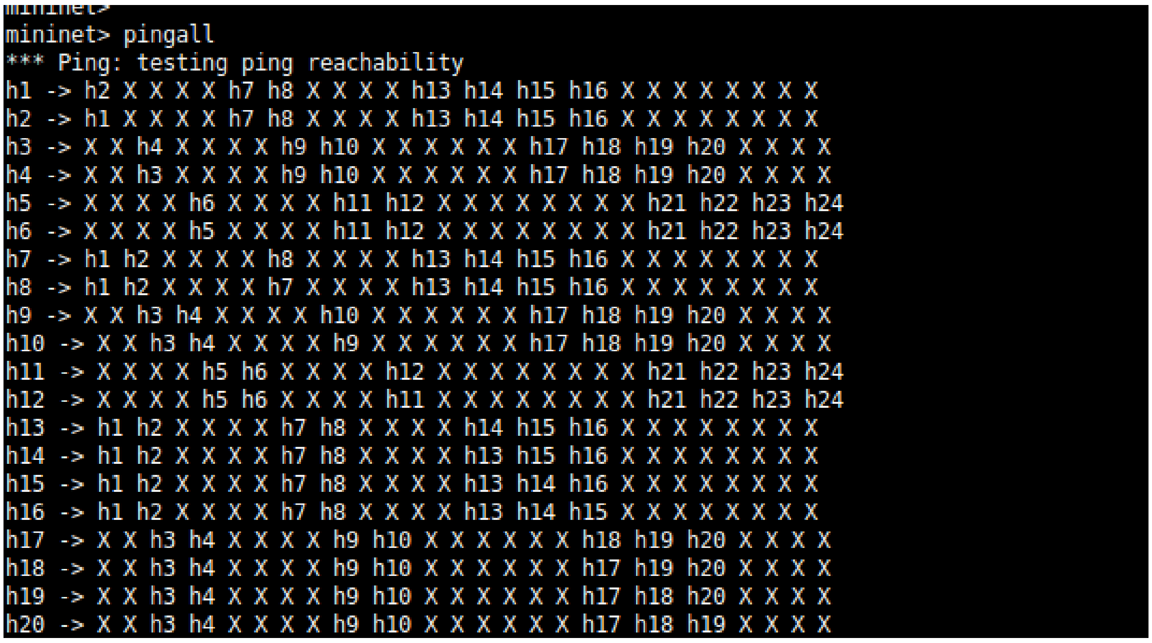
Figure 4.4: Model 1 Scenario 2 PING Test



```
onos> interfaces
h1: port=of:0000000000000001/1
h2: port=of:0000000000000001/2
h3: port=of:0000000000000001/3
h4: port=of:0000000000000001/4
h5: port=of:0000000000000001/5
h6: port=of:0000000000000001/6
h7: port=of:0000000000000001/7
h8: port=of:0000000000000001/8
h9: port=of:0000000000000001/9
h10: port=of:0000000000000001/10
h11: port=of:0000000000000001/11
h12: port=of:0000000000000001/12
h13: port=of:0000000000000001/13
h14: port=of:0000000000000001/14
h15: port=of:0000000000000001/15
h16: port=of:0000000000000001/16
h17: port=of:0000000000000001/17
h18: port=of:0000000000000001/18
h19: port=of:0000000000000001/19
h20: port=of:0000000000000001/20
h21: port=of:0000000000000001/21
h22: port=of:0000000000000001/22
h23: port=of:0000000000000001/23
h24: port=of:0000000000000001/24
onos>
```

```
onos> interfaces
h1: port=of:0000000000000001/1
h2: port=of:0000000000000001/2
h3: port=of:0000000000000001/3
h4: port=of:0000000000000001/4
h5: port=of:0000000000000001/5
h6: port=of:0000000000000001/6
h7: port=of:0000000000000001/7
h8: port=of:0000000000000001/8
h9: port=of:0000000000000001/9
h10: port=of:0000000000000001/10
h11: port=of:0000000000000001/11
h12: port=of:0000000000000001/12
h13: port=of:0000000000000001/13
h14: port=of:0000000000000001/14
h15: port=of:0000000000000001/15
h16: port=of:0000000000000001/16
h17: port=of:0000000000000001/17
h18: port=of:0000000000000001/18
h19: port=of:0000000000000001/19
h20: port=of:0000000000000001/20
h21: port=of:0000000000000001/21
h22: port=of:0000000000000001/22
h23: port=of:0000000000000001/23
h24: port=of:0000000000000001/24
onos>
```

Figure 4.5: Model 1 Scenario 3 Implementation

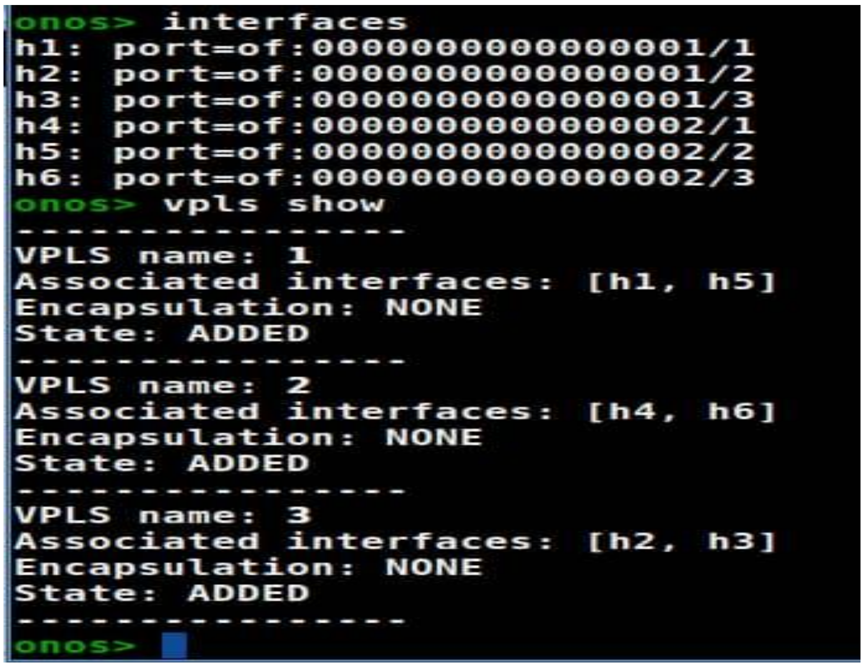


```
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X X X h7 h8 X X X X h13 h14 h15 h16 X X X X X X X X
h2 -> h1 X X X X h7 h8 X X X X h13 h14 h15 h16 X X X X X X X X
h3 -> X X h4 X X X X h9 h10 X X X X X X h17 h18 h19 h20 X X X X
h4 -> X X h3 X X X X h9 h10 X X X X X X h17 h18 h19 h20 X X X X
h5 -> X X X X h6 X X X X h11 h12 X X X X X X X X h21 h22 h23 h24
h6 -> X X X X h5 X X X X h11 h12 X X X X X X X X h21 h22 h23 h24
h7 -> h1 h2 X X X X h8 X X X X h13 h14 h15 h16 X X X X X X X X
h8 -> h1 h2 X X X X h7 X X X X h13 h14 h15 h16 X X X X X X X X
h9 -> X X h3 h4 X X X X h10 X X X X X X h17 h18 h19 h20 X X X X
h10 -> X X h3 h4 X X X X h9 X X X X X X h17 h18 h19 h20 X X X X
h11 -> X X X X h5 h6 X X X X h12 X X X X X X X X h21 h22 h23 h24
h12 -> X X X X h5 h6 X X X X h11 X X X X X X X X h21 h22 h23 h24
h13 -> h1 h2 X X X X h7 h8 X X X X h14 h15 h16 X X X X X X X X
h14 -> h1 h2 X X X X h7 h8 X X X X h13 h15 h16 X X X X X X X X
h15 -> h1 h2 X X X X h7 h8 X X X X h13 h14 h16 X X X X X X X X
h16 -> h1 h2 X X X X h7 h8 X X X X h13 h14 h15 X X X X X X X X
h17 -> X X h3 h4 X X X X h9 h10 X X X X X X h18 h19 h20 X X X X
h18 -> X X h3 h4 X X X X h9 h10 X X X X X X h17 h19 h20 X X X X
h19 -> X X h3 h4 X X X X h9 h10 X X X X X X h17 h18 h20 X X X X
h20 -> X X h3 h4 X X X X h9 h10 X X X X X X h17 h18 h19 X X X X
```

Figure 4.6: Model 1 Scenario 3 PING Test

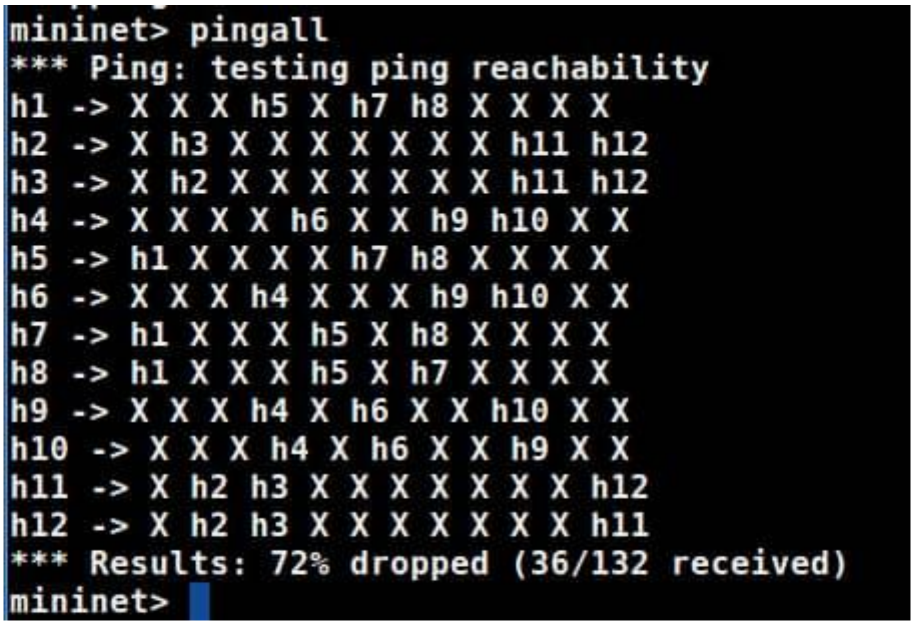
Table 4.1: Model One Results

Scenario	Number of switches	Number of VPNs	Number of sites/VPN	Flow table size
1	1	3	2	16
2	1	3	4	52
3	1	3	8	150



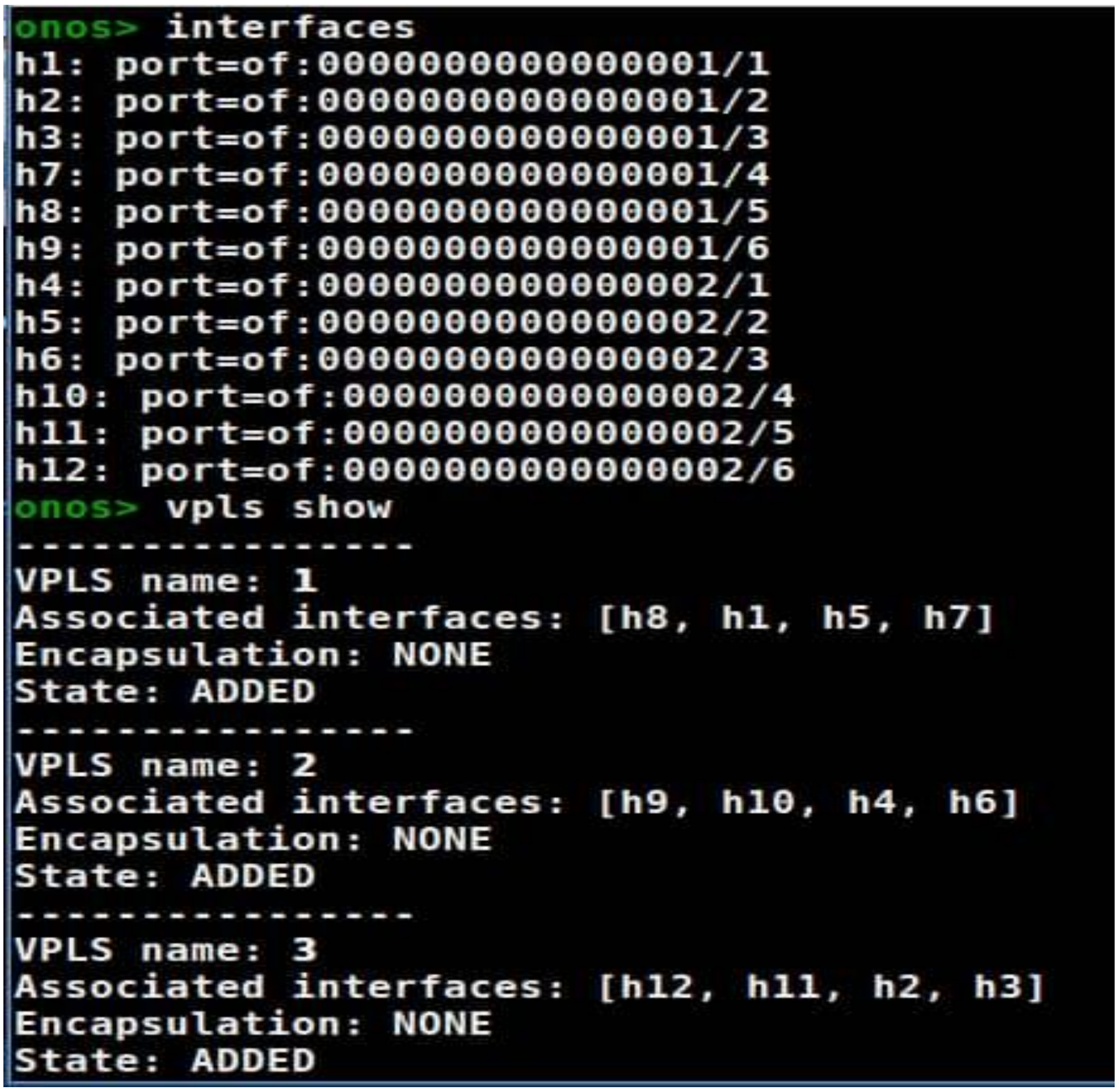
```
onos> interfaces
h1: port=of:0000000000000001/1
h2: port=of:0000000000000001/2
h3: port=of:0000000000000001/3
h4: port=of:0000000000000002/1
h5: port=of:0000000000000002/2
h6: port=of:0000000000000002/3
onos> vpls show
----------------
VPLS name: 1
Associated interfaces: [h1, h5]
Encapsulation: NONE
State: ADDED
----------------
VPLS name: 2
Associated interfaces: [h4, h6]
Encapsulation: NONE
State: ADDED
----------------
VPLS name: 3
Associated interfaces: [h2, h3]
Encapsulation: NONE
State: ADDED
----------------
onos>
```

Figure 4.7: Model 2 Scenario 1 Implementation



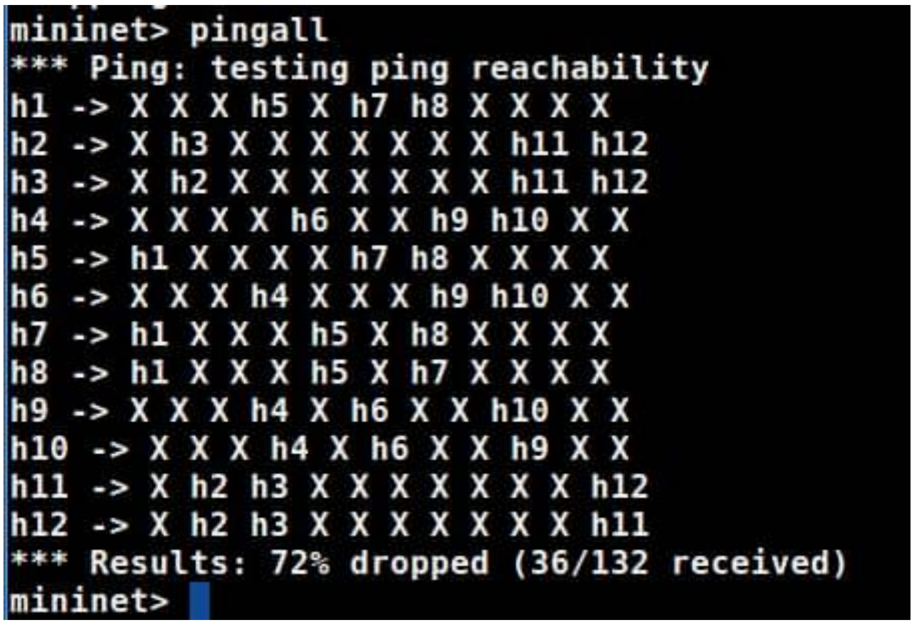
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X h5 X h7 h8 X X X X
h2 -> X h3 X X X X X X X h11 h12
h3 -> X h2 X X X X X X X h11 h12
h4 -> X X X X h6 X X h9 h10 X X
h5 -> h1 X X X X h7 h8 X X X X
h6 -> X X X h4 X X X h9 h10 X X
h7 -> h1 X X X h5 X h8 X X X X
h8 -> h1 X X X h5 X h7 X X X X
h9 -> X X X h4 X h6 X X h10 X X
h10 -> X X X h4 X h6 X X h9 X X
h11 -> X h2 h3 X X X X X X X h12
h12 -> X h2 h3 X X X X X X X h11
*** Results: 72% dropped (36/132 received)
mininet>
```

Figure 4.8: Model 2 Scenario 1 PING Test



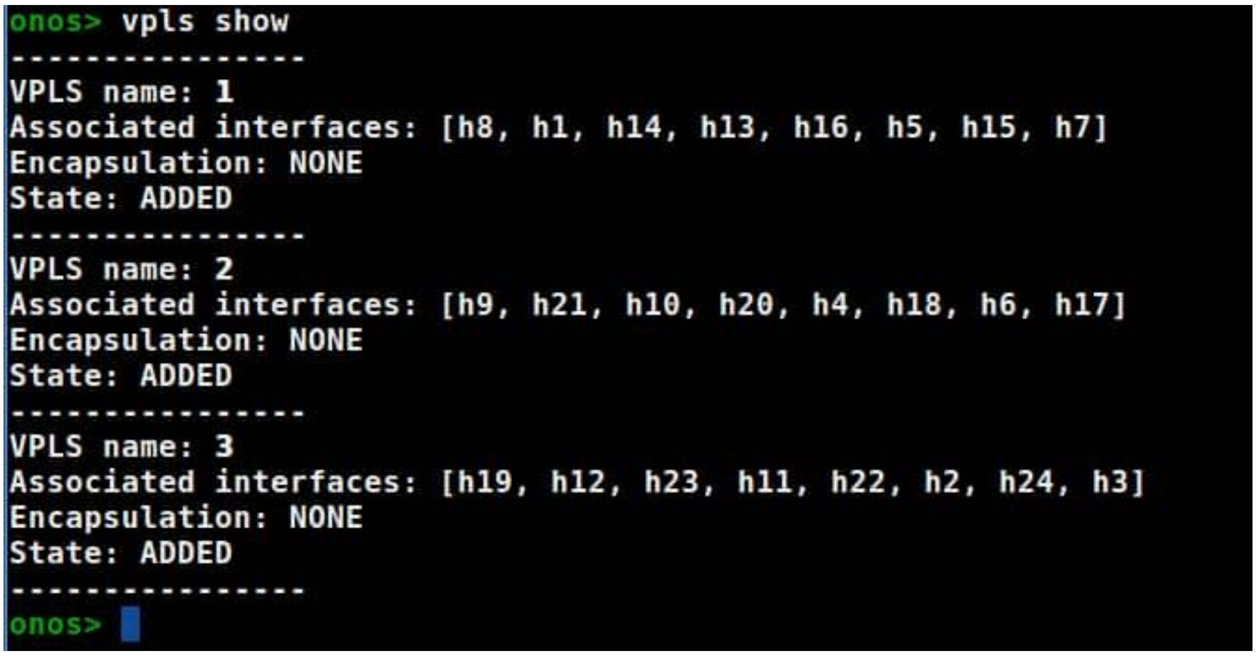
```
onos> interfaces
h1: port=of:0000000000000001/1
h2: port=of:0000000000000001/2
h3: port=of:0000000000000001/3
h7: port=of:0000000000000001/4
h8: port=of:0000000000000001/5
h9: port=of:0000000000000001/6
h4: port=of:0000000000000002/1
h5: port=of:0000000000000002/2
h6: port=of:0000000000000002/3
h10: port=of:0000000000000002/4
h11: port=of:0000000000000002/5
h12: port=of:0000000000000002/6
onos> vpls show
----------------
VPLS name: 1
Associated interfaces: [h8, h1, h5, h7]
Encapsulation: NONE
State: ADDED
----------------
VPLS name: 2
Associated interfaces: [h9, h10, h4, h6]
Encapsulation: NONE
State: ADDED
----------------
VPLS name: 3
Associated interfaces: [h12, h11, h2, h3]
Encapsulation: NONE
State: ADDED
```

Figure 4.9: Model 2 Scenario 2 Implementation



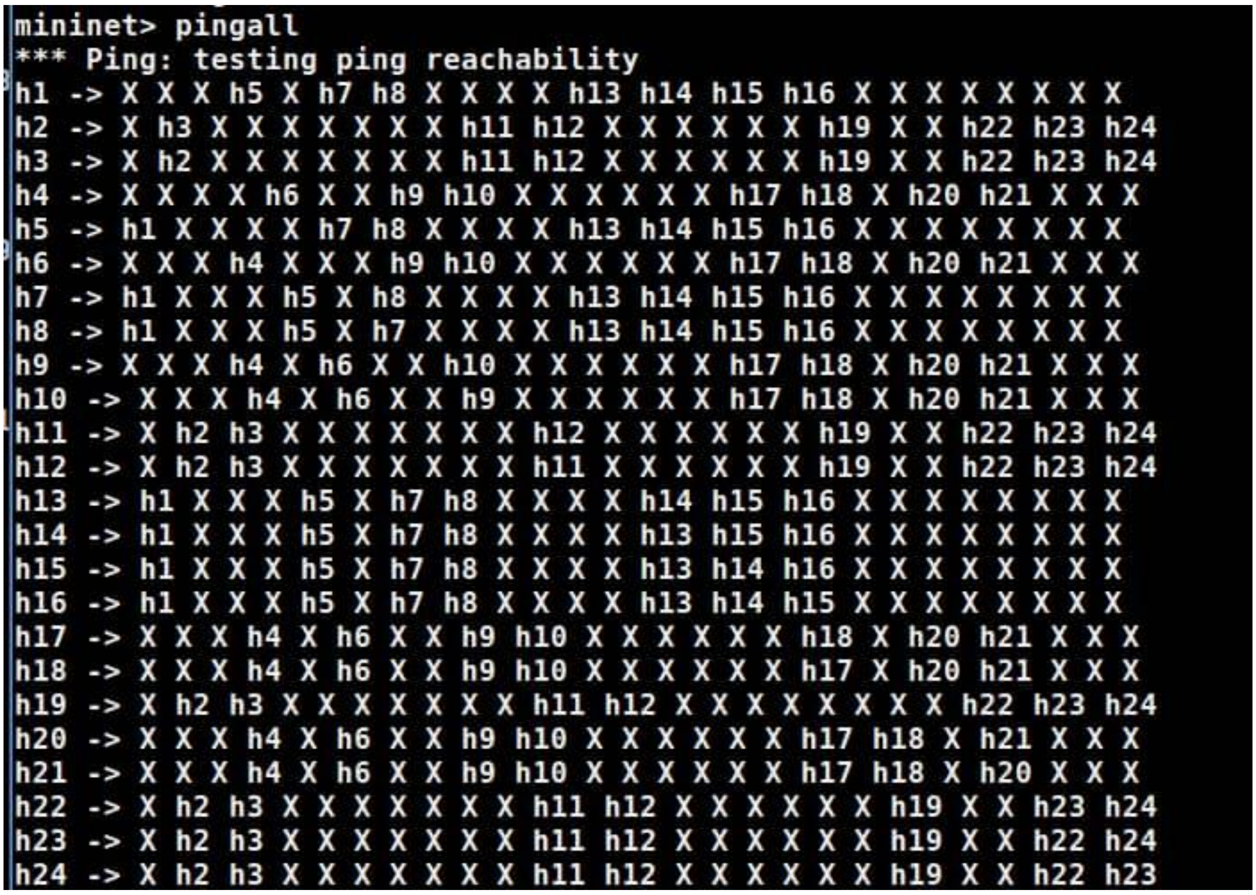
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X h5 X h7 h8 X X X X
h2 -> X h3 X X X X X X X h11 h12
h3 -> X h2 X X X X X X X h11 h12
h4 -> X X X X h6 X X h9 h10 X X
h5 -> h1 X X X X h7 h8 X X X X
h6 -> X X X h4 X X X h9 h10 X X
h7 -> h1 X X X h5 X h8 X X X X
h8 -> h1 X X X h5 X h7 X X X X
h9 -> X X X h4 X h6 X X h10 X X
h10 -> X X X h4 X h6 X X h9 X X
h11 -> X h2 h3 X X X X X X X h12
h12 -> X h2 h3 X X X X X X X h11
*** Results: 72% dropped (36/132 received)
mininet>
```

Figure 4.10: Model 2 Scenario 2 PING Test



```
onos> vpls show
----------------
VPLS name: 1
Associated interfaces: [h8, h1, h14, h13, h16, h5, h15, h7]
Encapsulation: NONE
State: ADDED
----------------
VPLS name: 2
Associated interfaces: [h9, h21, h10, h20, h4, h18, h6, h17]
Encapsulation: NONE
State: ADDED
----------------
VPLS name: 3
Associated interfaces: [h19, h12, h23, h11, h22, h2, h24, h3]
Encapsulation: NONE
State: ADDED
----------------
onos>
```

Figure 4.11: Model 2 Scenario 3 Implementation



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X h5 X h7 h8 X X X X h13 h14 h15 h16 X X X X X X X X
h2 -> X h3 X X X X X X X h11 h12 X X X X X X h19 X X h22 h23 h24
h3 -> X h2 X X X X X X X h11 h12 X X X X X X h19 X X h22 h23 h24
h4 -> X X X X h6 X X h9 h10 X X X X X X h17 h18 X h20 h21 X X X
h5 -> h1 X X X X h7 h8 X X X X h13 h14 h15 h16 X X X X X X X X
h6 -> X X X h4 X X X h9 h10 X X X X X X h17 h18 X h20 h21 X X X
h7 -> h1 X X X h5 X h8 X X X X h13 h14 h15 h16 X X X X X X X X
h8 -> h1 X X X h5 X h7 X X X X h13 h14 h15 h16 X X X X X X X X
h9 -> X X X h4 X h6 X X h10 X X X X X X h17 h18 X h20 h21 X X X
h10 -> X X X h4 X h6 X X h9 X X X X X X h17 h18 X h20 h21 X X X
h11 -> X h2 h3 X X X X X X X h12 X X X X X X h19 X X h22 h23 h24
h12 -> X h2 h3 X X X X X X X h11 X X X X X X h19 X X h22 h23 h24
h13 -> h1 X X X h5 X h7 h8 X X X X h14 h15 h16 X X X X X X X X
h14 -> h1 X X X h5 X h7 h8 X X X X h13 h15 h16 X X X X X X X X
h15 -> h1 X X X h5 X h7 h8 X X X X h13 h14 h16 X X X X X X X X
h16 -> h1 X X X h5 X h7 h8 X X X X h13 h14 h15 X X X X X X X X
h17 -> X X X h4 X h6 X X h9 h10 X X X X X X h18 X h20 h21 X X X
h18 -> X X X h4 X h6 X X h9 h10 X X X X X X h17 X h20 h21 X X X
h19 -> X h2 h3 X X X X X X X h11 h12 X X X X X X X X h22 h23 h24
h20 -> X X X h4 X h6 X X h9 h10 X X X X X X h17 h18 X h21 X X X
h21 -> X X X h4 X h6 X X h9 h10 X X X X X X h17 h18 X h20 X X X
h22 -> X h2 h3 X X X X X X X h11 h12 X X X X X X h19 X X h23 h24
h23 -> X h2 h3 X X X X X X X h11 h12 X X X X X X h19 X X h22 h24
h24 -> X h2 h3 X X X X X X X h11 h12 X X X X X X h19 X X h22 h23
```

Figure 4.12: Model 2 Scenario 3 PING Test

Table 4.2: Model Two Results

Scenario	Number of switches	Number of VPNs	Number of sites/VPN	Flow table size
1	2	3	2	27
2	2	3	4	74
3	2	3	8	192

5. Results & Discussion

5.1 Discussion:

Althought we use in our study only one switch as in the first scenario and two switches as in the second one, and only three VPLSs with maximum eight sites per every one witch is very simple part of WAN network it seem to be an ISP having two PEs routers,and the customer sites represented by only one host and there is no Qos configured or even routing protocols or security polices like what is happened in reaility the site may be ahuge data center with large number of servers with very complicated security polices Qos and more than one routing protocol, we opsarved that from first scenario the total number of flowes increse more than three times when the the sites scales to daboule.The flowes also increse (about daboule)when the number of Open Flow switches dabouled**.**

The emerging paradigm of SDN promises to simplify network management,and build reialable and scaleable network.in this study we opsarve that network management is mutch eyseir because it centrlized in the controller.but as we disscused the scalabilty is aserios issue for SDN because as anetwork scale the controller must be deail with large amount of flowes concurently witch need repotness at booth software and hardware of the controllers.

6. Conclusion & Recommendation

6.1 Conclusion:

In this study, we extended the observation of the reliability and scalability issues to an SDN that is beyond the controllers. This calls for attention to such issues of controller performance and controllers clustering and way that the controllers must be connected when VPLS customers' sites distributed among more than one controller.

6.2 Recommendation:

The next steps would be developing the methods and metrics to evaluate SDN controllers' performance and SDN WAN design and SDN related frameworks, and to seek and establish general guidelines in designing and implementing the frameworks without jeopardizing the scalability in SDN.

REFERENCES

[1] Behzad Mirkhanzadeh, Naeim Taheri, Siavash Khorsandi, "SDxVPN: A Software-Defined Solution for VPN Service Providers" Network Operations and Management Symposium (NOMS) February 2016.

[2] Shuhao Liu and Baochun Li "On Scaling Software-Defined Networking in Wide-Area Networks" June 2015 TSINGHUA SCIENCE AND TECHNOLOGY.

[3] Barath Raghavan, Teemu Koponen, Ali Ghodsi, "Software-Defined Internet Architecture: Decoupling Architecture from Infrastructure" Proceedings of the 11th ACM Workshop on Hot Topics in Networks.

[4] Nick Feamster, Jennifer Rexford, Ellen Zegura, "The Road to SDN: An Intellectual Historyof Programmable Networks" ACM SIGCOMM Computer Communication Review April 2014.

[5] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, V. Maglaris, " Combining OpenFlow and sFlow for an effective and scalable Anomaly Detection and Mitigation mechanism on SDN Environments" Computer Networks April 2014.

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

[7] GR Wright,WR Stetevens"TCP IP",1995

[8] Z Fu,SF Wu,H Huang,K Loh,F Gong "IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution"Springer,20017

[9] Understanding the virtual private LAN Service, Article available at : https://www.juniper.net/documentation/en_US/releaseindependent/nce/topics/concept/vpls-understanding.html

[10] LUC GHEIN "MPLS fundamentals" November 21, 2006.

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

[12]. https://www.opendaylight.org/ecosystem-solutions

[13] Article available at : http://www.projectfloodlight.org/documentation/

[14] Y Zhao, "On Controller Performance in Software-Defined Networks" IEEEXplor,2015

APPENDICES

A. Paython Script for Model 2 Scenario 1:

```
from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info
def treeTopo():
    net = Mininet( controller=RemoteController )
    info( '*** Adding controller\n' )
    net.addController('c0')
    info( '*** Adding hosts\n' )
    h1 = net.addHost( 'h1')
    h2 = net.addHost( 'h2')
    h3 = net.addHost( 'h3')
    h4 = net.addHost( 'h4')
    h5 = net.addHost( 'h5')
    h6 = net.addHost( 'h6')
    info( '*** Adding switches\n' )
    s1 = net.addSwitch( 's1' )
    s2 = net.addSwitch( 's2' )
    info( '*** Creating links\n' )
    net.addLink( h1, s1 )
    net.addLink( h2, s1 )
    net.addLink( h3, s1 )
    net.addLink( h4, s2 )
    net.addLink( h5, s2 )
    net.addLink( h6, s2 )
    net.addLink( s1, s2 )
    info( '*** Starting network\n')
    net.start()
        info( '*** Running CLI\n' )
    CLI( net )
    info( '*** Stopping network' )
    net.stop()
if __name__ == '__main__':
    setLogLevel( 'info' )
    treeTopo()
```

B. Paython Script for Model 2 Scenario 2:

```
from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info
def treeTopo():
    net = Mininet( controller=RemoteController )
    info( '*** Adding controller\n' )
    net.addController('c0')
    info( '*** Adding hosts\n' )
    h1 = net.addHost( 'h1')
    h2 = net.addHost( 'h2')
    h3 = net.addHost( 'h3')
    h4 = net.addHost( 'h4')
    h5 = net.addHost( 'h5'
    h6 = net.addHost( 'h6')
    h7 = net.addHost( 'h7')
    h8 = net.addHost( 'h8')
    h9 = net.addHost( 'h9')
    h10 = net.addHost( 'h10')
    h11 = net.addHost( 'h11')
    h12 = net.addHost( 'h12')
    info( '*** Adding switches\n' )
    s1 = net.addSwitch( 's1' )
    s2 = net.addSwitch( 's2' )
    info( '*** Creating links\n' )
    net.addLink( h1, s1 )
    net.addLink( h2, s1 )
    net.addLink( h3, s1 )
    net.addLink( h4, s2 )
    net.addLink( h5, s2 )
    net.addLink( h6, s2 )
    net.addLink( h7, s1 )
    net.addLink( h8, s1 )
    net.addLink( h9, s1 )
    net.addLink( h10, s2 )
    net.addLink( h11, s2 )
    net.addLink( h12, s2 )
    net.addLink( s1, s2 )
    info( '*** Starting network\n')
    net.start()
    info( '*** Running CLI\n' )
    CLI( net )
    info( '*** Stopping network' )
    net.stop()
if __name__ == '__main__':
    setLogLevel( 'info' )
    treeTopo()
```

C. Paython Script for Model 2 Scenario 3:

```
from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info
def treeTopo():
    net = Mininet( controller=RemoteController )
    info( '*** Adding controller\n' )
    net.addController('c0')
    info( '*** Adding hosts\n' )
    h1 = net.addHost( 'h1')
    h2 = net.addHost( 'h2')
    h3 = net.addHost( 'h3')
    h4 = net.addHost( 'h4')
    h5 = net.addHost( 'h5')
    h6 = net.addHost( 'h6')
    h7 = net.addHost( 'h7')
    h8 = net.addHost( 'h8')
    h9 = net.addHost( 'h9')
    h10 = net.addHost( 'h10')
    h11 = net.addHost( 'h11')
    h12 = net.addHost( 'h12')
    h13 = net.addHost( 'h13')
    h14 = net.addHost( 'h14')
    h15 = net.addHost( 'h15')
    h16 = net.addHost( 'h16')
    h17 = net.addHost( 'h17')
    h18 = net.addHost( 'h18')
    h19 = net.addHost( 'h19')
    h20 = net.addHost( 'h20')
    h21 = net.addHost( 'h21')
    h22 = net.addHost( 'h22')
    h23 = net.addHost( 'h23')
    h24 = net.addHost( 'h24')
```

```
info( '*** Adding switches\n' )
s1 = net.addSwitch( 's1' )
s2 = net.addSwitch( 's2' )
info( '*** Creating links\n' )
net.addLink( h1, s1 )
net.addLink( h2, s1 )
net.addLink( h3, s1 )
net.addLink( h4, s2 )
net.addLink( h5, s2 )
net.addLink( h6, s2 )
net.addLink( h7, s1 )
net.addLink( h8, s1 )
net.addLink( h9, s1 )
net.addLink( h10, s2 )
net.addLink( h11, s2 )
net.addLink( h12, s2 )
net.addLink( h13, s1 )
net.addLink( h14, s1 )
net.addLink( h15, s1 )
net.addLink( h16, s2 )
net.addLink( h17, s2 )
net.addLink( h18, s2 )
net.addLink( h19, s1 )
net.addLink( h20, s1 )
net.addLink( h21, s2 )
net.addLink( h22, s2 )
net.addLink( h23, s2 )
net.addLink( h24, s2 )
net.addLink( s1, s2 )
info( '*** Starting network\n')
```

```
    net.start()

    info( '*** Running CLI\n' )
    CLI( net )
    info( '*** Stopping network' )
    net.stop()
if __name__ == '__main__':
    setLogLevel( 'info' )
    treeTopo()
```

D. Ubuntu CLI:

ubuntu CLI	**Description**
ubuntu@sdnhubvm:~[06:09]$ cd Desktop/onos-1.13.2/bin	Go to directory: Desktop/onos-1.13.2/bin
ubuntu@sdnhubvm:~/Desktop/onos-1.13.2/bin[06:09]$ sudo su	running as root user
root@sdnhubvm:/home/ubuntu/Desktop/onos-1.13.2/bin# ./service-start	RUN ONOS controller
root@sdnhubvm:/home/ubuntu/Desktop/onos-1.13.2/bin# sudo mn --topo single,6 --controller remote	open mininet immplement anetwork topology with one switch and six hosts connceted to acntrollet with IP127.0.0.1 in port number 6633

E. ONOS CLI:

ONOS CLI	Description
onos> app activate org.onosproject.openflow	activate open flow application
onos> cfg set org.onosproject.openflow.controller.impl.OpenFlowControllerImpl openflowPorts 6633	change controller port to be 6633
onos> app activate org.onosproject.vpls	activate VPLS application
onos> create vpls 1	create VPLS and name it 1
onos> interface-add of:0000000000000001/1 h1	connect h1 to port 1 in the controller
onos> vpls add-if 1 h1	associate h1 to VPLS 1
onos> vpls list	list all configured VPLSs
onos> vpls show	Show all configured VPLSs in details

F. Mininet CLI:

Mininet CLI	Description
pingall	ping test from every host to all other hosts in the network