Benha University

Faculty of Engineering at Shoubra

Electronics and Communications Department

Software Defined Networks:
Emulation, Implementation and Load Balancing

Supervised by:

Assoc. Prof. Dr. Adly Shahat Tag Eldien

Submitted by:

- **Abdullah Mohamed Ragab**
- **Osama Said Mahmoud**
- **Ahmed Reda Ibrahim**
- **Ahmed Mohamed Sobhy**
- **Ahmed Yasser Abdelhamid**
- **Ahmed Yasser Helmy**
- **Shrouk Mohamed Abdelmenam**
- **Salma Sameh Mohamed**

Egypt, 2020

Abstract

Traditional IP networks are complex and very hard to manage. It is both difficult to configure the network according to predefined policies, and to reconfigure it to respond to faults, load and changes. To make matters even more difficult, current networks are also vertically integrated: the control and data planes are bundled together. Software-Defined Networking (SDN) emerging paradigm that promises to change this state of affairs, by breaking vertical integration, separating the network's control logic from the underlying routers and switches, promoting (logical) centralization of network control, and introducing the ability to program the network. Through the Controller, network administrators can quickly and easily make and push out decisions on how the underlying systems (switches, routers) of the forwarding plane will handle the traffic. The most common protocol used in SDN networks to facilitate the communication between the Controller (called the Southbound API) and the switches is currently Open-Flow.

In this project, our purpose is to shed light on SDN related issues and give insight into the architecture of this revolutionary network model through the following five phases. We have used virtual phase that combined between traditional network and SDN. Then, we have tested and used two open-source controllers: POX and Ryu. We have provided implementation of the well-known traditional network STP protocol on Ryu controllers. In addition, an implementation of a Load Balance Algorithm has been introduced on Pox and another component. We have mentioned HPE VAN as a commercial controller in a simple topology. Mininet, a network emulator, has been used for the mentioned scenarios. Finally, we have proposed a framework for connecting traditional networks with SDN technology through the combination of SDN network, cisco routers and switches.

Acknowledgments

First and foremost, we wish to express our deepest sense of gratitude goes to our supervisor

Assoc. Prof. Dr. Adly Shahat Tag Eldien

For his support, guidance and assistance in completing our project. We appreciate his efforts, instructions, and suggestions during this work.

Content

Chapter 6

Mininet

Chapter 7

SDN Controllers

References

Conclusion and Future Work

LIST OF FIGURES

Chapter 1

Introduction

Chapter 2

Traditional Network

Chapter 3

SDN

Chapter 4

Traditional Network vs. SDN

Chapter 5

OpenFlow

Chapter 6

Mininet

Chapter 7

SDN Controllers

Chapter 8

Tools

Chapter 9

Load balancing Algorithms

LIST OF TABLES

Chapter 2

Traditional Network

Chapter 4

Traditional Network vs. SDN

Chapter 7

SDN Controllers

LIST OF ABBREVIATIONS

SDN	Software-Defined Networking
CP	Control Plane
SLB	Server Load Balancing
URL	Uniform Resource Locator
LAN	Local Area Network
WAN	Wide Area Network
OSI	Open System Interconnection
MAC	Medium Access Control
VLAN	Virtual LAN
QoS	Quality-of-Service
VM	virtual machine
ACL	Access Lists
DP	Data Path
ODL	OpenDayLight
API	Application programming interface
CBLB	CPU-based load balancing
CMBLB	CPU-Memory-based load balancing
BWBLB	Bandwidth-based load balancing
OF	OpenFlow
TOS	Type Of Service
OF	Switch OpenFlow-enabled switch
TCP	Transmission Control Protocol
APIC	Application Policy Infrastructure Controller

VAN	Virtual Application Networks
IP	Internet Protocol
HTTP	Hypertext Transfer Protocol
STP	Spanning Tree Protocol
BPDU	Bridge Protocol Data Units
PVST	Per-VLAN Spanning Tree
MSTP	Multiple Spanning Tree Protocol
R-PVST	Rapid Per-VLAN Spanning Tree
R-STP	Rapid Spanning-Tree
R-STP	Rapid Spanning-Tree
R-STP	Rapid Spanning-Tree

Chapter 1
Introduction

1.1 General Concepts

SDN (Software Defined Networking) has recently received widespread attention from customers, vendors and channel partners. As time goes by, SDN has become one of the most common ways for organizations to deploy applications. This technology helps organizations to deploy applications faster and reduce overall deployment costs. Over the years, the technology has been announced as the next focus of the networking industry. Many people are trying to figure out what SDN is and how it will affect their work as a network engineer. It's time to delve into this emerging technology. This book will help you understand SDN and make SDN VS Traditional networking to see which leads the way.

1.1.1 General Idea of Information Technology

Information technology (**IT**) is the use of computers to store, retrieve, transmit, and manipulate data or information, often in the context of a business or other enterprise. **IT** is considered to be a subset of information and communications technology (ICT).

An information technology system (**IT** system) is generally an information system, a communication system or, more specifically speaking, a computer system including all hardware, software and peripheral equipment operated by a limited group of users.

Humans have been storing, retrieving, manipulating, and communicating information since the Sumerians in Mesopotamia developed writing in about 3000 BC but the term information technology in its modern sense first appeared in a 1958 article published in the Harvard Business. Its definition consists of three categories: techniques for processing, the application of statistical and mathematical methods to decision-making, and the simulation of higher-order thinking through computer programs.

1.1.2 Relation between Information Technology and networking

Networking is a specific branch of information technology (IT), and is one of the fastest-growing areas of IT. It involves the processes of building, using the maintaining computer networks including hardware, software and protocols so that multiple computing devices can share data, voice and video. Networking can include home or business computer networks, and wired or wireless computer networks. As well as their classification, computer networks can also differ in their design.

- Local-area networks (LANs).
- Wide-area networks (WANs).
- Metropolitan-area networks (MANs).
- Campus-area networks (CANs).
- Home-area networks (HANs).

1.2 What’s Networking?

A network is a group of two or more computer systems linked together. There are many types of computer networks.

1.2.1 Network models:

● **OSI model:**

The Open Systems Interconnection model (OSI model) is a conceptual model that characterizes and standardizes the communication functions of a telecommunication or computing system without regard to their underlying internal structure and technology.

The model partitions a communication system into seven abstraction layers: Physical layer, Data-link layer, network layer, transport layer, Session layer, Presentation layer, Application layer.

● **TCP/IP model:**

The Transmission Control Protocol / Internet Protocol (TCP/IP) was created by the Department of Defense (DoD) to make sure and protect data integrity A TCP/IP network can be a truly reliable and flexible one.

TCP/IP is based on a four-layer reference model: Application layer,

Transport layer, Internet layer, Network interface layer.

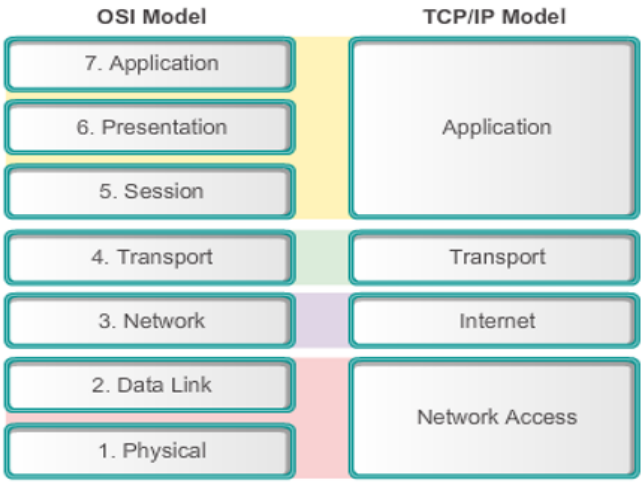


Figure 1.1: OSI and TCP-IP models

1.2.2 Network components

- **Links:**

  There're two types of links: wired such as coaxial cable, fiber, serial, smart serial and wireless.

- **Hubs:**

  It's a layer 2 network device which enables connection between all network elements and hosts without requiring any address.

- **Switches:**

  It's a layer 2 network device which enables connection between all network elements and hosts using layer 2 address called (MAC).

- **Routers:**

  It's a layer 3 network device which enables connection between all network elements and hosts using layer 3 address called (IP).

1.2.3 Some Network fundamentals

- **IP:**

  (Internet Protocol address) is a numerical label (layer 3 address) assigned to all devices (computers, printers, cameras etc..) to enable communication between all networks there're two versions IPv4 (designed as a 32bit number) and IPv6 (designed as 128bit number).

- **MAC:**
  - (Media Access Control address) also called (Hardware/physical address) is a unique identifier assigned to Network Interface Card (NIC), a Network device may have more than NIC and each NIC must have a unique MAC address.
  - It's used as network address for most IEEE 802 network technologies such as: Ethernet, WIFI.
- **APIs:**

  (Application Programming Interface) is a specification of how some software components should interact with each other. In practice, an API is usually a library that includes specification for variables, routines, object classes, and data structures. An API specification can take many forms, including an international standard (e.g., POSIX), vendor documentation (e.g., the Junos SDK), or the libraries of a programming language.

- **Vulnerabilities**:
  - Anomalies in a network refers to circumstances which cause network operations to deviate from normal behavior.
  - Vulnerabilities are inherent weaknesses in the design, implementation and management of a networked system; because of vulnerabilities the system is susceptible to threads.
- **Firmware:**
  - Firmware is software that's installed on a small memory chip on a hardware device into a read-only memory.
  - Firmware allows hardware to be updated.
- **OpenWrt:**
  - It's a highly extensible GNU/Linux distribution for embedded devices. Unlike many other distributions for these routers, OpenWrt is built from the ground up to be a full-featured, easily modifiable operating system for your router.
  - In practice, this means that you can have all the features you need with none of the bloat, powered by a Linux kernel that's more recent than most other distributions.

1.3 Introduction to SDN

SDN environment uses open, application programmatic interfaces (APIs) to support all the services and applications running over the network. These APIs facilitate innovation and enable efficient service orchestration and automation. As a result, SDN enables a network administrator to shape traffic and deploy services to address changing business needs, without having to touch each individual switch or router in the forwarding plane.

- **APIs in SDN**:
    I. Northbound API
    II. Southbound API
    III. Eastbound API
    IV. Westbound API

Chapter 2

Traditional Network

2.1 What is a computer network?

A computer network comprises two or more computers that are connected either by cables (wired) or WIFI (wireless)—with the purpose of transmitting, exchanging, or sharing data and resources. You build a computer network using hardware (e.g., routers, switches, access points, and cables) and software (e.g., operating systems or business applications).

Geographic location often defines a computer network.

For example, a LAN (local area network) connects computers in a defined physical space, like an office building, whereas a WAN (wide area network) can connect computers across continents. The internet is the largest example of a WAN, connecting billions of computers worldwide.

You can further define a computer network by the protocols it uses to communicate, the physical arrangement of its components, how it controls traffic, and its purpose.

Computer networks enable communication for every business, entertainment, and research purpose. The internet, online search, email, audio and video sharing, online commerce, live-streaming, and social networks all exist because of computer networks.

2.2 Computer network types

As networking needs evolved, so did the computer network types that serve those needs. Here are the most common and widely used computer network types:

- **LAN (local area network):** A LAN connects computers over a relatively short distance, allowing them to share data, files, and resources. For example, a LAN may connect all the computers in an office building, school, or hospital. Typically, LANs are privately owned and managed.
- **WLAN (wireless local area network):** A WLAN is just like a LAN but connections between devices on the network are made wirelessly.
- **WAN (wide area network):** As the name implies, a WAN connects computers over a wide area, such as from region to region or even continent to continent. The internet is the largest WAN, connecting billions of computers worldwide. You will typically see collective or distributed ownership models for WAN management.
- **MAN (metropolitan area network):** MANs are typically larger than LANs but smaller than WANs. Cities and government entities typically own and manage MANs.
- **VPN (virtual private network):** A VPN is a secure, point-to-point connection between two network end points (see ‘Nodes’ below). A VPN establishes an encrypted channel that keeps a user’s identity and access credentials, as well as any data transferred, inaccessible to hackers.

2.3 Networking Foundation

2.3.1 Switches

A switch is a device that connects other devices and manages node-to-node communication within a network, ensuring data packets reach their ultimate destination. While a router sends information between networks, a switch sends information between nodes in a single network.

2.3.2 Routers

A router is a physical or virtual device that sends information contained in data packets between networks. Routers analyze data within the packets to determine the best way for the information to reach its ultimate destination. Routers forward data packets until they reach their destination node.

2.3.3 links

The most common network cable types are Ethernet twisted pair, coaxial, and fiber optic. The choice of cable type depends on the size of the network, the arrangement of network elements, and the physical distance between devices.

2.4 Unique identifiers of network

Below given are some unique network identifiers:

- **Hostname:**

  Every device of the network is associated with a unique device, which is called hostname.

- **IP Address:**

  IP (Internet Protocol) address is as a unique identifier for each device on the Internet. Length of the IP address is 32-bits. IPv6 address is 64 bits.

- **DNS Server:**

  DNS stands for Domain Name System. It is a server which translates URL or web addresses into their corresponding IP addresses.

- **MAC Address:**

  MAC (Media Access Control Address) is known as a physical address is a unique identifier of each host and is associated with the NIC (Network Interface Card). General length of MAC address is: 12-digit/ 6 bytes/ 48 bits

- **Port:**

  Port is a logical channel which allows network users to send or receive data to an application. Every host can have multiple applications running. Each of these applications are identified using the port number on which they are running.

2.5 Planes

In the context of telecommunications, the word “plane” means an area of operations, and it is used to distinguish between the traffic flows. Traditional networking is rooted in fixed-function network devices, such as a switch or router. These devices each have certain functions that operate well together and support the network. If the network's functions are implemented as hardware constructs, then its speed is usually bolstered.

Any traditional network component is basically a combination of two planes; control plane and data plane. The data plane is a part of a network through which user packets are transmitted. It is a theoretical term used to conceptualize the flow of data packets through a network infrastructure. It is often included in diagrams and illustrations to give a visual representation of user traffic. The data plane is also known as the user plane, the forwarding plane or the carrier plane.

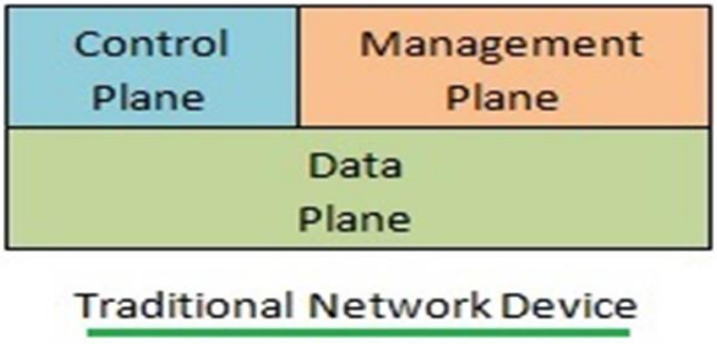


Figure 2.1: Traditional Network Device

In traditional implementations, as referred in figure 2.1, what has been called the user plane, the control plane and the management plane have all been integrated into the firmware of routers. The use of packets to identify and delineate digital information requires some form of data lookup in order to determine packet destination. Routing and forwarding tables provide the answer, and these are an integral part of the routing protocols that manage network traffic. The data plane forwards traffic to the next hop along the path based on information gleaned from data plane packets. These packets travel through routers to create digital conversations in a way that is transparent to internet users.

2.6 Architecture

In the traditional network architecture, control plane and data plane are integrated and any changes to the system are dependent upon physical network devices, the protocols and software they support. Limited changes can be performed to the overall system as the network devices bottleneck logical network traffic flows.

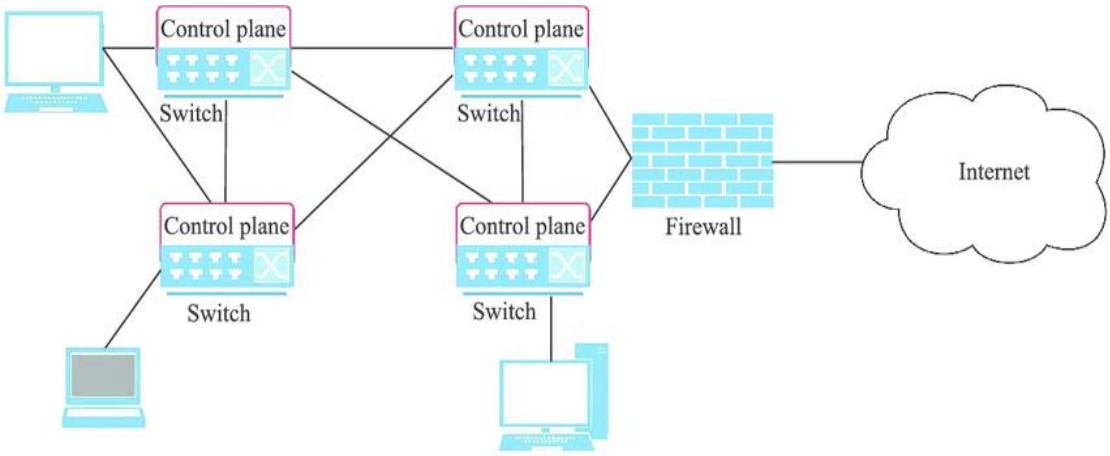


Figure 2.2: Computer network architecture

Computer network architecture in figure 2.2 defines the physical and logical framework of a computer network. It outlines how computers are organized in the network and what tasks are assigned to those computers. Network architecture components include hardware, software, transmission media (wired or wireless), network topology, and communications protocols.

2.7 Traditional network infrastructure

Traditionally, datacenter network infrastructure for large companies or large compute farms was built based on a three- layer hierarchical model, which Cisco calls the “hierarchical inter-networking model”. It consists of core layer switches which

connect to distribution layer switches (sometimes called aggregation switches), which in turn connect to access layer switches. Access layer switches are frequently located at the top of a rack, so, these are also known as top-of-rack (TOR) switches. Most network infrastructure is still laid out this way today.

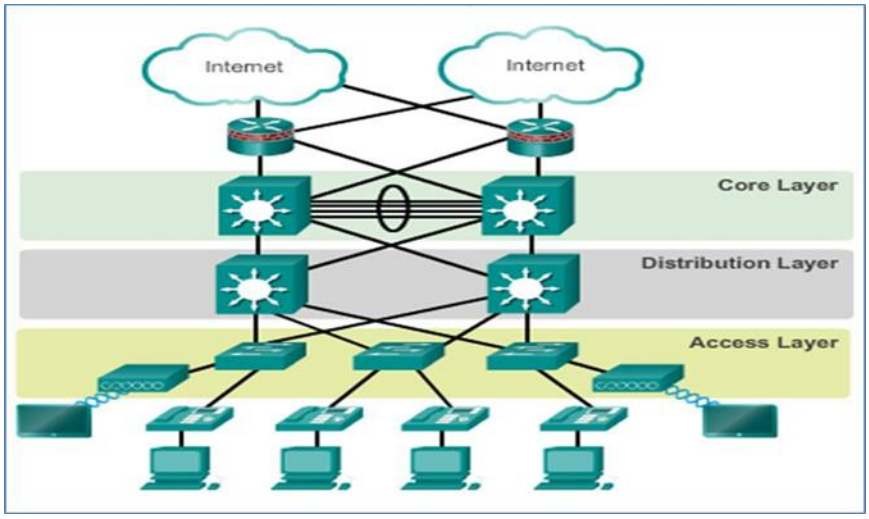


Figure 2.3: The Hierarchical Model in a Datacenter Network Switching Architecture

The good news with this hierarchical model is that traffic between two nodes in the same rack, if at Layer 2 of the network stack, is sent with low latency. If the access switches are 10Gb, then the communication can have high throughput as well. Also, this type of configuration allows for a vast number of ports at the access layer.

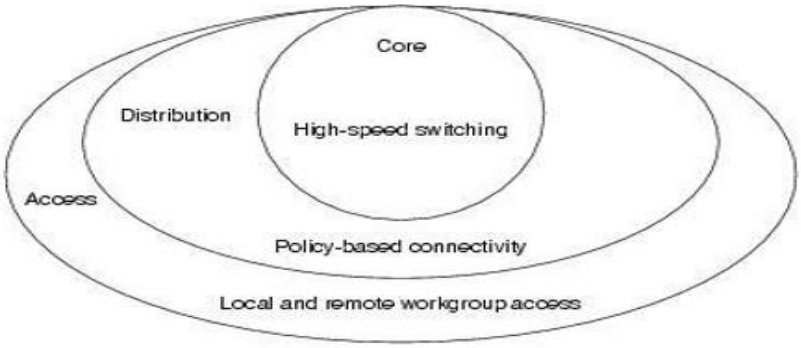


Figure 2.4: Layers Hierarchical

The bad news is, well, everything else about this model. It's expensive. East – west communication, say between racks of gear, means that traffic travels to the aggregation layer and frequently to the data center core. These multiple hops, frequently across oversubscribed backplanes, take a very long time – 50usec or more via traditional vendors via traditional vendors and their traditional solutions. Any Layer 3 traffic needs to leave the rack and reach the aggregation tier of switches before- being routed, even back to the same rack it came from. And east-west traffic isn't the exception in modern application deployments, it's the norm. This traffic is from applications talking to each other, talking to databases, and talking to IP-attached storage.

2.7.1 Core layer

The core layer is a high-speed switching backbone and should be designed to switch packets as fast as possible. This layer of the network should not perform any packet manipulation, such as access lists and filtering, that would slow down the switching of packets.

2.7.2 Distribution layer

The distribution layer is the smart layer in the three-layer model. Routing, filtering, and QoS policies are managed at the distribution layer. Distribution layer devices also often manage individual branch-office WAN connections. This layer is also called the Workgroup layer.

In the non-campus environment, the distribution layer can be a redistribution point between routing domains or the

demarcation between static and dynamic routing protocols. It can also be the point at which remote sites access the corporate network. The distribution layer can be summarized as the layer that provides policy-based connectivity.

2.7.3 Access layer

End-stations and servers connect to the enterprise at the access layer. Access layer devices are usually commodity switching platforms, and may or may not provide layer 3 switching services. The traditional focus at the access layer is minimizing "cost-per-port": the amount of investment the enterprise must make for each provisioned Ethernet port. This layer is also called the desktop layer because it focuses on connecting client nodes, such as workstations to the network.

As the traditional network grow and expand, it must:

- **Support critical applications**

  Critical application is very important in case of (Failure Domains) Failure Domains, are areas of a network that are impacted when a critical device or network service experiences problems such as data center.

  Redundant links and enterprise class equipment minimize disruption of network. Smaller failure domains reduce the impact of a failure on company productivity. Smaller failure domains also simplify troubleshooting. Switch block deployment – each switch block acts independently of the others. Failure of a single device does not impact the whole network.

- **Support converged network traffic**

Network convergence is the efficient coexistence of telephone, video and data communication within a single network. The use of multiple communication modes on a single network offers convenience and flexibility that are not possible with separate infrastructures. Network convergence is also called media convergence.

- **Support diverse business needs (scalability)**

Use expandable, modular equipment or clustered devices. Include design modules that can be added, upgraded, and modified, without affecting the design of the other functional areas of the network. Create a hierarchical addressing scheme.

Use routers or multilayer switches to limit broadcasts and filter traffic.

- **Provide centralized administrative control**

To provide a high-reliability network, Hierarchical Network Design will be installed. This model divides the network functionality into three distinct layers.

2.8 Switched Networks

2.8.1 Distribution Layer:

A distribution switch is a distribution layer switch, which uplinks to upper layer core switch and links down to the access/edge switch. Simply put, an aggregation switch in between functions a bridge between core layer switch and

access layer switch. This is also the reason why we call distribution switch as aggregation switch.

2.8.2 Distribution Switch Functions:

Why distribution layer switch is needed in network layers? In general, aggregation switch reacts on multiple switch aggregation and layer 3 routing functionality. It also supports complex policy implementation such as QOS and packet filtering.

2.8.3 Distribution Switch Protocols:

STP	To resolve switched network loop
VLAN	Divide users in multi LANs and mitigate broadcast
HSRP	Resolve multiple gateway issue in switched networks
Routing	Allow local networks to reach remote networks

Table 2.1: Distribution Switch Protocols

2.9 Virtual LANs (VLANS):

VLANs can segment LAN devices without regard for the physical location of the user or device.

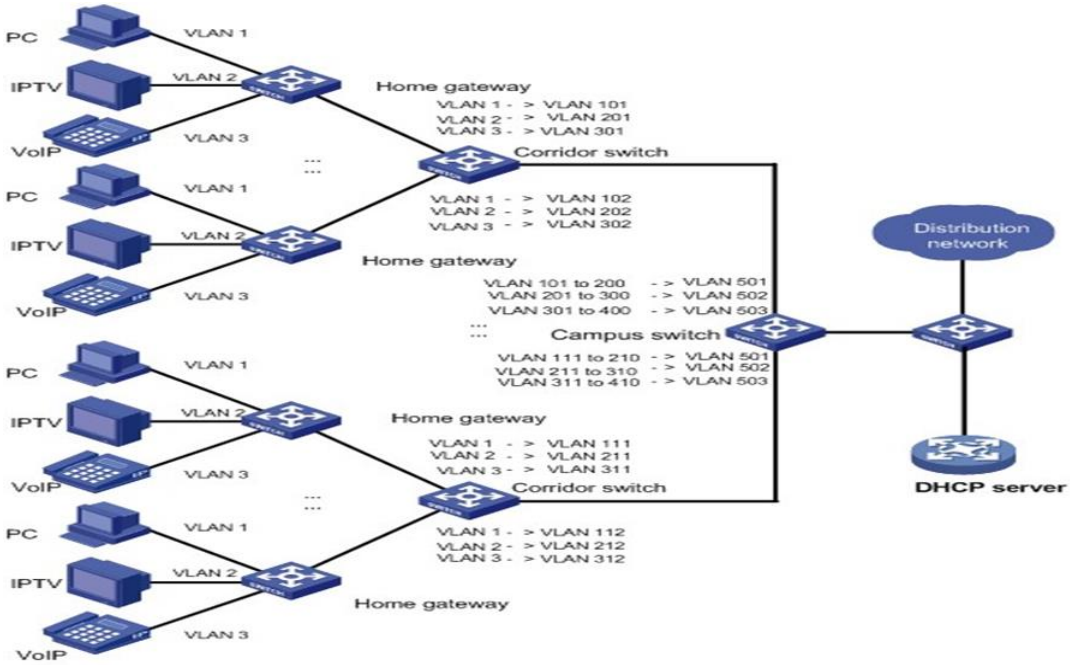


Figure 2.5: *VLAN Architecture*

2.9.1 The advantages of using VLANs:

1. A VLAN is a single broadcast domain.
2. Users are only able to communicate within the same VLAN
3. Users don't have to be grouped physically together.
4. Hosts on the same VLAN are unaware of the VLAN's existence.

2.9.2 VLAN ranges

- Normal range VLANs: VLAN numbers from 1 to 1,005
- Extended Range VLANs: VLAN numbers from 1,006 to 4,096

➢ In the project we will always use

Vlan10	For managers
Vlan20	For HR
Vlan30	For sales
Vlan40	For accountants

Table 2.2: Vlans Numbers

to allow administrator to manage ,admin and audit all network users LAN Redundancy.

- Layer 2 switches cannot forward traffic between VLANs without the assistance of a router.
- Inter-VLAN routing is a process for forwarding network traffic from one VLAN to another, using a router. So we will use SVI inter VLAN routing to allow end users from VLANs 10, 20 to communicate with each other's and other sites through The Layer 3 switches at the distribution layer also we will use these Layer 3 Switches in the distribution layer as a multiple Getaways for the end users.

2.10 Spanning Tree Protocol

Prevent loops from occurring using strategically placed blocking state ports:

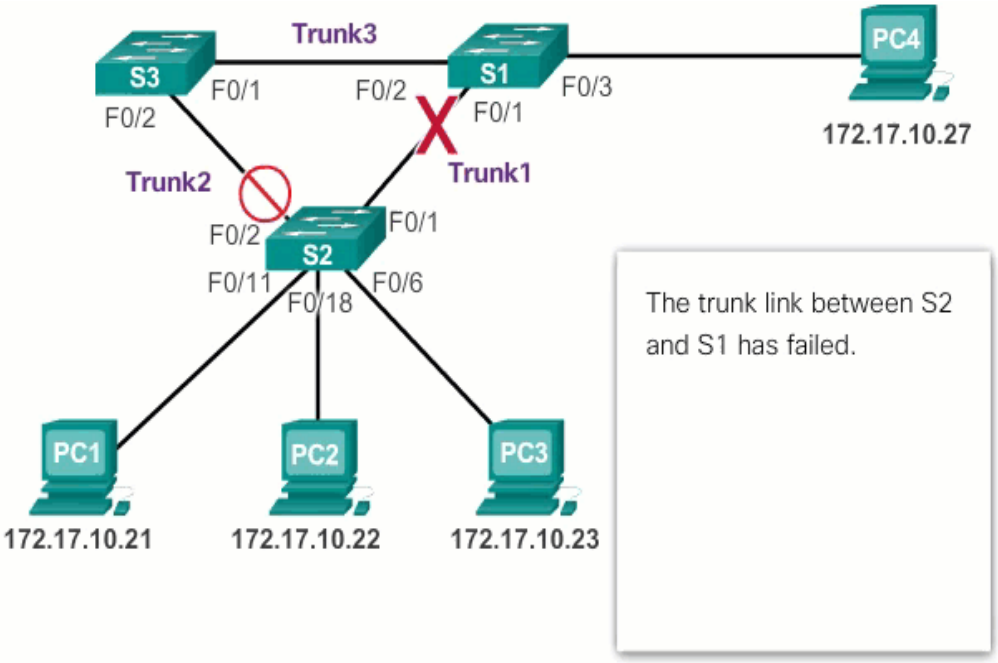


Figure 2.6: ***Blocked port by STP 22***

SPANING TREE PROTOCOL COMMAND:

Switch# configure terminal

Switch (config)# spanning-tree vlan, vlan_ID

Switch (config)# end

Switch# show spanning-tree vlan vlan_ID

2.10.1 STP types

1) **IEEE 802.1D (default STP)**

   Features:

   - Slow.
   - Requires high resources (processor, RAM).
   - Doesn't support VLAN.

2) **pair VLAN spanning tree (PVST)**

   Features:

   - Slow
   - High resources
   - Support VLAN

3) **IEEE 802.1W (rapid STP)**

   Features:

   - Rapid
   - Medium resources
   - Doesn't support VLAN

4) **Rapid Per VLAN Spanning Tree (RPVST)**

   Features:

   - Very rapid
   - High resources
   - Support VLAN

   We have chosen RPVST protocol in our study case

2.11 HSRP (Redundancy)

Hot Standby Router Protocol (HSRP) - A Cisco-proprietary FHRP designed to allow for transparent failover of a first-hop IPv4 device. Active device is the device that is used for routing packets. Standby device is the device that takes over when the active device fails. Function of the HSRP standby router is to monitor the operational status of the HSRP group and to quickly assume packet-forwarding responsibility if the active router fails.

2.11.1 HSRP operations layer 3 redundancy

HSRP defines a group of switches L3 one active and one standby.

Virtual IP and mac address are shared between the two switch L3.

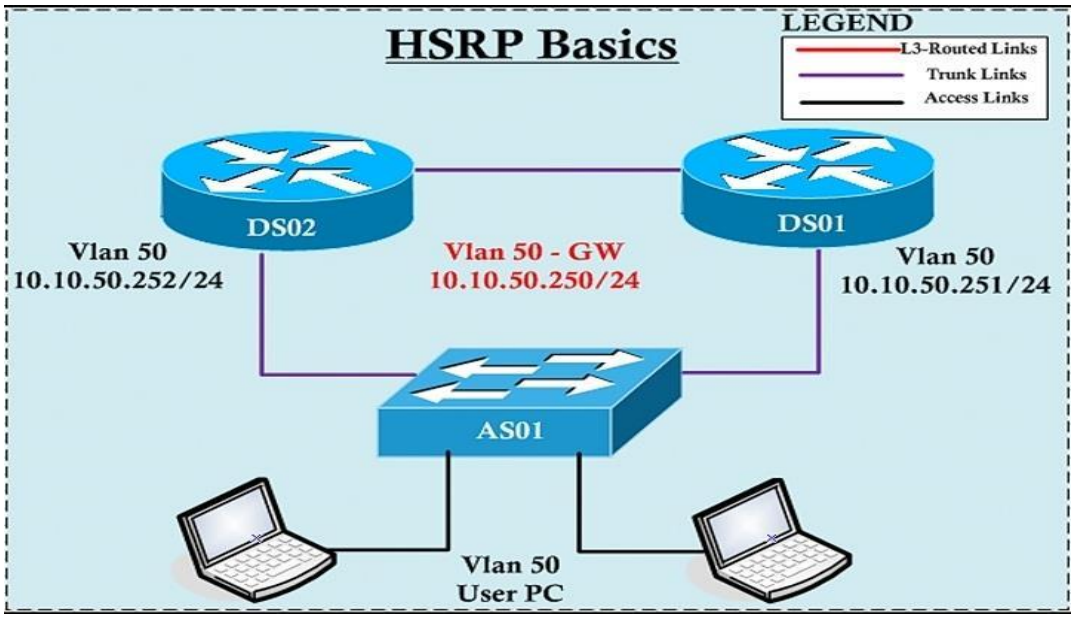


Figure 2.7: HSRP Basics

2.11.2 HSRP configuration

Step 1: Configure the virtual IP address for the group.

Step 2: Configure the priority for the desired active router to be greater than 100.

Step 3: Configure the active router to preempt the standby router in cases where the active router comes online after the standby router.

To verify HSRP state use the show standby command to increase bandwidth we have created Ethernet channel between distribution layer and access layer. Ethernet channel, which is a cisco protocol that aims to collect physical interfaces (1-8) into one logical interface.

Note: physical ports status must be trunk.

2.12 Routing CORE Layer:

Routing protocol specifies how routers communicate with each other, disseminating information that enables them to select routes between any two nodes on a computer Network.

Reaching Remote Networks

There are two ways to assign ip address to devices:

1) Static Manually for network devices and servers: Manually entered into the route table using static routes, Static routes are not automatically updated and must be reconfigured when topology changes.
2) Dynamic DHCP for end users; dynamically (Automatically) learned using a routing protocol.

2.12.1 Static Routing

Three uses for static routes:

- Smaller networks that are not expected to grow.
- Routing to and from stub networks.
- A single default route to represent a path to any network not found in the routing table.

2.12.2 Dynamic routing protocols

- Discovery of remote networks.
- Maintaining up-to-data routing information.
- Choosing the best path to destination networks.
- Ability to find a new best path if the current path is no longer available.

2.12.3 IPv4 dynamic Routing Protocols

EIGRP	Enhanced Interior Gateway Routing Protocol
OSPF	Open Shortest Path First
RIP	Routing Information Protocol

TABLE 2.3: IPv4 dynamic Routing Protocols

2.13 The Access layer

Switched network devices which provide a network connection for your end devices like PCs.

A hierarchical switched LAN allows more flexibility, traffic management, and additional features:

- Quality of service
- Additional security

- Support for wireless networking and connectivity
- Support for new technologies.

2.13.1 Switched network protocols

STP	To resolve switched network loop
VLAN	Divide users in multi LANs and mitigate broadcast
Port security	To prevent the end user ports from attacking the network

TABLE 2.4: Switched network protocols

STP was covered in the distribution layer but we must modify the STP default configuration by changing the priority of layer 3 switches (the default priority for all switches is 32768 and we shall change it to the lowest value 4096 in order to force the switch to be root) to be the lowest in order to make them one root primary and the other root secondary to secure the BPDU message and control the loop form the distribution side.

Also we must change the access layer switches priority to the highest value 61440 in order to never been a root switches ,in additional in all the end users ports we should enable the **STP** port fast to deny these ports to listen to **BPDU** message coming from the root to all switches to prevent local network attacks.

VLANS, also covered in the distribution layer but we need to enable them at the access layer to shrink switched network broadcast and prevent the loop.

Switch Port Security :(secure user access with mac address)

Port security limits the number of valid MAC addresses allowed to transmit data through a switch port. This add more security and prevent users from attacking the switched network devices through Vlan hopping attack or DHCP attack.

If a port has port security enabled and an unknown MAC address sends data, the switch presents a security violation.

```
interface FastEthernet0/1
 switchport access vlan 10
 switchport mode access
 switchport port-security mac-address sticky
 spanning-tree portfast
 spanning-tree bpduguard enable
!
interface FastEthernet0/2
 switchport access vlan 10
 switchport mode access
 switchport port-security mac-address sticky
 spanning-tree portfast
 spanning-tree bpduguard enable
!
interface FastEthernet0/3
 switchport access vlan 10
 switchport mode access
 switchport port-security mac-address sticky
 spanning-tree portfast
 spanning-tree bpduguard enable
!
```

Figure 2.8: Codes to Make Vlans

2.13.2 Methods use to configure MAC addresses within port security

1. Static secure MAC addresses – manually configure switch port port-security mac-address.
2. Dynamic secure MAC addresses – dynamically learned and removed if the switch restarts
3. Sticky secure MAC addresses – dynamically learned and added to the running configuration (which can later be saved to the startup-config to permanently retain the MAC addresses)

2.13.3 Violation Modes:

- Protect – data from unknown source MAC addresses are dropped; a security notification IS NOT presented by the switch
- Restrict - data from unknown source MAC addresses are dropped; a security notification IS presented by the switch and the violation counter increments.
- Shutdown – (default mode) interface becomes error-disabled and port LED turns off.

And we will use the violation shutdown in our devices for more security reasons switch port security .

Figure 2.9: Port Security

Chapter 3

SDN (Software Defined Network)

3.1 SDN Technology

Software-Defined Networking (SDN) technology is an approach to cloud computing that facilitates network management and enables programmatically efficient network configuration in order to improve network performance and monitoring. SDN is meant to address the fact that the static architecture of traditional networks is decentralized and complex while current networks require more flexibility and easy troubleshooting. SDN attempts to centralize network intelligence in one network component by disassociating the forwarding process of network packets (data plane) from the routing process (control plane). The control plane consists of one or more controllers which are considered as the brain of SDN network where the whole intelligence is incorporated. However, the intelligence centralization has its own drawbacks when it comes to security, scalability and elasticity and this is the main issue of SDN.

SDN was commonly associated with the Open Flow protocol for remote communication with network plane elements for the purpose of determining the path of network packets across network switches.

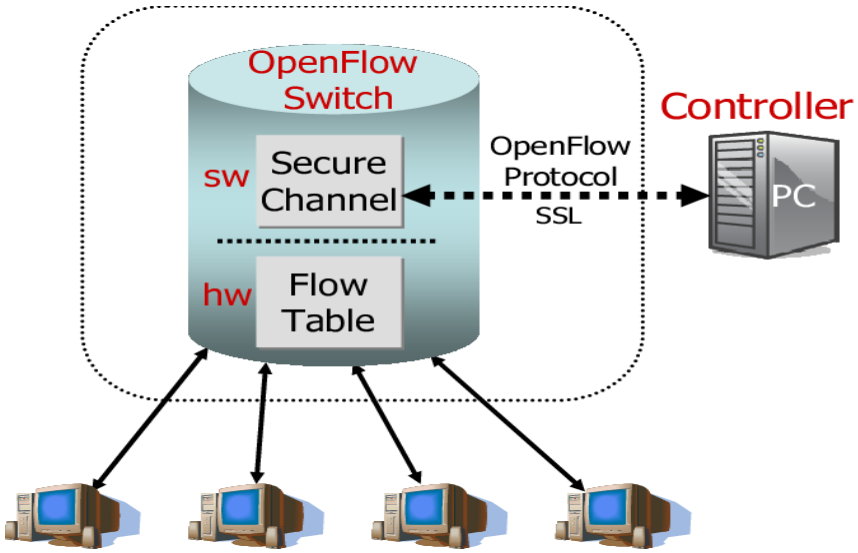


Figure 3.1: SDN Model

Instead of having a distributed control architecture, it consolidates all the control in a single node called the “Network Controller” which is simply a software running on commercial server platform. Network forwarding devices no longer participate in the network control & only forward packets based on the set of rules installed from the network controller.

The network controller programs the forwarding rules of the forwarding devices through the “OpenFlow protocol” hence the network forwarding devices are called “OpenFlow switches”.

3.2 Main features of SDN

- **Separate Control (brains) plane and Data (muscle) plane entities:**

  Network intelligence and state are logically centralized

  The underlying network infrastructure is abstracted from the applications.

- **Execute or run Control plane software on general purpose hardware:**

  Decouple from specific networking hardware, Use commodity servers.
- **Have programmable data planes – APIs:**

  Maintain, control and program data plane state from a central entity.

3.3 Why we need SDN?

- **Virtualization:** Use network resource without worrying about where it is physically located, how much it is, how it is organized, etc. Abstraction ⇒ Virtualization.
- **Orchestration:** Should be able to control and manage thousands of devices with one command.
- **Programmable:** Should be able to change behavior on the fly.
- **Dynamic Scaling:** Should be able to change size, quantity Virtualization Scaling.
- **Automation:**
  a. To lower OpEx minimize manual involvement
  b. Troubleshooting
  c. Reduce downtime
  d. Policy enforcement
  e. Add new workloads, sites, devices, and resources
- **Visibility:**

  Monitor resources, connectivity.
- **Performance:**
  a. Optimize network device utilization
  b. Traffic engineering/Bandwidth management

c. Capacity optimization
d. Load balancing
e. High utilization
f. Fast failure handling

- **Multi-tenancy**: Tenants need complete control over their addresses, topology, and routing, security.
- **Service Integration:** Load balancers, firewalls, Intrusion Detection Systems (IDS), provisioned on demand and placed appropriately on the traffic path

3.4 SDN Architecture

SDN broadly consists of three layers:

- Application layer
- Control layer
- Infrastructure layer

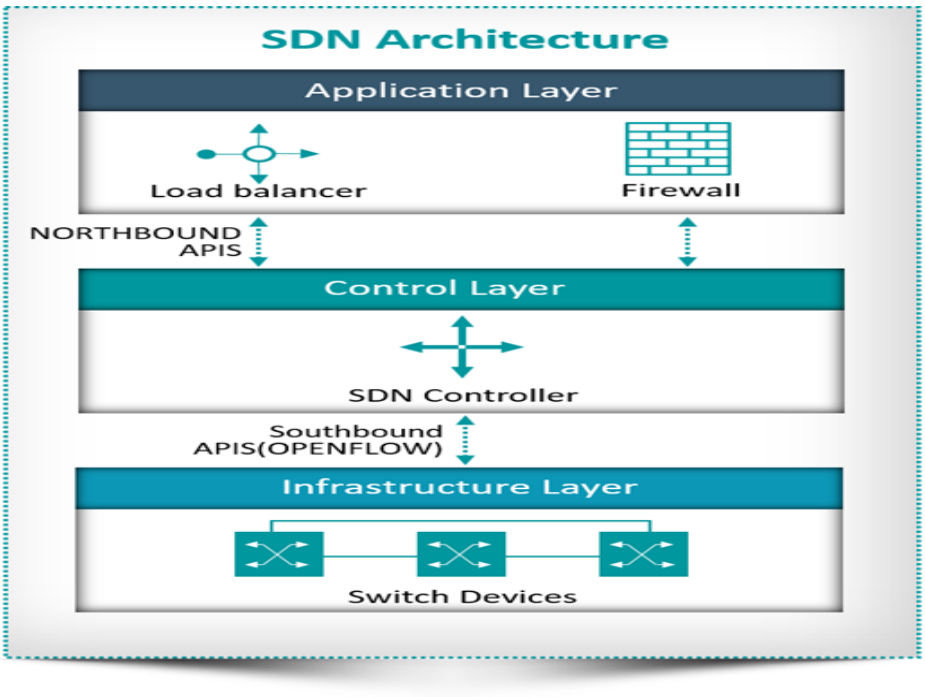


Figure 3.2: SDN Architecture

1. **Application layer**

   The application player comprises one or more applications, each of which has exclusive control of a set of resources exposed by one or more SDN controllers. Additional interfaces to applications are not precluded.

   An application may invoke or collaborate with other applications.

   An application may act as an SDN controller in its own right.

2. **Control Layer**

   The controller plane comprises a set of SDN controllers, each of which has exclusive control over a set of resources exposed by one or more network elements in the data plane (its span of control).

   Additional interfaces to SDN controllers are not precluded

   - The minimum functionality of the SDN controller is to faithfully execute the requests of the applications it supports, while isolating each application from all others.
   - To perform this function, an SDN controller may communicate with peer SDN controllers, subordinate SDN controllers, or non-SDN environments, as necessary.
   - A common but non-essential function of an SDN controller is to act as the control element in a feedback loop, responding to network events to recover from failure, reoptimize resource allocations, or otherwise.

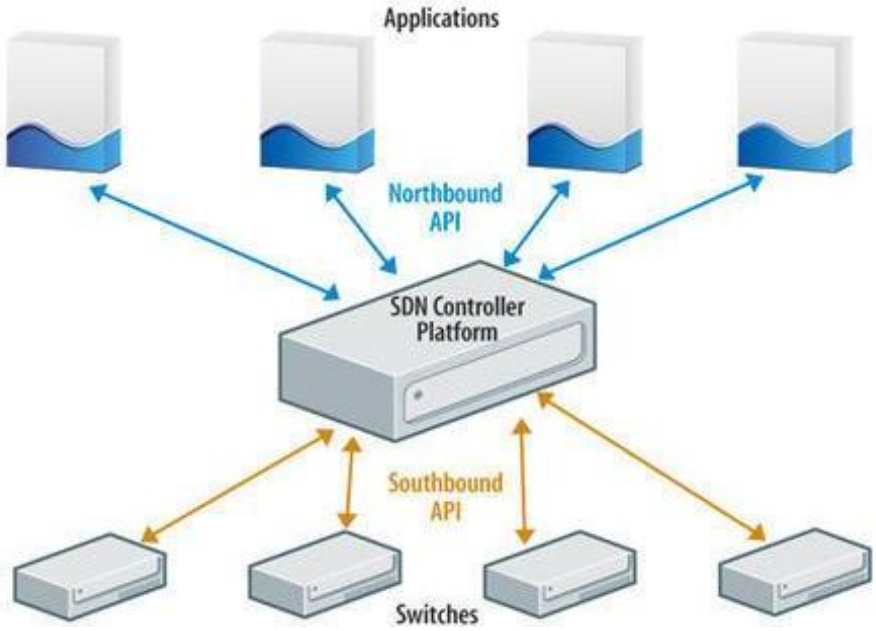


Figure 3.3: SDN controller

- **Northbound interface**: is meant for communication with upper, Application layer and would be in general realized through REST APIs of SDN controllers.
- **Southbound interface**: is meant for communication with lower, Infrastructure layer of network elements and would be in general realized through southbound protocols OpenFlow, Netconf, Ovsdb, etc.
- **The east-west interfaces**: which are not yet standardized, allow communication between the multiple controllers. They use a system of notification and messaging or a distributed routing protocol like BGP and OSPF.

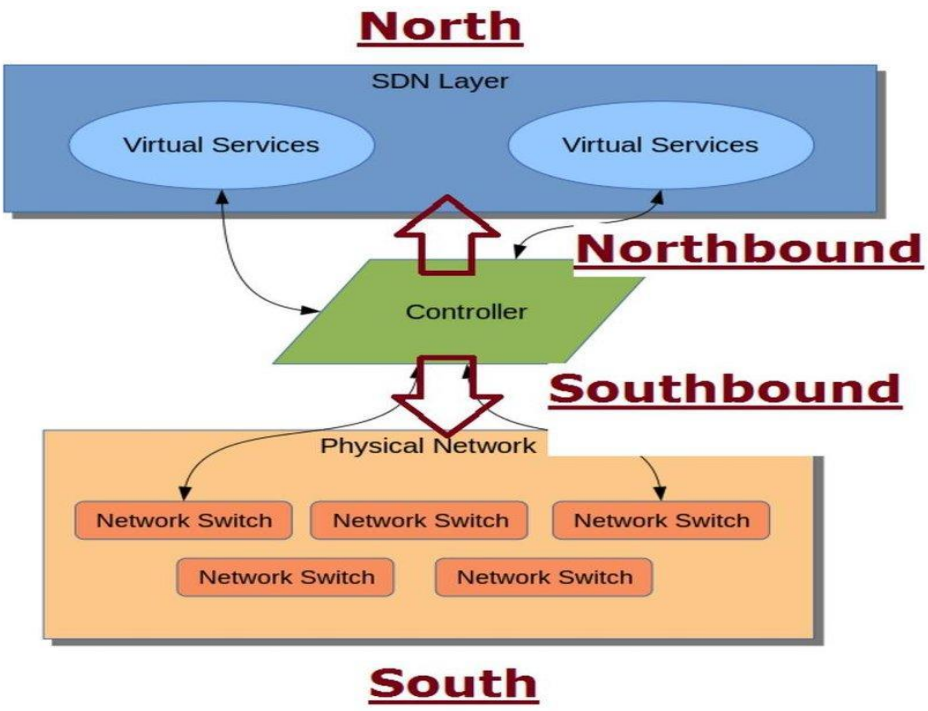


Figure 3.4: North & south Bounds

3. Infrastructure Layer

(Is also known as data plane) is composed of various networking equipment which forms underlying network to forward network traffic. It could be a set of network switches and routers in the data center. This layer would be the physical one over which network virtualization would be laid down through the control layer (where SDN controllers would sit and manage underlying physical network).

In SDN architecture, Application programming interfaces (APIs) are used for implementing common network services which are customized to meet business demands such as routing, load balancing, access control, bandwidth management, energy management, QoS.

The nodes at control layer are called as controllers, and they send information (e.g., routing, switching, priority, etc.) to the data plane nodes associated with them.

After receiving the information from control node, the networking devices in the data plane update their forwarding table according to the information received from the control plane. The control nodes can further be architecturally classified into centralized and distributed mode.

In centralized mode, a single central controller sends switching, routing and other information to all networking hardware in the network.

Meanwhile, in distributed mode, there are plenty of controllers associated with certain networking hardware that send information to them.

The centralized mode possesses a risk of single point of failure; therefore load-balancing and redundancy mechanisms are often applied in centralized approach deployment.

3.5 SDN Advantages

SDN will change the way that network engineers and designers build and operate their networks to achieve business requirements. With the introducing of SDN, networks have become open standards and easy to program and manage.

SDN will give the network administrators more control on their networks allowing them to optimize their networks to reduce the overall cost; therefore, SDN seems to be the perfect solution to today’s huge data networking problems.

Some of the main SDN benefits can be summarized below:

- **Directly programmable**:
  SDN network is directly programmable because the control plane is decoupled from forwarding (data) plane.
- **Agile**:
  It is easy to control the network traffic. The behavior of the network can adapt fast to changes like failures or new traffic patterns.
- **Centrally controlled**:
  As mentioned above, there is a centralized device called “controller” and considered as the network brain which provides the network control and overall view of the network. Instead of configuring thousands of devices, the network configuration is performed on the controller to apply network rules and policies. As a result, it saves effort and time.
- **Open standards-based**:
  Most of the SDN controllers are not vendor specific; in addition, they are open source which enables easy programming.
- **Network management Simplicity**:
  With SDN, the network can be viewed and managed as a single node which will transfer complicated default network management tasks to be abstracted in a rather easy way to manage interfaces.
- **Fast service deployment**:
  New features and applications can be deployed in a fast manner within hours instead of many days and years. SDN has been designed to simplify network configuration and facilitate innovation.

- **Automated configuration**: Manually configuration tasks such as assigning VLAN and configuring QoS can be provisioned automatically by applying specific applications.
- **Saving cost, effort and time**:
  With the aid of using SDN technology, there is no need to purchase expensive network devices to get higher and specialized features because SDN allows implementing applications due to the network requirements. Operators can implement features in the software they control, rather than having to wait for a vendor to add them to their proprietary products. Centralized control helps to eliminate manual interaction with the hardware, improving the uptime of the network and saving the effort.
- **CAPEX reduction**:
  Separating the data plane from the control plane brings to a simpler hardware and increases the possibility of more competence between hardware manufacturers, since the devices do not depend on the proprietary software.

3.6 SDN Disadvantages

- It requires change in entire network infrastructure to implement SDN protocol and SDN controller. Hence it requires complete reconfiguration of the network. These increases cost due to reconfiguration.
- Staff need to be trained.
- New management tools need to be procured and everyone should be trained to use it.
- Security is a big challenge in SDN.

- Single point of failure.

3.7 SDN applications

SDN has gained a lot of attention in recent years because it allows programmability in the current networking architectures and enables easier and faster network innovation. Facilitates software implementations of complex networking applications. Additionally, there is the expectation for more flexibility by dynamically adding new features to the network in the form of networking applications. This concept is known from mobile phone operating systems, such as Apple's iOS and Google's Android, where "apps" can dynamically be added to the system.

To give an idea of how importance SDN is, the list below mentioned some SDN applications:

- **Appliance Virtualization**: Firewalls, Content distribution, and Gateways.
- **Service Assurance**: Content-specific traffic routing for optimal Quality of Experience (QoE), Congestion control based on network conditions, and Dynamic policy-based traffic engineering.
- **Traditional Control Plane**: Network discovery and Path computation and Optimization.
- **Network Virtualization**: Virtual network control on shared infrastructure, Multitenant network automation & API.
- **Application Enhancement**: Specific SDN application, reserved bandwidth for application needs, Geo-distributed applications, and intelligent network responses to app needs.

- **Server Load Balancing Application**: Load balancing is a necessary requirement for enterprise networks so it can provide high availability and scalability for the requests to a particular service. Normally such functionality of balancing loads among several servers is implemented by a dedicated device that is implemented in the network. Although with SDN, an OpenFlow switch can deal with this functionality automatically and will distribute the traffic to different servers; therefore, it is possible to write an application that works on top of the controller that can provide a scalable and efficient load-balancing application. With such application, the need for a dedicated middle in the network will be eliminated.
- **Data Centers Upgrading**: Data centers are an essential internal part of many large-scale companies. For example, Google, Facebook, Amazon, and Yahoo have large numbers of data centers to accommodate the huge number of requests and response to them quickly. Such data centers are extremely expensive and complicated to maintain and run. SDN and OpenFlow allow companies to cut costs of setting up and configuring the data center since data forwarding parts of the network can be managed from a central location.

3.8 SDN Security Challenges in SDN Environments

Within the architecture, we need to:

- Secure the Controller: as the centralized decision point, access to the SDN Controller needs to be tightly controlled.
- Protect the Controller: if the SDN Controller goes down (for example, because of a DDoS attack), so goes the network, which means the availability of the SDN Controller needs to be maintained.
- Establish Trust: protecting the communications throughout the network is critical. This means ensuring the SDN Controller, the applications loaded on it, and the devices it manages are all trusted entities that are operating as they should.
- Create a Robust Policy Framework: what's needed is a system of checks and balances to make sure the SDN Controllers are doing what you actually want them to do.
- Conduct Forensics and treatment: when an error happens, you must be able to determine what it was, recover, potentially report on it, and then protect against it in the future.

Chapter 4

Traditional Network VS. SDN

4.1 Traditional Network Disadvantages

Many steps are needed when an IT administrator needs to add or remove a single device in a traditional network:

- Time consuming: It takes a lot of time because the administrator will have to manually configure multiple devices (Routers, Switch,) or if he want to update numerous configuration settings or to do any managements to tools for updating or loading new applications program which can be made by Cisco, Microsoft or any other company, so it's consuming time.
- Error prone: It increased the probability of errors because of personal configuration for each device.
- Limit automation and innovation because everything are made by hand.
- Investment for hardware and software can be costly for initial set-up
- If you don't take proper security precautions like file encryption, firewalls then your data will be at risk.
- Some components of the network design may not last for many years, and it will become useless or malfunction and need to be replaced.
- Frequent server failure and issues of regular cable faults.

4.2 Manual configurations strain traditional networks

This new digital paradigm is putting a strain on conventional network architectures to keep pace. In a conventional networking environment, network devices have control plane functions that are used to ‘make decisions’, relying on programmable rules, on where to send frames or packets that enter the network device, such as a switch. In this environment, whenever adding new applications or changing rules for how traffic flows across the network, changes must be made directly to the physical devices. Imagine how challenging this scenario is for IT teams with dozens or hundreds of network devices spread across branch offices or data centers? In trying to meet the requirements of today’s digital businesses, organizations are finding themselves restricted by the limitations of traditional enterprise network environments.

- **Static environment:** Because implementing network-wide policies is time-consuming and complex in a traditional network, the network often remains ‘untouched.’ A rigid network environment makes it harder for businesses to take advantage of new opportunities, like adding new applications or web services, stifling innovation of handcuffing business growth.
- **Inability to scale:** Scalability is also reduced because networks are overburdened and can’t quickly be re-programmed to accommodate mission-critical traffic.
- **Vendor lock-in:** Another challenge of traditional networking environment is that companies are often

locked in with a single vendor, with no standard protocols to configure equipment across network manufacturers. This rigidity limits the opportunity to tailor the network to meet individual business needs.

4.3 Why Companies Are Shifting to SDN?

As data centers continue to change and traditional networking fails to adapt to the environment, vendors are turning to SDN. Here are some reasons:

- First, the proliferation of cloud services means that users need unfettered access to infrastructure, applications and IT resources. And this comes with requirement for more storage, computing, and bandwidth.
- Second, IT is becoming a consumer commodity where BYOD (bring-your-own-device) trend requires networks to be flexible and secure enough to protect data and assets as well as to meet compliance regulations and standards.

However, traditional networking cannot meet the increasing demands because it must adhere to product cycles and proprietary interfaces typical in vendor-specific environments. Network operators are often hindered when trying to customize the programming of the network. Adding and moving devices or bolstering capacity to traditional networking is complex and time-consuming. It requires manual access to individual devices and consoles. The reason SDN becomes an alternative is that it allows administrators to configure resources and bandwidth instantaneously and bring flexibility, efficiency, and resiliency to the data center. It also eliminates the need to invest in more physical infrastructure.

4.4 The Advantages of SDN

There are numerous advantages of SDN. SDN increases network flexibility through holistic management of the network and enables rapid innovation. The SDN model has the potential to make significant improvements to service request response times, security, reliability and scalability. It could also reduce costs by automating many processes that are currently done manually, which are resource intensive, slow and costly due to the use of restrictive commodity hardware. SDN offers a more efficient and flexible network that increases the speed of service delivery. It delivers cost savings on hardware and also offers the ability to test new protocols in hindsight.

4.5 Is SDN different from traditional networking?

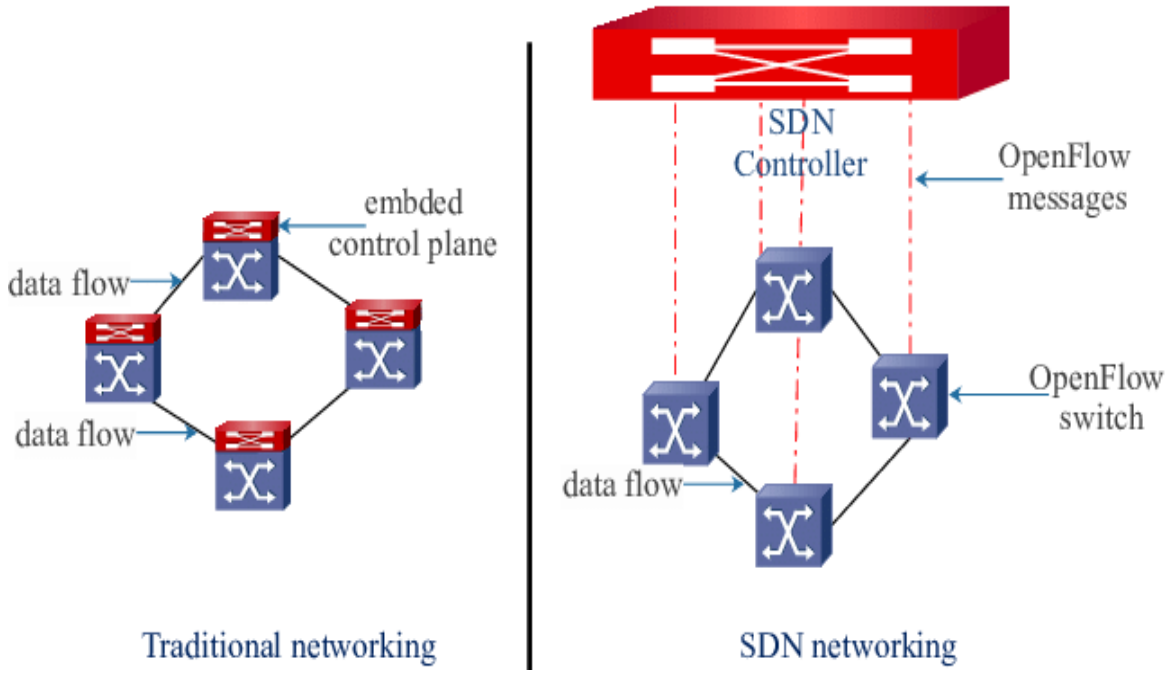


Figure 4.1: Architecture Difference between Traditional IP

There are three most important differences between traditional networking and SDN.

First, the SDN controller has a northbound interface that communicates with applications via application programming interfaces (APIs). This enables application developers to program the network directly. While traditional networking works through using protocols.

Second, SDN is a software-based network, which allows users to control virtual-level resource allocation through the control plane and to determine network paths and proactively configure network services. While traditional networking relies on physical infrastructure (such as switches and routers) to establish connections and run properly.

Third, SDN has more ability to communicate with devices throughout the network than traditional networking. SDN allows resources to be provisioned from a centralized location, and offer administrators the right to control traffic flow from a centralized user interface through more rigorous review. It virtualizes the entire network and gives users more control over their network capabilities. However, for traditional networking, the control plane is located in a switch or router, which is particularly inconvenient. The administrators cannot easily access it to dictate traffic flow.

4.6 Traditional Networks vs. SDN

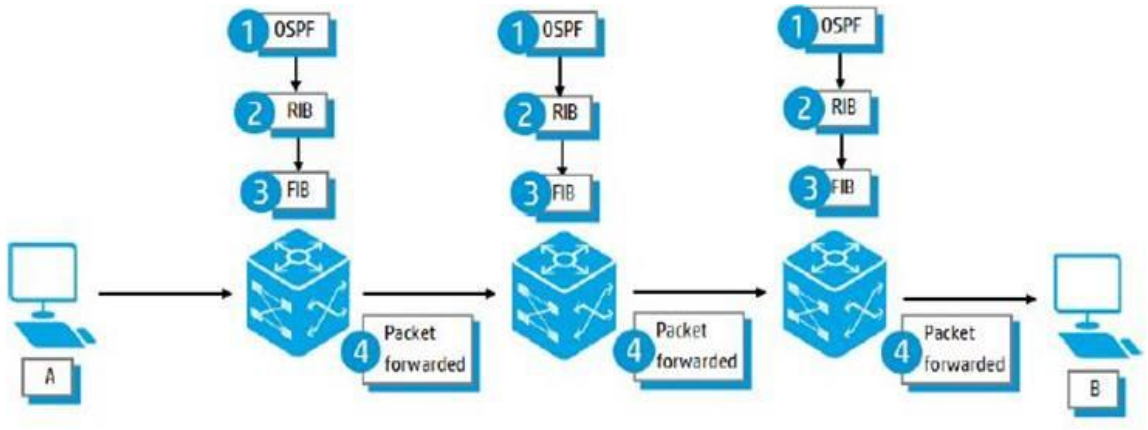


Figure 4.2: Traditional Network - Configuration complexity increases the need for synchronization which limits performance of devices.

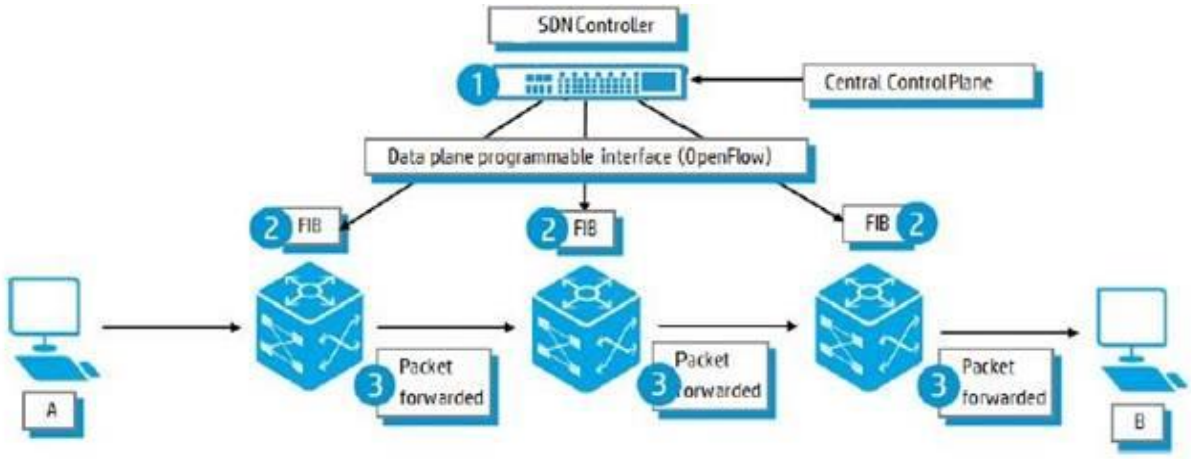


Figure 4.3: SDN network, modular control and central view of network components.

4.7 Real lab STUDY CASE: SOHO Company

SOHO company: it is a company from one site consists of 50 Employee divided into 8 departments interconnected together with data center and internet So, to design a network for this company we find that we need the following devices depending on network hierarchal design model.

- 4 Switches (Cisco Catalyst 2960-x Switch) ACCESS LAYER.
- 2 Multilayer switches (Cisco Catalyst 3650 Series Switches) DISTRIBUTION LAYER.
- Router (Cisco 2911 Integrated Services Router) CORE LAYER.

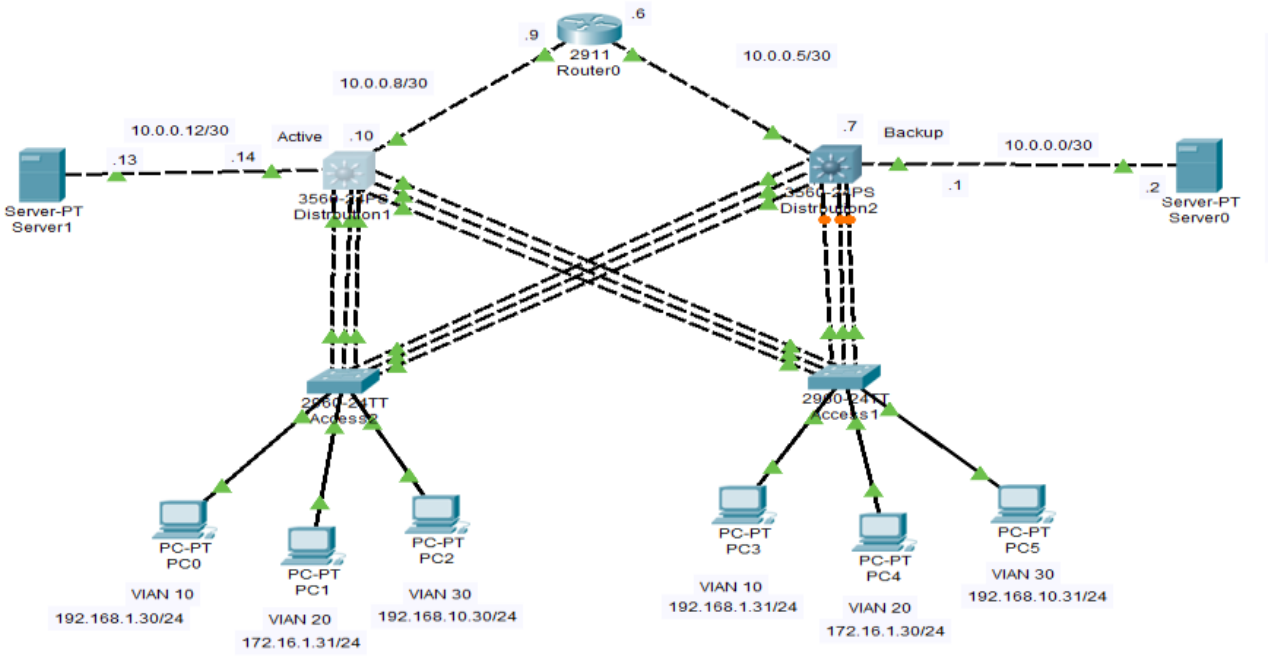


Figure 4.4: Design of SOHO company.

4.7.1 Infrastructure Cost

Component	Cost
Switch 2960	4000$*4=16000 USD
Switch 3560	9000$* 2 = 18000 USD
Router 2911	9000$
RAC and Power Distribution and cables	2000$
PC For each employee	500 USD * 50 = 25000 USD

Table 4.1: TOTAL COST = AROUND 70,000 USD.

4.7.2 The programs that we have already dealt with.

Packet Tracer which is a cross-platform visual simulation tool designed by CISCO systems that allows users to create network topologies and imitate modern computer networks. The software allows users to simulate the configuration of CISCO routers and switches using a simulated command line interface (CLI).

You can install this program from this link

https://www.cisco.com

4.7.3 Program setup.

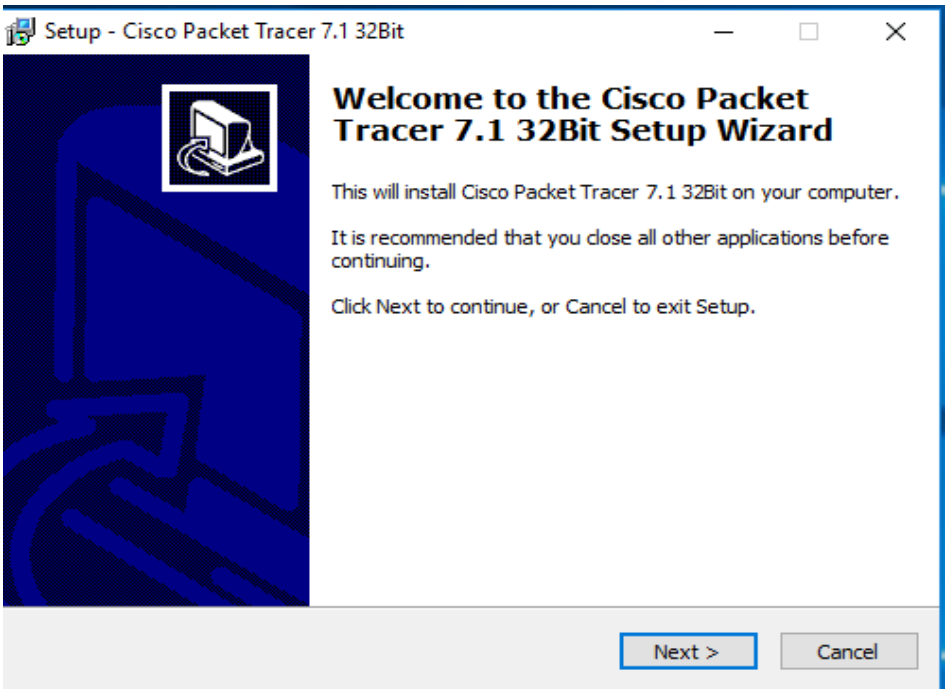


Figure 4.5: Step 1 to install packet tracer

After five steps (click next) you will get:

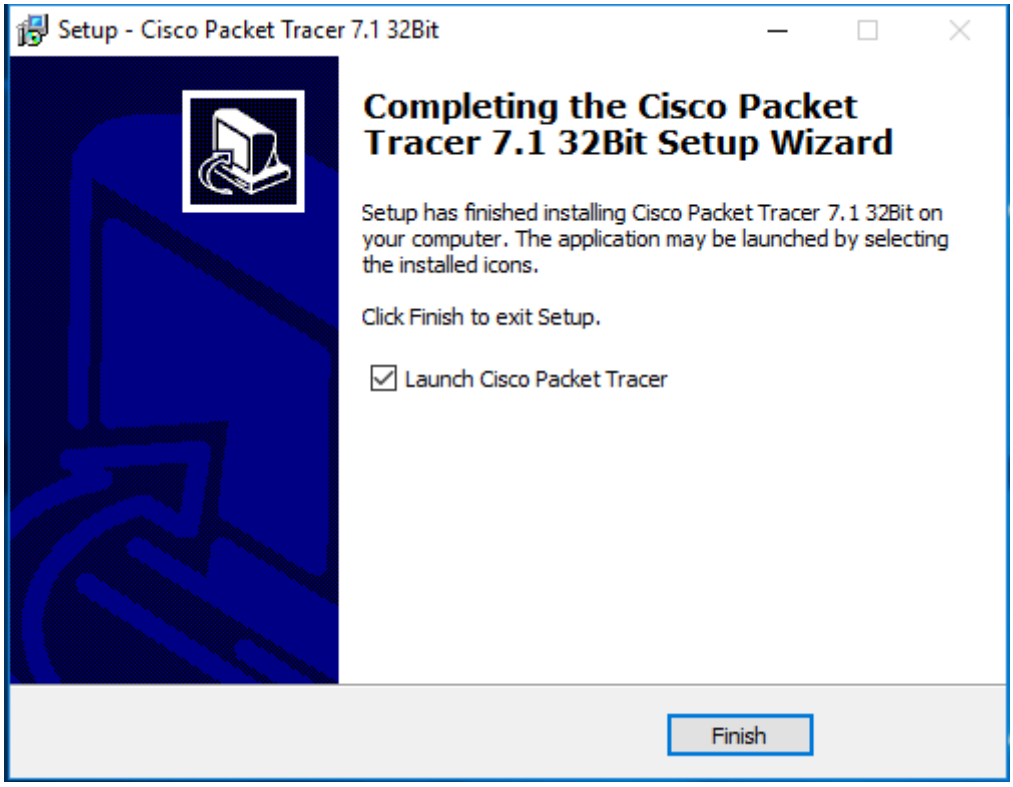


Figure 4.6: Last step to install packet tracer

You successfully installed cisco packet tracer.

Chapter 5

OpenFlow

5.1 Open Flow

OpenFlow is an open standard protocol specifically designed for the SDN networks that allows the communication between the control and data planes.

OpenFlow was originally imagined and implemented as part of network research at Stanford University. Its original focus was to allow the creation of experimental protocols on campus networks that could be used for research and experimentation. Prior to that, universities had to create their own experimentation platforms from scratch. What evolved from this initial kernel of an idea was a view that OpenFlow could replace the functionality of layer 2 and layer 3 protocols completely in commercial switches and routers. This approach is commonly referred to as the clean slate proposition. In 2011,a non-profit consortium called the Open Networking Foundation (ONF) was formed by a group of service providers to commercialize, standardize, and promote the use of OpenFlow in production networks. The ONF is a new type of Standards Development Organization in that it has a very active marketing department that is used to promote the OpenFlow protocol and other SDN-related efforts. The organization hosts an annual conference called the Open Networking Summit as part of these efforts. In the larger picture, the ONF has to be credited with bringing attention to the phenomenon of software-defined networks.

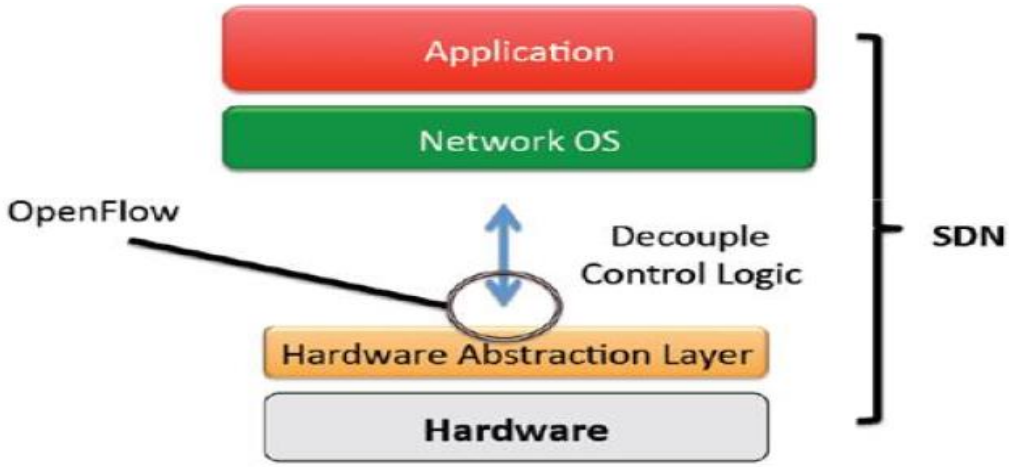


Figure 5.1: Open Flow Protocol.

5.2 OpenFlow Architecture

The OpenFlow network architecture consists of three basics concepts:

1. OpenFlow switches that compose the data plane.
2. The control plane consists of one or more OpenFlow controllers.
3. A secure control channel connects the switches with the control plane.

The switches communicate with the hosts and with each other using the data path software can provide and the controller communicates with the switches using the control path as shown in Figure 5.2 .

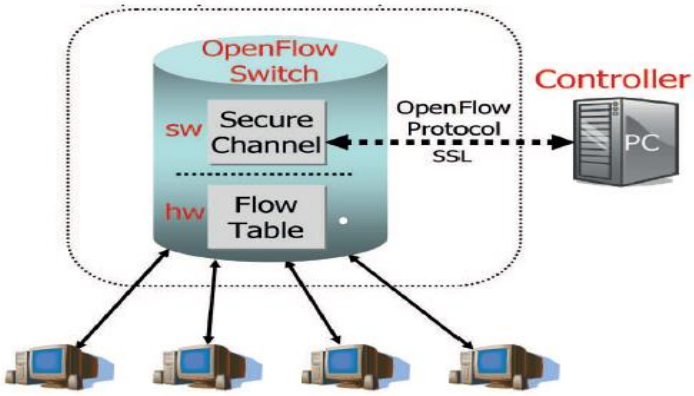


Figure 5.2: OpenFlow Network Architecture.

The connection between the OpenFlow controller and the switch is secured using Secure Sockets Layer (SSL) or Transport Layer Security (TLS) cryptographic protocols.

OpenFlow is based on a switching device with an internal flow table and provides an open, programmable, virtualized switching platform to control switch hardware via software.It can implement the function of a switch, router or even both, and enables the control path of a networking device to be controlled programmatically via OpenFlow protocol as shown in Figure 5.3.

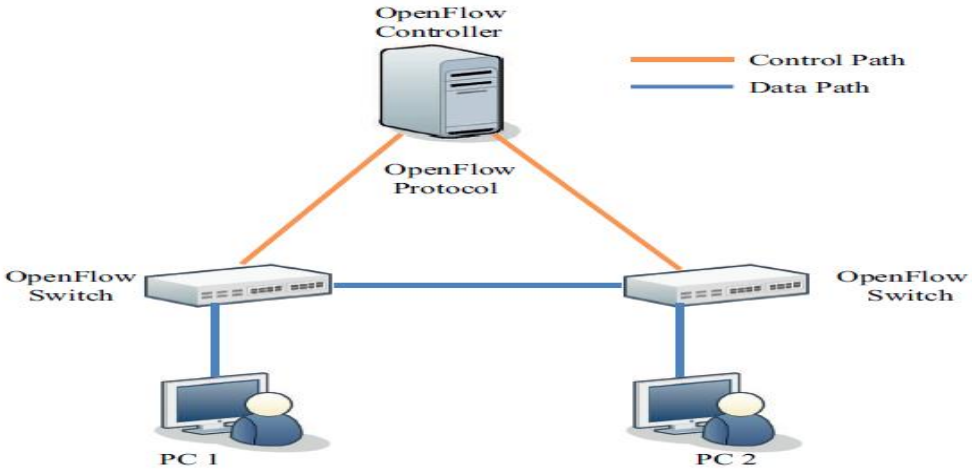


Figure 5.3: OpenFlow physical level Architecture.

5.3 Open Flow switch

5.3.1 Open Flow switch

The OpenFlow-enabled switch is the basic forwarding device that forwards the packets according to its flow table which is similar to traditional forwarding tables but not managed and maintained by the switch. The OpenFlow switch may contain one or more flow tables that perform packet lookups and forwarding. Each flow table in the switch contains a set of flow entries where each flow entry consists of match fields, counters, and a set of instructions or actions to be applied on the matched packets.

It makes forwarding decision by looking into its maintained flow table entries and finding an exact match on the specific fields of the incoming packets such as port number, source or destination IPv4 address, table entry, there resides an associated action that will be performed on the incoming source or destination MAC address, etc. For each flow table entry, there resides an associated action that will be performed on the incoming packets. For every incoming packet, the switch tries to find a matching entry through its flow table and forwards the packets based on the associated action.

If several matches are found, packets are matched against a specific flow entry based on prioritization, i.e., the flow entry with the highest priority is selected.

In case of the incoming packets flow entries do not match with the flow table of a switch, then, depending upon the

configuration of OpenFlow network. the switch sends them to the controller to make a further decision or continue them to next flow table.

The process of packet forwarding mechanism can be illustrated in the flowchart in Figure 5.4.

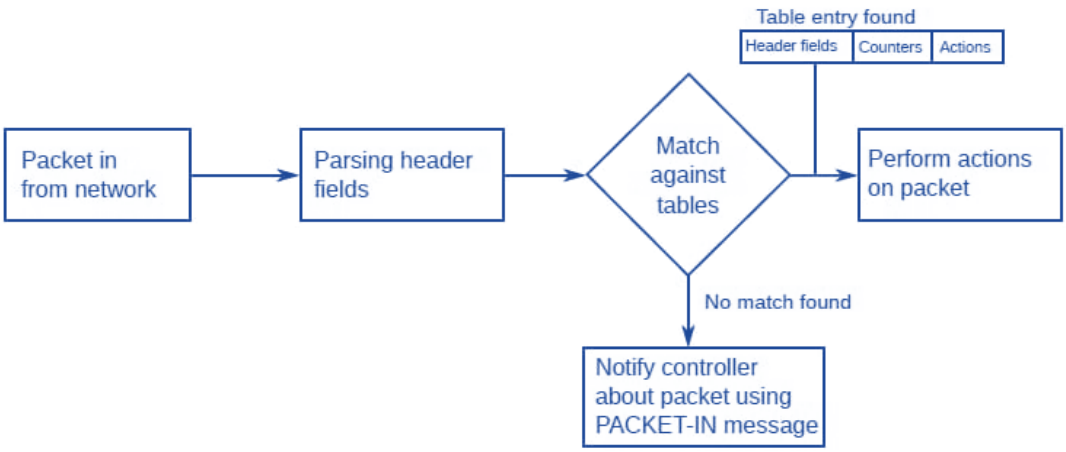


Figure 5.4: Basic Packet Process mechanism for OpenFlow switch.

5.3.2 open flow switch ports.

OpenFlow ports are the network interfaces for passing packets between OpenFlow processing and the rest of the network.

OpenFlow switches connect logically to each other via their OpenFlow ports. An OpenFlow switch makes a number of OpenFlow ports available for OpenFlow processing. The set of OpenFlow ports may not be identical to the set of network interfaces provided by the switch hardware, some network interfaces may be disabled for OpenFlow, and the OpenFlow switch may define additional Open-Flow ports. OpenFlow packets are received on an ingress port and processed by the OpenFlow pipeline which may forward them to an output

port. The packet ingress port is a property of the packet throughout the OpenFlow pipeline and represents the Open-Flow port on which the packet was received into the OpenFlow switch.

The ingress port can be used when matching packets. The OpenFlow pipeline can decide to send the packet on an output port using the output action, which defines how the packet goes back to the network.

5.3.3 Physical Ports

The OpenFlow physical ports are switch defined ports that correspond to a hardware interface of the switch. For example, on an Ethernet switch, physical ports map one-to-one to the Ethernet interfaces.

In some deployments, the OpenFlow switch may be virtualized over the switch hardware. In those cases, an OpenFlow physical port may represent a virtual slice of the corresponding hardware interface of the switch.

5.3.4 Logical ports

The OpenFlow logical ports are switch defined ports that don’t correspond directly to a hardware interface of the switch.

Logical ports are higher level abstractions that may be defined in the switch using non-OpenFlow methods (e.g. link aggregation groups, tunnels, loopback interfaces).

Logical ports may include packet encapsulation and may map to various physical ports. The processing done by the logical port must be transparent to OpenFlow processing and those ports must interact with OpenFlow processing like Open-Flow physical ports.

The only differences between physical ports and logical ports is that a packet associated with a logical port may have an extra metadata field called Tunnel-ID associated with it and when a packet received on a logical port is sent to the controller, both its logical port and its underlying physical port are reported to the controller.

5.3.5 Reserved ports

The OpenFlow reserved ports are defined by this specification. They specify generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods, such as” normal” switch processing.

ALL: Represents all ports the switch can use for forwarding a specific packet. Can be used only as an output port. In that case a copy of the packet is sent to all standard ports, excluding the packet ingress port and ports that are configured OFPPC NO FWD.

CONTROLLER: Represents the control channel with the OpenFlow controller Can be used as an ingress port or as an output port. When used as an output port, encapsulate the packet in a packet-in message and send it using the

OpenFlow protocol. When used as an ingress port, identify a packet originating from the controller.

TABLE: Represents the start of the OpenFlow pipeline. This port is only valid in an output action in the action list of a packet-out message, and submits the packet to the first flow table so that the packet can be processed through the regular OpenFlow pipeline.

IN PORT: Represents the packet ingress port. Can be used only as an output port, send the packet out through its ingress port.

ANY: Special value used in some OpenFlow commands when no port is specified (port wild carded). Cannot be used as an ingress port nor as an output port.

5.4 OpenFlow Tables

This section describes the components of flow tables, along with the mechanics of matching and action handling.

5.4.1 Flow Table

This section describes the components of flow table entries and the process by which incoming packets are matched against flow table entries.

OpenFlow protocol consists of three major fields: header fields, counters, and action that reside in every flow entry in a flow table. figure 5.5.

	Ethernet				IP				
Ingress port	Src	Dst	Type	VLAN ID	Src	Dst	Proto	Src port	Dst port

Figure 5.5: flow table

5.4.1.1 Header Fields

Header fields or match fields: These fields identify the flow by matching packets with certain fields which can be seen in Fig.5.6.

Figure 5.6: OpenFlow Packet Header Fields

5.4.1.2 Counters

Counters are maintained per-table, per-flow, per-port and per queue. OpenFlow compliant counters may be implemented in software and maintained by polling hardware counters with more limited ranges.

5.4.1.3 Actions

Each flow entry is associated with zero or more actions that dictate how the switch handles matching packets. If no forward actions are present, the packet is dropped. Action lists for inserted flow entries MUST be

processed in the order specified. However, there is no packet output ordering guaranteed within a port. For example, an action list may result in two packets sent to two different VLANs on a single port. These two packets may be arbitrarily re-ordered, but the packet bodies must match those generated from a sequential execution of the actions.

A switch may reject a flow entry if it cannot process the action list in the order

specified, in which case it should immediately return an unsupported flow error. Ordering within a port may vary between vendor switch implementations. A switch is not required to support all action types - just those marked ” Required Actions ” below. When connecting to the controller, a switch indicates which of the ”Optional Actions” it supports. OpenFlow-compliant switches come in two types: OpenFlow-only, and OpenFlow-enabled.

Figure 5.7: Fields from packets used to match against flow entries.

OpenFlow-only switches support only the required actions below, while OpenFlow enabled switches, routers, and access points may also support the **NORMAL Action:** Either type of switch can also support the FLOOD action.

Required Action: Forward. OpenFlow switches must support forwarding the packet to physical ports and the following virtual ones:

ALL: Send the packet out all interfaces, not including the incoming interface.

CONTROLLER: Encapsulate and send the packet to the controller

LOCAL: Send the packet to the switches local networking stack.

TABLE: Perform actions in ow table. Only for packet-out messages.

IN PORT: Send the packet out the input port.

Optional Action: Forward. The switch may optionally support the following virtual ports:

NORMAL: Process the packet using the traditional forwarding path supported by the switch (i.e., traditional L2, VLAN, and L3 processing) The switch may check the VLAN field to determine whether or not to forward the packet along the normal processing route. If the switch cannot forward entries for the OpenFlow specific VLAN back to the normal processing route, it must indicate that it does not support this action.

FLOOD: Flood the packet along the minimum spanning tree, not including the incoming interface.

The controller will only ask the switch to send to multiple physical ports simultaneously if the switch indicates it supports this behavior in the initial handshake.

Optional Action: Enqueue. The enqueue action forwards a packet through a queue attached to a port.

Forwarding behavior is dictated by the configuration of the queue and is used to provide basic Quality-of-Service (QoS) support.

Required Action: Drop. A flow-entry with no specified action indicates that all matching packets should be dropped.

Figure 5.8: shows an example for open flow table The first five columns represent the packet headers that can be matched (this is what defines a flow).

The column “Action” represents the action, defined by the controller, that the switch must perform when it matches on that row. Finally, the last column represents the number of packets received by the switch that matched that flow.

For example, we can see that all packages with Transmission Control Protocol (TCP) destination port 25 will be discarded and that the switch has already discarded 100 of those packets. The unknown packets (all the first 5 columns have only an *) are forwarded to the controller, which is the default behavior, the controller can then decide what action to perform to those packets.

Mac src	Mac dest	IP src	IP dest	TCP port	Action	Counter
*	10:20:*	*	*	*	Port 11	235
*	*	*	123.8.2.1	*	Port 2	300
*	*	*	*	25	Drop	100
*	*	*	*	*	Controller	455

Figure 5.8: flow table contents.

5.4.2 Pipeline Processing.

OpenFlow-compliant switches come in two types:

OpenFlow-only, and OpenFlow hybrid.

OpenFlow-only switches support only OpenFlow operation, in those switches all packets are processed by the OpenFlow pipeline, and cannot be processed otherwise.

OpenFlow-hybrid switches support both OpenFlow operation and normal Ethernet switching operation, i.e. traditional L2 Ethernet switching, VLAN isolation, L3 routing (IPv4 routing, IPv6 routing...), ACL and QoS processing. Those switches should provide a classification mechanism outside of OpenFlow that routes traffic to either the OpenFlow pipeline or the normal pipeline.

For example, a switch may use the VLAN tag or input port of the packet to decide whether to process the packet using one pipeline or the other, or it may direct all packets to the OpenFlow pipeline. This classification mechanism is outside the scope of this specification. An OpenFlow-hybrid switch may also allow a packet to go from the OpenFlow pipeline to the normal pipeline through the NORMAL and FLOOD reserved ports. The OpenFlow pipeline of every OpenFlow switch contains multiple flow tables, each flow table containing multiple flow entries.

The OpenFlow pipeline processing defines how packets interact with those flow tables (see Figures 5.9).An OpenFlow switch is required to have at least one flow table, and can optionally have more flow tables. An OpenFlow switch with only a single flow table is valid, in this case pipeline processing is greatly simplified.

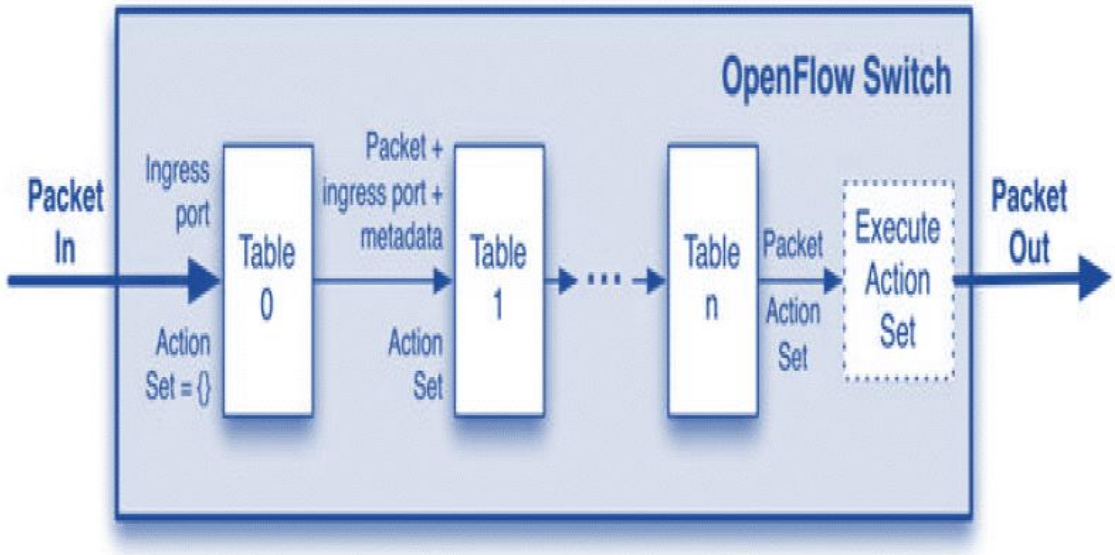


Figure 5.9: packets are matched against multiple tables in the pipeline.

The flow tables of an OpenFlow switch are sequentially numbered, starting at 0.

Pipeline processing always starts at the first flow table the packet is first matched against flow entries of flow table 0. Other flow tables may be used depending on the outcome of the match in the first table.

Figure 5.10: Packet flow through the processing pipeline.

When processed by a flow table, the packet is matched against the flow entries of the flow table to select a flow entry. If a flow entry is found, the instruction set included in that flow entry is executed, those instructions may explicitly direct the packet to another flow table (using the Go to Instruction, where the same process is repeated again. A flow entry can only direct a packet to a flow table number which is greater than its own flow table number, in other words pipeline processing can only go forward and not backward. Obviously, the flow entries of the last table of the pipeline cannot include the Go to instruction. If the matching flow entry does not direct packets to another flow table, pipeline processing stops at this table. When pipeline processing stops, the packet is processed with its associated action set and usually forwarded.

If a packet does not match a flow entry in a flow table, this is a table miss. The behavior on a table miss depends on the table configuration. A table-miss flow entry in the flow table may specify how to process unmatched packets: Options include dropping them, passing them to another table or sending them to the controller over the control channel via packet-in messages.

5.5 Flow types

Flows populated from the OpenFlow controller can be classified into types:

• **Reactive:** In this type, the controller is idle until it receives the first packet from the OpenFlow switch. The controller parses the incoming packet and then inserts a new flow entry in the switch's flow table. Each new flow entry needs a small additional setup time. If a connection between the controller and the switch fails and the switch does not have the ability to forward the packet as a traditional switch, it will be unable to forward the packets to the hosts.

• **Proactive:** In this type, the controller pre-installs the flow entries into the switch's flow table without the need to receive the first packet of the flow.

It does not require additional time for flow setup and in case of link failure between the controller and the switch, the traffic will not be disrupted.

Fig. 5.11 shows the methodology of the reactive and proactive flows for a simple Open-Flow topology.

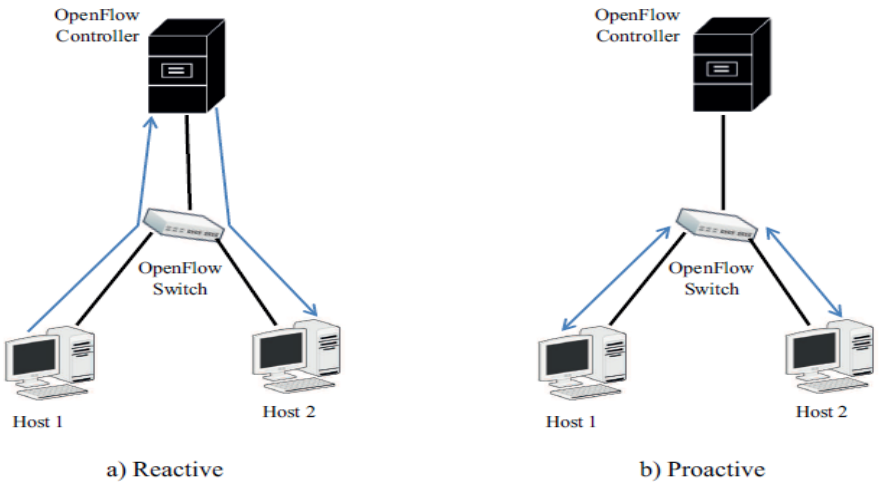


Figure 5.11: Reactive and Proactive Flows.

5.6 OpenFlow communication messages.

There are two classes of communication exist in the OpenFlow protocol.

5.6.1 Controller-to-switch.

Controller-to-switch messages are established by the controller and sent to the switch. There are different types of these massages:

• **Configuration:** This message enables the controller to set and query configuration parameters in the switch. The switch answers the query and sends the information needed to the controller.

• **Modify-State:** It is used to add/delete or modify flows in the flow tables and to set switch port properties.

• **Read-State:** These messages enable the controller to collect statistics from the switches' flow-tables, ports, and the individual flow entries.

• **Features:** Upon Transport Layer Security (TLS) session establishment, the controller sends a features request message to the switch. The switch must reply with a feature reply that specifies the capabilities supported by the switch.

• **Barrier:** Barrier request/reply messages are used by the controller to ensure message dependencies have been met or to receive notifications for completed operations.

• **Role-Request:** Role-Request messages are used by the controller to set the role of its OpenFlow channel, or query

that role. This is mostly useful when the switch connects to multiple controllers.

• **Asynchronous-Configuration:** The Asynchronous-Configuration message are used by the controller to set an additional filter on the asynchronous messages that it wants to receive on its OpenFlow channel, or to query that filter. This is mostly useful when the switch connects to multiple controllers and commonly performed upon establishment of the OpenFlow channel.

• **Send-Packet:** The controller uses these messages to send packets out of a specified port on the switch.

5.6.2 Asynchronous Messages

These messages are initiated by the switch and send to the controller. There are three main asynchronous messages:

• **Packet-in:** A packet-in message contains a packet to be sent to the controller. Either because it does not have a match for any flow entry in the switch flow table or it matches an action that orders the switch to send it to the controller.

• **Flow-Removed:** Flow table entries are added with hard timeout values by the controller’s flow-modification message. The hard timeout indicates the lifetime of a flow. After the expiration of this value, the flows are automatically removed. Those events are announced to the controller by Flow-Removed messages.

• **Port-status:** The switch is expected to send port-status messages to the controller as port configuration state changes. These events include change IN PORT status (for example, if it was brought down directly by a user) or a change in port status as specified by 802.1D.

• **Error:** The switch is able to notify the controller of problems using error messages.

5.6.3 Symmetric

Symmetric messages are sent without solicitation, in either direction.

• **Hello:** Hello messages are exchanged between the switch and controller upon connection start up.

• **Echo:** Echo request/reply messages can be sent from either the switch or the controller, and must return an echo reply. They are mainly used to verify the liveness of a controller-switch connection, and may as well be used to measure its latency or bandwidth.

• **Experimenter:** Experimenter messages provide a standard way for OpenFlow to offer additional functionality within the OpenFlow message type space. This is a staging area for features meant for future OpenFlow revisions.

5.7 Demonstration of the messages exchanged in Open- Flow network.

In order to demonstrate the messages in real OpenFlow network, we used Mininet network emulator to emulate two hosts connected to a switch and a controller as shown in Fig. 5.10.For this demonstration, we first explain the Switch-Controller connection establishment, then host-to-host communication through the OpenFlow switch and controller.

5.7.1 Message establishment between a switch and a controller.

When a switch connects to an OpenFlow network, it establishes a TCP handshake with the controller IP address and a default port of 6633. The session is established when the switch and controller successfully exchange Hello and echo request/reply messages to negotiate some parameters of the OpenFlow protocol such as OF version, bandwidth, latency, and liveness of their connection. After establishing the session, the controller sends the switch a feature request message to see which ports are available in the switch. The switch responds with a feature reply message that contains a list of ports, ports' speed, and the supported tables and actions. Fig. 5.12 illustrates the sequence of processing for the packets.

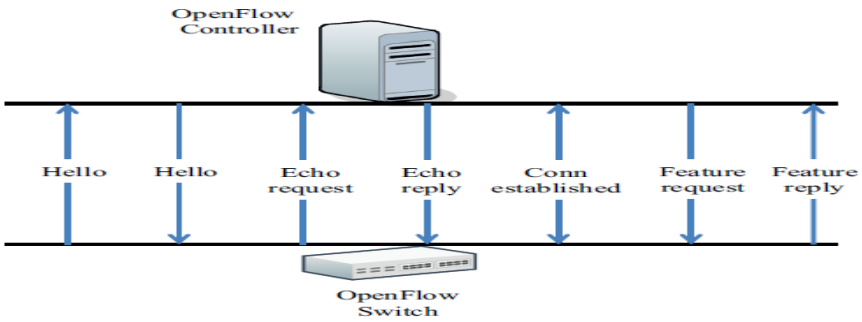


Figure 5.12: Communication messages between OpenFlow switch and controller.

5.7.2 Messages exchanged between two hosts

To demonstrate how the host-to-host connection is performed in an OpenFlow network, we used the Ping tool to send ICMP packets from h1 to h2 and vice versa.

The process starts when h1 sends an ARP request to the switch, asking for h2 MAC address, the switch does not know how to deal with this packet, therefore, it sends the packet as a PACKET-IN message to the controller.

The controller answers with a PACKET-OUT message that has an action instruct the switch to send the packet out to all ports except the incoming port (in this case just port 2) and wait for replying to that request.

When h2 replies for that request, the switch also sends that reply to the controller because it has no idea where to forward that packet. When the controller receives the ARP reply, it sends FLOW-MOD message to install a new flow entry for the future ARP replies from h2 that is destined to h1 to be forwarded directly by the switch without notifying the controller.

The same process happens when h1 sends the ICMP request/reply, and when h2 sends an ARP request for resolving h1 MAC address.

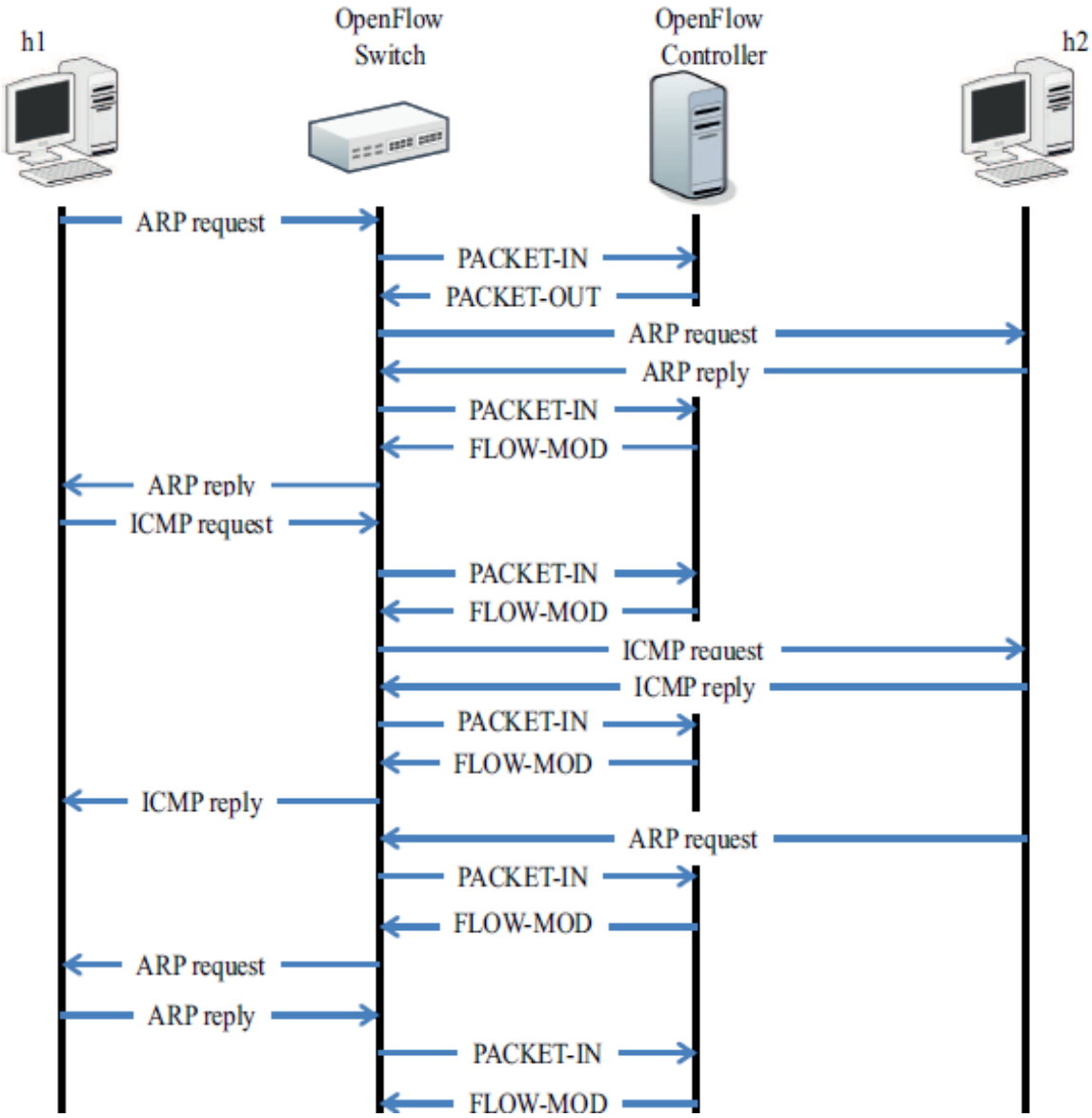


Figure 5.13: Ping process between h1 and h2.

5.8 OpenFlow controllers

The controller is considered as the heart and the main part in the OpenFlow network which makes network decisions. There are various types of OpenFlow controllers such as POX, Floodlight, OpenDaylight, Ryu, etc. Figure 5.14 shows the main difference between some of the open-source OpenFlow controllers based on their characteristics.

SDN controller is responsible for manipulating the switch's flow table, and for communications between applications and network devices using OpenFlow protocol.

Along with its primary function, it can further be extended to perform additional critical tasks such as routing, load balancing, network access.

The controllers can be classified into two main categories:

1. Open-source controller.
2. Commercial closed-source controller.

The Open-source controllers are available for research and development with the ability to develop various APIs on their platform to perform certain tasks. There are many Opensource OpenFlow controllers; the major distinction between them is the programming language they are written in. Below is a list of Open-source controllers based on their programming language:

- C: Trema (also Ruby).
- C++: NOX (also Python).
- Java: Beacon Floodlight and OpenDaylight

- Python: POX and RYU.

a controller runs on a network attached server and can serve one or multiple switches depending on the network design. It can be designed with centralized hierarchy where one controller handles and controls all the switches in a network or distributed hierarchy where two or more controllers handle and control two or more groups of switches in a network.

In a centralized hierarchy, if the controller fails then all network operations are interrupted and it poses a single point of failure in an OpenFlow network. In the distributed hierarchy, all the controllers should have the same copy of the network topology view in real-time to avoid the packet losses. The network topology view includes the switch level topology; the locations of users, hosts, and other network elements and services. The list below represents some types of Commercial OpenFlow controllers:

- Cisco Application Policy Infrastructure Controller (APIC)
- HP Virtual Application Networks (VAN) SDN Controller
- IBM Programmable Network Controller
- VMware NSX controller

	NOX	POX	Ryu	Floodlight	OpenDaylight
Language	C++	Python	Python	Java	Java
Developer	Nicira	Nicira	NTT, OSRG group	BigSwitch	Linux foundation
OpenFlow	1.0, 1.1, 1.2, 1.3	1.0	1.0, 1.1, 1.3, 1.4, 1.5	1.0	1.0, 1.3
Throughput in Kilo flow/sec	30	30	33	15	10
Distributed	No	No	Yes	Yes	Yes
Learning curve	Moderate	Easy	Moderate	Strong	Strong
Well-documented	No	No	Yes	Yes	Yes

Figure 5.14: OpenFlow controller comparison.

Chapter 6

Mininet

6.1 What's Mininet?

Mininet is a network emulator. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. A Mininet host behaves just like a real machine and run arbitrary programs (including anything that is installed on the underlying Linux system.) The programs you run can send packets through what seems like a real Ethernet interface, with a given link speed and delay. Packets get processed by what looks like a real Ethernet switch, router. and of Course, it supports many OpenFlow features In short, Mininet's virtual hosts, switches, links, and controllers are the real thing they are just created using software rather than hardware - and for the most part their behavior is similar to discrete hardware elements. It is usually possible to create a Mininet network that resembles a hardware network, or a hardware network that resembles a Mininet network, and to run the same binary code and applications on either platform.

6.2 Why Mininet?

- It is fast - starting up a simple network takes just a few seconds.
- You can run real programs: anything that runs on Linux is available for you to run, from web servers to TCP window monitoring tools to Wireshark.

- You can customize packet forwarding: Mininet's switches are programmable using the OpenFlow protocol. Custom Software-Defined Network designs that run in Mininet can easily be transferred to hardware OpenFlow switches.
- You can run Mininet on your laptop, on a server, in a VM, on a native Linux box (Mininet is included with Ubuntu 12.10+) or in cloud.
- You can use it easily: you can create and run Mininet experiments by writing simple (or complex if necessary) Python scripts.
- Mininet is an open source project, so you are encouraged to examine its source code on https://github.com/mininet, modify it, fix bugs, file issues/feature re-quests, and submit patches/pull requests.

6.3 How to install Mininet?

To install Mininet 2.2 Beta on a new virtual machine, we will execute the following steps:

1. Install Ubuntu Server 14.04 on a new VirtualBox virtual machine image
2. Update the Ubuntu Server system and install required software
3. Install Mininet 2.2 on the Ubuntu Server virtual machine

- **Install Ubuntu Server on a VM:**
  Download the Ubuntu Server 18.04 ISO image from ubuntu.com. Create a new virtual machine in VirtualBox and install Ubuntu Server on it.

When installing Ubuntu Server, you can use all the default configurations and choose setting specific to your needs that will be obvious to you (like country, keyboard layout, and time zone). In our example, I chose to name the server *mininet-2-2* and to create a user id *brain* so all the following examples will use these names. You may choose different names.

- **Update the Ubuntu Server VM:**
  First download and install system updates

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get dist-upgrade
```

- **Install required software:**
  Install git. Git is the software version control system used by the Mininet project.

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

- **Install Mininet 2.2 from source code:**
  Use git to download the Mininet 2.2.0 source code.

```
$ git clone git://github.com/mininet/mininet
```

6.4 Inside the Virtual Machine:

• **Openvswitch:** Virtual switch programmable using OpenFlow.

• **Mininet:** Network emulation platform.

```
$sudo mn –topo single,3 –mac –switch ovsk – controller
```

• **Wireshark:** Graphical tool for viewing packets with OF protocol plug-in. Start Wireshark: sudo Wireshark ,Start capture packets going through interface ”lo” and Decode as OFP

• **dpctl:** Command-line utility for checking switch status and manually inserting flow entries.

• **Check supported commands in manual:** man dpctl

• Multiple OpenFlow controllers with sample apps prepackaged NOX, POX, Ryu, and OpenDayLight.

6.5 Mininet topologies:

6.5.1 Mininet default topologies

• **Setup 1:** Mininet-based Single Switch

```
$ sudo mn –topo single,3 –mac –controller remote –switch ovsk
```

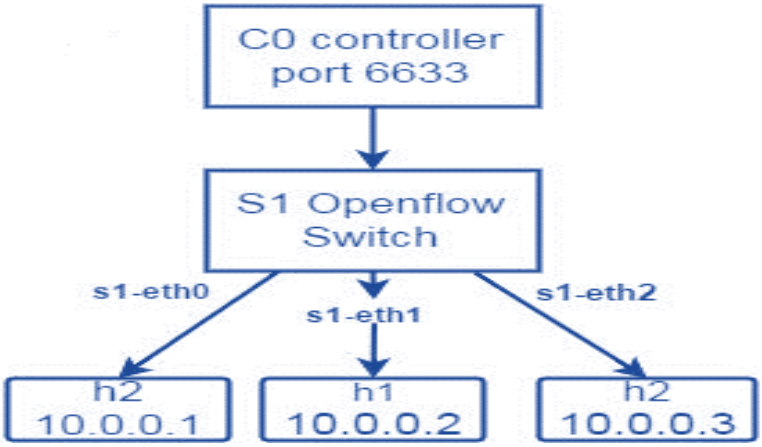


Figure 6.1: Simple topology by single switch using Mininet

• **Setup 2:** Linear topology with 2 switches

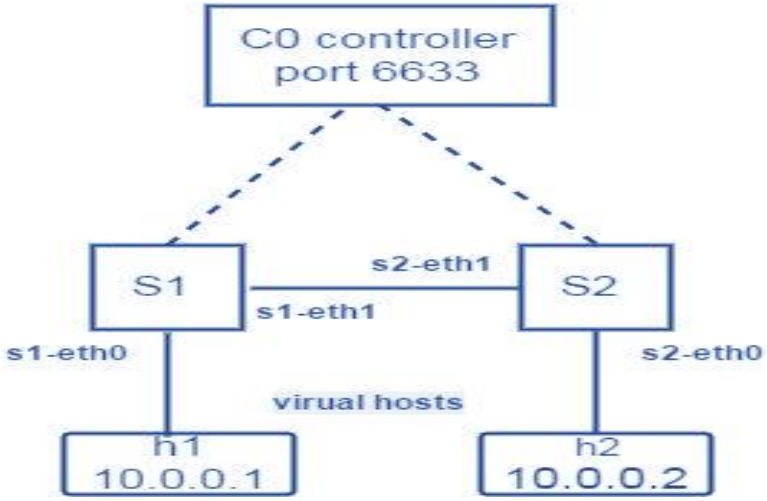


Figure 6.2: Linear topology by 2 switches using Mininet

6.5.2 Mininet Custom Topologies:

we use Mininet from which we create three virtual hosts and one OpenFlow switch with the remote controller by using the following command:

```
$sudo mn --topo=single,3 --mac --controller=remote
switch=ovsk,Protocols=OpenFlow1
```

6.6 Miniedit

The Mininet network simulator includes MiniEdit, a simple GUI editor for Mininet. MiniEdit is an experimental tool created to demonstrate how Mininet can be extended.

6.6.1 Start Miniedit

The MiniEdit script is located in Mininet's examples folder. To run MiniEdit, execute the command:

```
$ sudo ~/mininet/examples/miniedit.py
```

6.6.2 Miniedit user interface

MiniEdit has a simple user interface that presents a canvas with a row of tool icons on the left side of the window, and a menu bar along the top of the window.

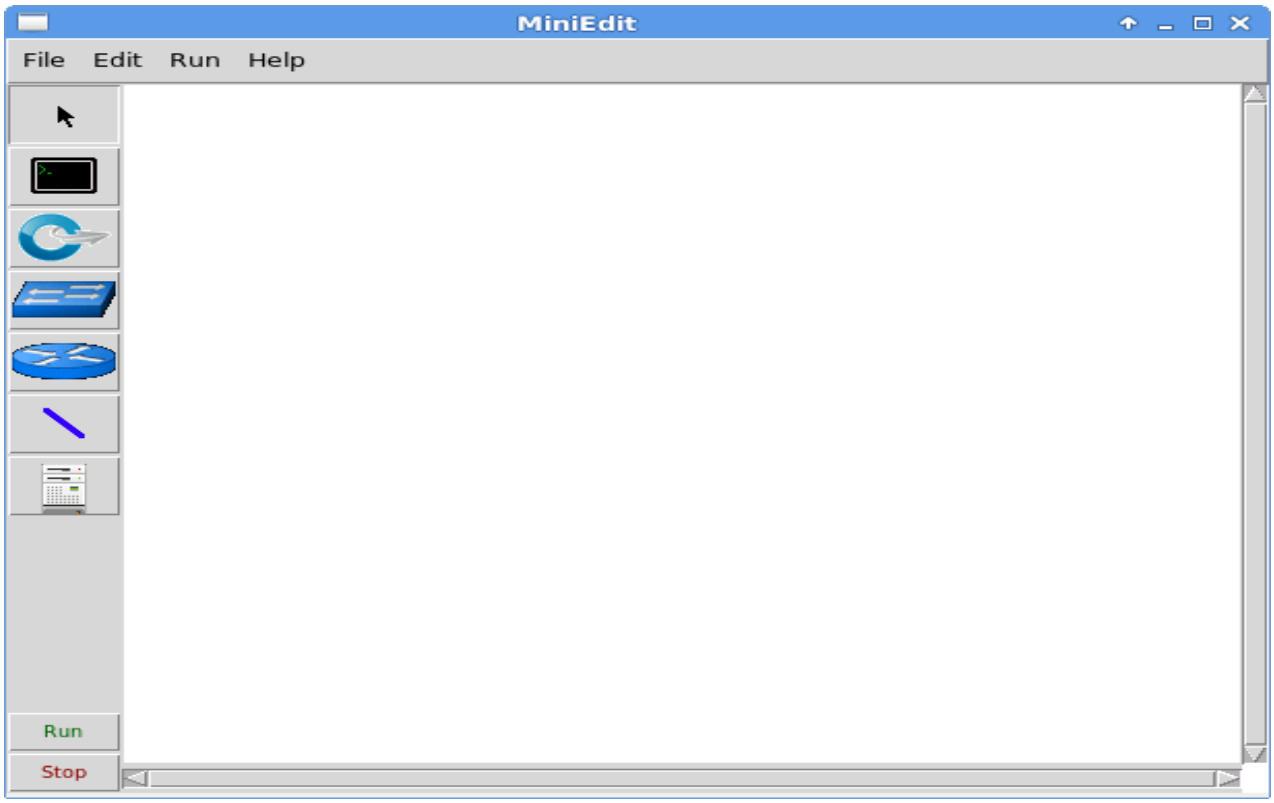


Figure 6.3: MiniEdit GUI

- **The icons represent the following tools**:

  The Select tool is used to move nodes around on the canvas. Click and drag any existing node.

  Interestingly the Select tool is not needed to select a node or link on the canvas. To select an existing node or link, just hover the mouse pointer over it — this works regardless of the tool that is currently active — and then either right-click to reveal a configuration menu for the selected element or press the Delete key to remove the selected element.

 The Host tool creates nodes on the canvas that will perform the function of host computers. Click on the tool, then click anywhere on the canvas you wish to place a node. As long as the tool remains selected, you can keep adding hosts by clicking anywhere on the canvas. The user may configure each host by right-clicking on it and choosing Properties from the menu.

 The Switch tool creates OpenFlow-enabled switches on the canvas. These switches are expected to be connected to a controller. The tool operates the same way as the Hosts tool above. The user may configure each switch by right-clicking on it a choosing Properties from the menu.

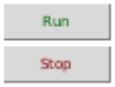
 The Legacy Switch tool creates a learning Ethernet switch with default settings. The switch will operate independently, without a controller. The legacy switch cannot be configured and is set up with Spanning Tree disabled, so do not connect legacy switches in a loop.

 The Legacy Router tool creates a basic router that will operate independently, without a controller. It is basically just a host with IP Forwarding enabled. The legacy router cannot be configured from the MiniEdit GUI.

 The NetLink tool creates links between nodes on the canvas. Create links by selecting the NetLink tool, then clicking on one node and dragging the link to the target

node. The user may configure the properties of each link by right-clicking on it a choosing Properties from the menu.

 The Controller tool creates a controller. Multiple controllers can be added. By default, the MiniEdit creates a Mininet OpenFlow reference controller, which implements the behavior of a learning switch. Other controller types can be configured. The user may configure the properties of each controller by right-clicking on it a choosing Properties from the menu.



The Run starts Mininet simulation scenario currently displayed in the MiniEdit canvas. The Stop button stops it. When MiniEdit simulation is in the “Run” state, right-clicking on network elements reveals operational functions such as opening a terminal window, viewing switch configuration, or setting the status of a link to “up” or “down”.

6.6.3 Create a custom network topology using MiniEdit

First, we will add some hosts to network scenario. Click on the *Host* icon, then move the pointer to the location on the MiniEdit canvas where you want the host to appear, then click again. A host icon will appear on the canvas.
As long as the *Host* tool is active, you can add more hosts. Keep clicking at each spot on the canvas where you want a host to appear. In this example, we will add ten hosts.

Add eight switches and three controllers using the same method: Click on the *Switch* tool and add switches, then click on the *Controller* tool and add controllers.

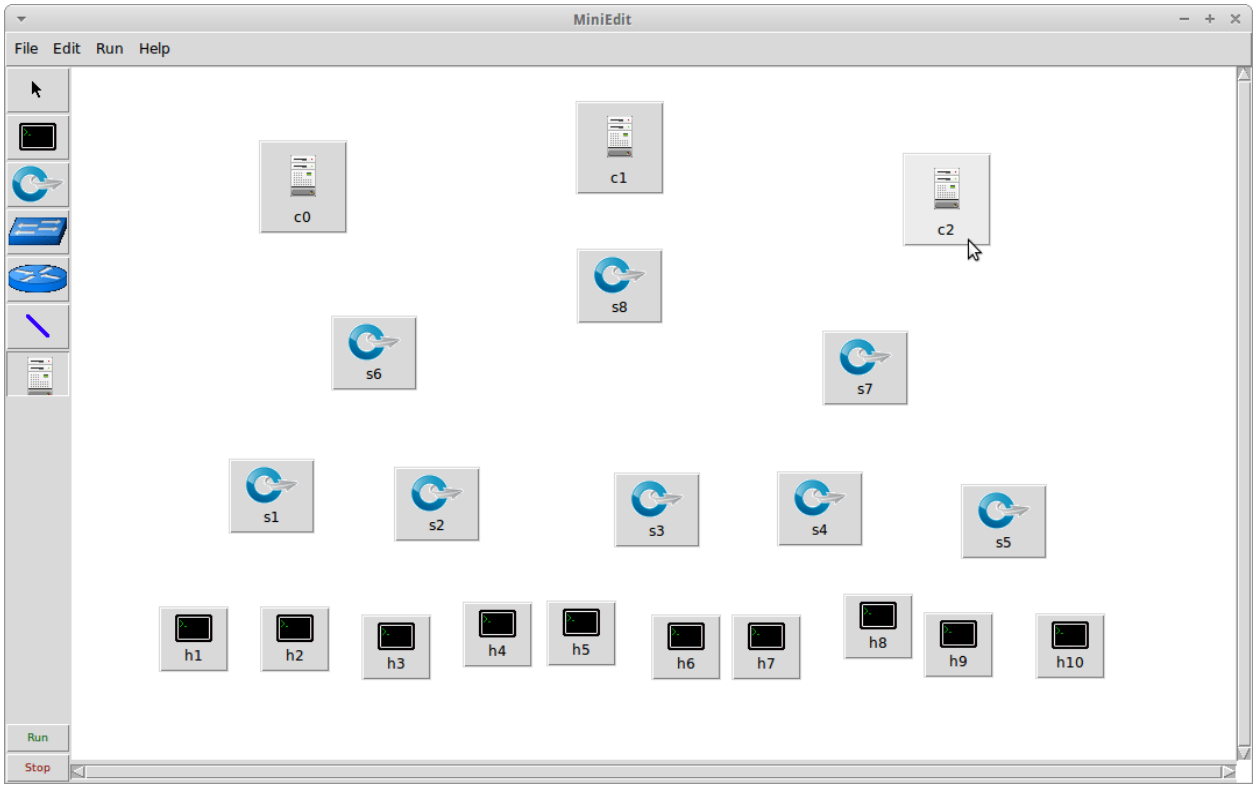


Figure 6.4: adding controllers, switches and Pcs using miniedit

Next, add links between the nodes on the canvas. Click on the *NetLink* tool, then click on a node and drag the link over to another node. For example: connect a host to a switch, or a switch to another switch. Connect every host to at least one switch. Connect the switches together to create a network. Then, connect each switch to one of the controllers. Your completed network should be similar to the network show in the screenshot below:

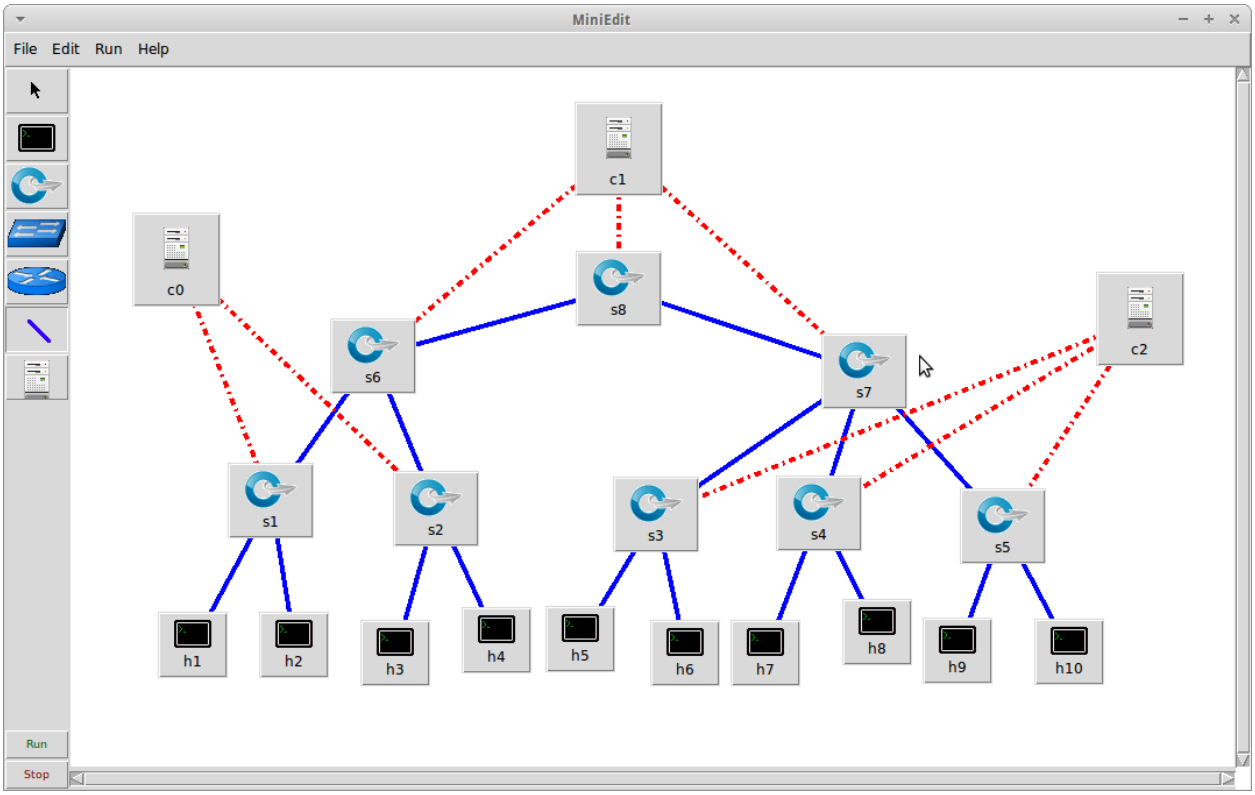


Figure 6.5: Connect controllers, switches and Pcs By Links using miniedit

We were able to create this complex custom network topology in a few minutes using MiniEdit. Manually writing a Mininet custom topology script to create this scenario would have taken a lot longer.

- **Configure the controllers**

We have three controllers. In this basic example, we will use the default OpenFlow Reference controller that comes built in to Mininet. However, we need to configure each controller so it uses a different port.

Right-click on each controller and select *Properties* from the menu that appears. The default port number for each controller is 6633. Change this so the port numbers used by controllers *c0*, *c1*, and *c2* are 6633, 6634, and 6635, respectively.

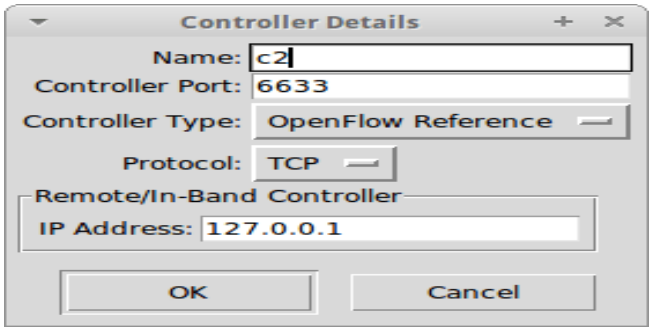


Figure 6.6: Configure the controllers

- **Set MiniEdit Preferences**

To set MiniEdit preferences, use the MiniEdit menu command, *Edit → Preferences*. In the dialogue box that appears, make the changes you need.

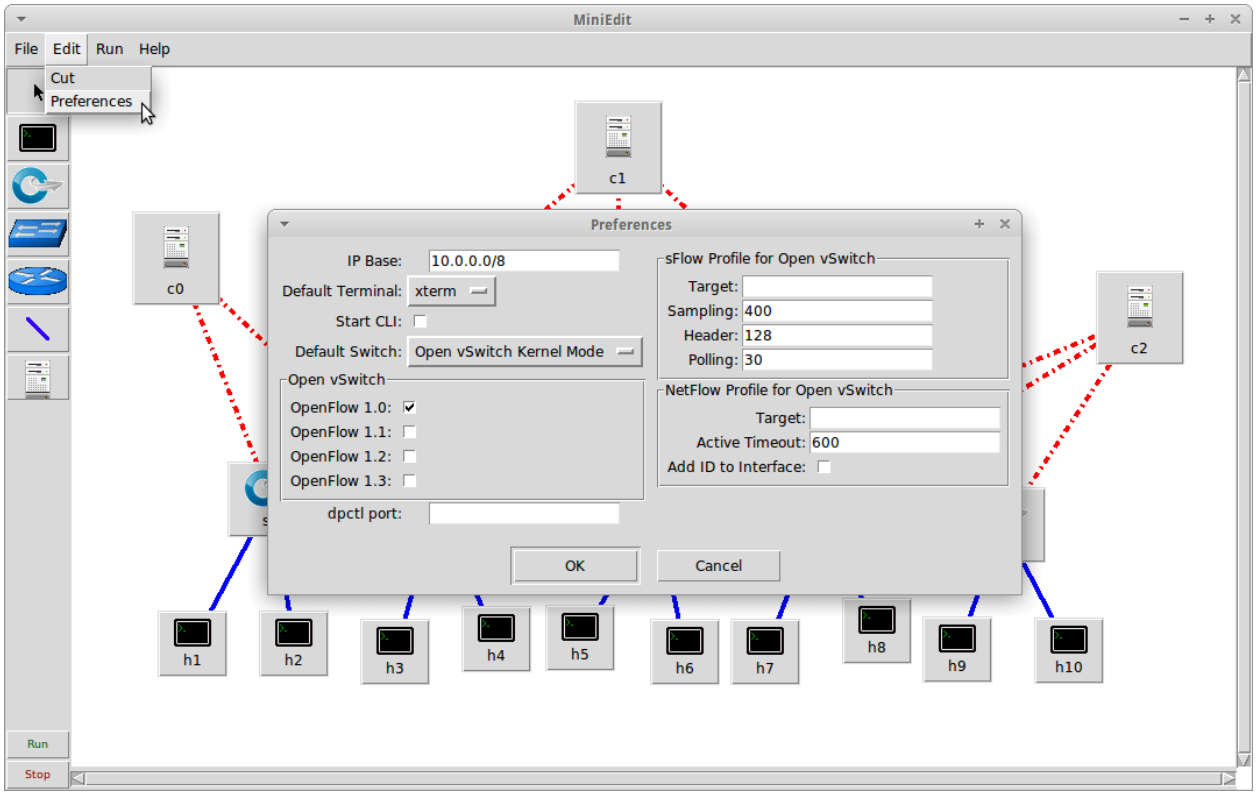


Figure 6.7: Set MiniEdit Preferences

- **Set the Start CLI Option Preferences**

By default, the MiniEdit console window does not give the user access to the Mininet command line interface. If you want to be able to use the Mininet CLI when a simulation is running, check the *Start CLI* box. You may also set the version of OpenFlow you will use in our scenario; we will use the CLI and will leave all other settings at default values.

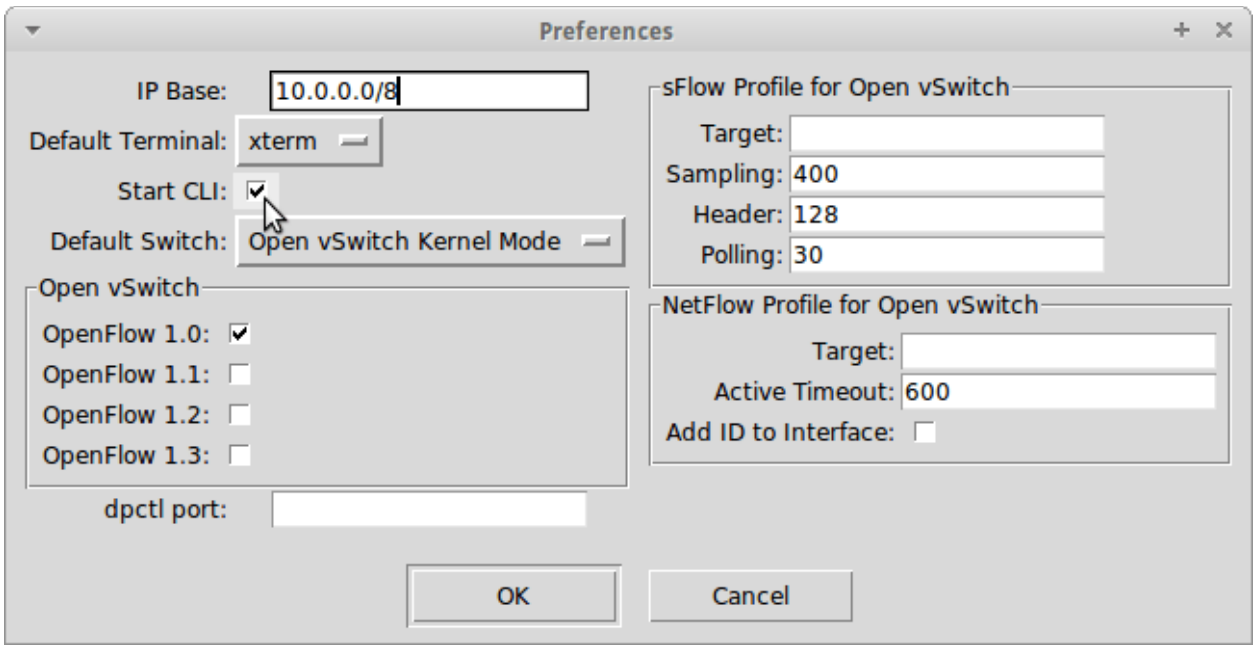


Figure 6.8: Set the Start CLI Option Preferences

- **Save the configuration**

Now we have a software-defined network scenario that should allow each host to communicate with any other host in the network.

We will save the MiniEdit topology file so we can load this scenario into MiniEdit in the future. We will also export a Mininet Python script that we can run in a terminal window to execute the scenario.

- **Save topology file**

To save the Mininet Topology (*.mn) file, click on File in the top menu bar and select Save from the drop-down menu. Type in a file name and save the file.

- **Save custom Mininet script**

To save the Mininet Custom Topology (*.py) file, click on File in the top menu bar and select Save Level 2 Script from the drop-down menu. Type in the file name and save the file.

- **Run the MiniEdit network scenario**

To start the simulation scenario, click the Run button on the MiniEdit GUI. In the terminal window from which you started MiniEdit, you will see some messages showing the progress of the simulation startup and then the Miniedit CLI prompt (because we checked Start CLI box in the MiniEdit preferences window).

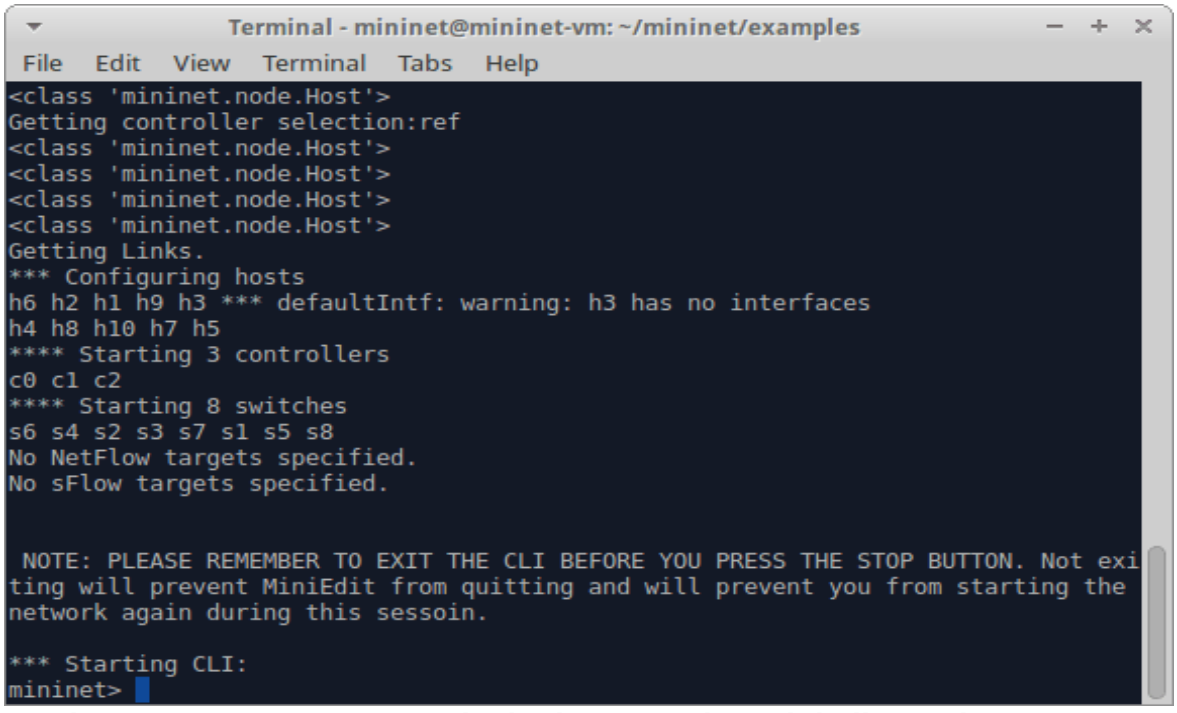


Figure 6.9: Output on Terminal When Running Miniedit topology

- **Experiment with the network**

After starting the simulation scenario, we will view the status of different elements in the network, open terminal windows, run network traffic, run programs on simulated hosts, and simulate network failures. These activities will demonstrate how to use some of the features of MiniEdit.

- **View Open vSwitch configurations**

First, check the switch configurations in the network simulation to verify that everything is set up correctly. You can run the MiniEdit menu command, *Run → Show OVS Summary* to see an listing of switch configurations. In this case, we can verify that each switch is listening to the correct controller on the correct port.

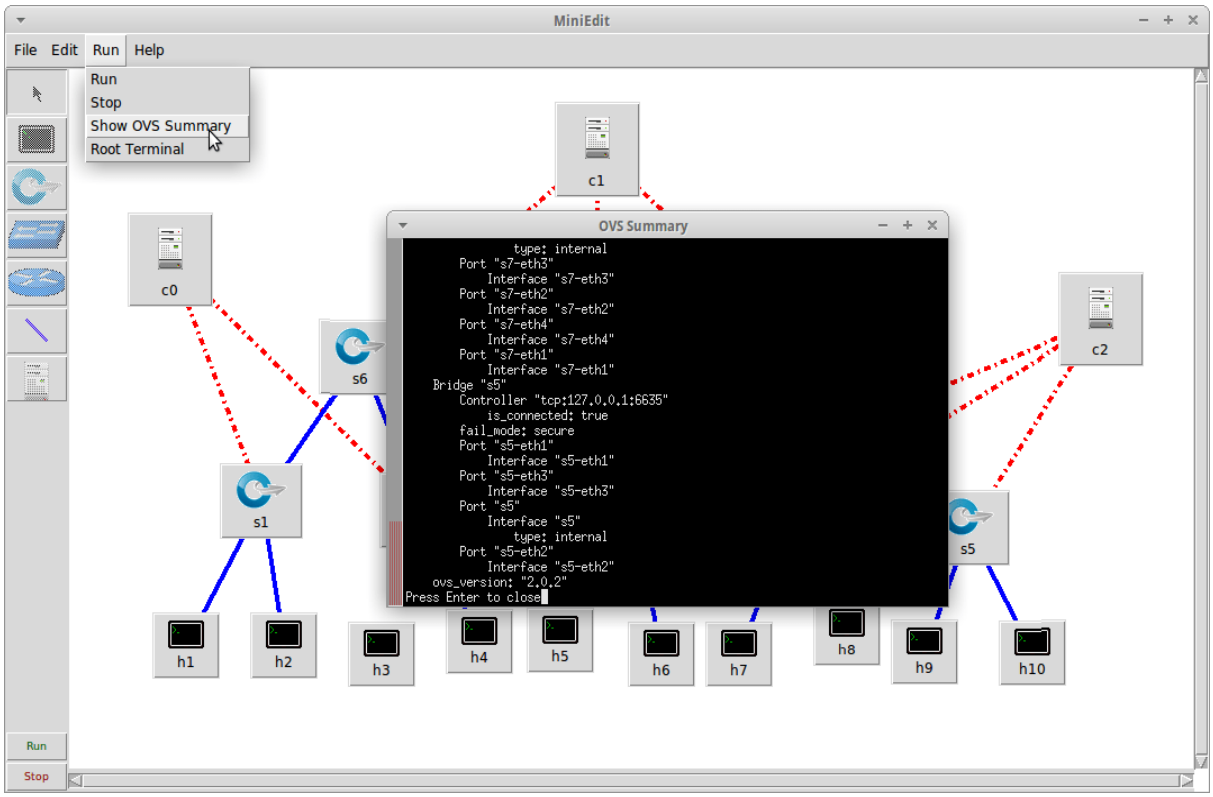


Figure 6.10: OVS Summary

Chapter 7

SDN Controllers

7.1 Introduction

The three most resonant concepts of SDN are programmability, the separation of the control and data planes, and the management of ephemeral network state in a centralized control model, regardless of the degree of centralization. The SDN controller is the embodiment of the idealized SDN frame-work, and in most cases, is a reflection of the framework.

In theory, an SDN controller provides services that can realize a distributed control plane, as well as abet the concepts of ephemeral state management and centralization. In reality, any given instance of a controller will provide a slice or subset of this functionality, as well as its own take on these concepts. In this chapter, we will detail the most popular SDN controller. Throughout the chapter, we have included embedded graphics of the idealized controller/framework that was just mentioned as a means to compare and contrast the various implementations of controllers. We have also included text that compares the controller type in the text to that ideal vision of a controller.

7.2 The general description of an SDN controller:

The general description of an SDN controller is a software system or collection of systems that together provides:

- Management of network state, and in some cases, the management and distribution of this state, may involve a database .These databases serve as a repository for information derived from the controlled network elements and related software as well as information controlled by SDN application including network state, some ephemeral configuration information, learned topology, and control session information). In some cases, the controller may have multiple, purpose-driven data management processes (e.g., relational and nonrelational databases). In other cases, other in-memory database strategies can be employed, too.
- A high-level data model that captures the relationships between managed re-sources, policies and other services provided by the controller. In many cases, these data models are built using the Yang modeling language.
- A modern, often RESTful (representational state transfer) application programming interface (API) is provided that exposes the controller services to an application. This facilitates most of the controller-to-application interaction. This interface is ideally rendered from the data model that describes the services and features of the controller. In some cases, the controller and its API are part of a development environment that generates the API code from the model. Some systems go further and provide robust development environments that allow expansion of

core capabilities and subsequent publishing of APIs for new modules, including those that support dynamic expansion of controller capabilities

- A secure TCP control session between controller and the associated agents in the network elements
- A standards-based protocol for the provisioning of application-driven network state on network elements
- A device, topology, and service discovery mechanism; a path computation system and potentially other network-centric or resource-centric information services

7.3 Top 3 features in most controllers

Each controller must have these features whatever the programming language.

1. Event-driven model

- Each module registers listeners or call-back functions.
- Example async events include PACKET IN, PORT STATUS, FEATURE REPLY, STATS REPLY

2. Packet parsing capabilities

- When switch sends an OpenFlow message, module extracts relevant information using standard procedures.

3. Switch send (msg), where msg can be.

- PACKET OUT with buffer id or fabricated packet
- FLOW MOD with match rules and action taken
- FEATURE REQUEST, STATS REQUEST, BARRIER REQUEST

7.4 How to choose the right SDN controller platform

Ten Things to Look for in an SDN Controller

1. **OpenFlow Support**

   OpenFlow is the first standard communications interface defined between the control and forwarding layers of an SDN architecture.

2. **Network Virtualization**

   Virtualization techniques such as VLANs and VRF are helpful, they are limited both in scope and in value. To add maximum value, network virtualization must be end-to-end and it must abstract and pool network resources in a manner sim-ilar to how server virtualization abstracts and pools compute resources. These capabilities enable the creation of tenant-specific virtual networks whose topology is decoupled from the topology of the underlying physical network, and it also enables IT organizations to dynamically create policy-based virtual networks to meet a wide range of requirements.

   One of the many advantages of decoupling the virtual networks from the physical networks is that it enables IT organizations to make changes to the physical network, such as scaling out capacity, without impacting the existing flows. In similar fashion, one of the many advantages of having the network virtualization be tenant-specific is that it enables complete isolation between each tenant.

3. Network Functionality

- Keep Virtual network totally isolated
- This includes the ability to discover multiple paths from origin to destination and to split the traffic across multiple links.

4. Scalability

- The number of switch that a controller can support
- The ability to create an SDN that spans multiple sites
- An SDN controller should be able to support a minimum of 100 switches.

5. Programmability

- Ability to apply sophisticated filter
- The SDN controller should provide templates that enable the creation of scriptable CLIs that allow for the dynamic programming of the network

6. Reliability

- Reliability: is the ability of the controller to work for a long time under an average workload without accidentally closing connections with switches or dropping packets from the switches.
- Hardware - software redundancy, hot swappable fans and power.

  the availability of external connections, it is important that the controller support technology and design alternatives, such as:

  i. the Virtual Router Redundancy Protocol (VRRP)
  ii. Multi-Chassis Link Aggregation Group (MC-LAG)

7. Performance

- The two of the key performance metrics:
  i. The flow setup time.
  ii. The number of flows per second.
- Flows can be setup in one of two ways :
  i. Reactively.
  ii. Proactively
- The key factors that affect flow setup time include:
  i. The processing power of the switches that are attached to the controller
  ii. I/O performance of the Controller.

8. Security

- sophisticated filter Keep Virtual network separate
- Support authentication of users
- Support security applications such DDoS protection

9. Centralized Monitoring and Visualization

- Monitor some classes of traffic and not others
- Visualize physical network and the virtual networks

10. Vendor characteristics

- Commitment
- Stability
- Engineering depth and prowess
- Relationships

7.5 list of SDN controllers

There are two types of SDN Controllers:

a. open-source controllers
b. Commercial Controllers

The following is a list of SDN Controllers, both commercial and open source.

7.5.1 open-source controllers

i. **Floodlight**

The Floodlight Open SDN Controller is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller. It is supported by a community of developers including a number of engineers from Big Switch Networks.

ii. **OpenContrail**

OpenContrail is an Apache 2.0-licensed project that is built using standards-based protocols and provides all the necessary components for network virtualization-SDN controller, virtual router, analytics engine, and published northbound APIs.

iii. **OpenDaylight Project**

The adoption of new technologies and pursuit of programmable networks has the potential to significantly improve levels of functionality, flexibility and adapt-ability of mainstream datacenter architectures. To leverage this abstraction to its fullest.

iv. **Protocol Oblivious Forwarding (POF) Controller**
Control to support POF, based on Floodlight with POF extensions.

v. **Standard University Beacon**
Beacon is a fast, cross-platform, modular, Java-based OpenFlow controller that supports both event-based and threaded operation.

vi. **Ryu OpenFlow Controller**
Ryu provides REST API by which Quantum server tells necessary information. Quantum Server manages the association networks to actual key value in addition to normal Quantum management information.

7.5.2 Commercial Controllers

There are a lot of this type of controller, the flowing are examples of them:

i. Active Resource Controller
ii. Calsoft Labs Virtual WLAN Controller
iii. Cisco Application Policy Infrastructure Controller (APIC)
iv. Cisco Application Policy Infrastructure Controller (APIC) Enterprise Module
v. CPLANE NETWORKS Controller
vi. Embedded Systems Software Development in Networking
vii. Extreme Networks Slalom
viii. HP Virtual Application Networks (VAN) SDN Controller
ix. IBM Programmable Network Controller
x. Inocybe Infrastructure Controller
xi. Italtel Session Border Controller

xii. Metaswitch Perimeta Session Border Controller

xiii. NCL Hinemos

7.6 HPE VAN

7.6.1 Introduction

1. HPE VAN definition

The HPE VAN SDN Controller provides a unified control point in an OpenFlow-enabled network, simplifying management, provisioning, and orchestration and enabling delivery of a new generation of application-based network services.

In the Hewlett Packard Enterprise Software Defined Networking (SDN) architecture, the control and data planes of the network are decoupled from each other, centralizing network intelligence and abstracting the underlying network infrastructure from applications. Controller software directly provisions physical and virtual switches under its control via the industry-standard OpenFlow protocol. Network ports, links, and topologies are all directly visible, enabling centralized policy administration and more effective path selection based on a dynamic, global view of the network. This dramatically simplifies the orchestration of multi-tenant environments and the enforcement of network policy for both mobile clients and servers,

The HPE VAN SDN Controller is designed to operate in a variety of computing environments, including campus, data center, service provider, private cloud, and public cloud.

2. HPE VAN controller features

- An enterprise-class platform for the delivery of a broad range of network innovations
- An extensible, scalable, and resilient controller architecture
- Compliance with OpenFlow 1.0 and 1.3 protocols
- Support for Hewlett Packard Enterprise and H3C OpenFlow-enabled switches
- Secure authentication using a local or remote Keystone server
- Controller teaming for distributed platform High Availability (HA) and scalability
- Embedded applications that provide common network services
- Open APIs enable SDN application developers to deliver innovative solutions that dynamically link business requirements to network infrastructure using either custom Java programs or general-purpose RESTful control interfaces.
- Integration with HPE Intelligent Management Center (IMC). HPE IMC provides full controller application life cycle management and monitoring, enhanced reporting and SDN network visualization.

3. List of controller embedded applications

The HPE VAN SDN Controller includes a default set of core network service applications that are installed as modules on the controller. The following applications are embedded in the controller and are installed when you install the controller:

- OpenFlow Link Discovery
- OpenFlow Node Discovery
- Path Diagnostics
- Topology Manager
- Topology Viewer

4. Accessing the controller

Interface	URL
Controller console UI	**https://SDN_CONTROLLER_ADDRESS:8443/sdn/ui**
REST API	**https://SDN_CONTROLLER_ADDRESS:8443/api**
REST API JSON data model	**https://SDN_CONTROLLER_ADDRESS:8443/sdn/v2.0/models**

Table 7.1: Controller URLs

SDN_CONTROLLER_ADDRESS is the IP address for your controller. The URI is case sensitive.

For detailed information about logging into the controller UI, see the HPE VAN SDN Controller Administrator Guide.

7.6.2 Downloading HPE VAN

1. Downloading the controller software

In the HPE VAN SDN Controller release 2.6, you can select the following downloads:

- HPE VAN SDN Controller virtual appliance
- HPE VAN SDN Controller Toolkit virtual appliance
- HPE VAN SDN Controller Debian package

You can get these downloads from either the HPE **My Networking Portal** or from the Hewlett Packard Enterprise SDN App Store.

To download the HPE VAN SDN Controller software packages from the My Networking Portal:

1. Go to the Hewlett Packard Enterprise Networking support site at www.hpe.com/networking/support.
2. Enter the HPE VAN SDN base product number **J9863AAE** in the **Enter product name/number** field.
3. Select the check box next to the HPE VAN SDN Controller product, then click **Show selected items**.
4. In the lower right quadrant of the **My Support** screen, click **Software downloads**.
5. On the **Download software** screen, select and download the software package of your choice to your local machine.

6. Unzip the software package.
7. Read the latest **HPE VAN SDN** *Controller Release Notes* included with the software package.

To download the HPE VAN SDN Controller software packages from the Hewlett Packard Enterprise SDN App Store:

1. Log in to the Hewlett Packard Enterprise SDN App Store at www.hpe.com/networking/sdnappstore.
2. Select **Controller** from the list of available applications and download the software package of your choice (either .deb or .ova file).
3. Use the links provided to download the latest *HPE VAN SDN Controller Release Notes* and other documentation

7.6.3 Running HPE VAN

- Import virtual machine

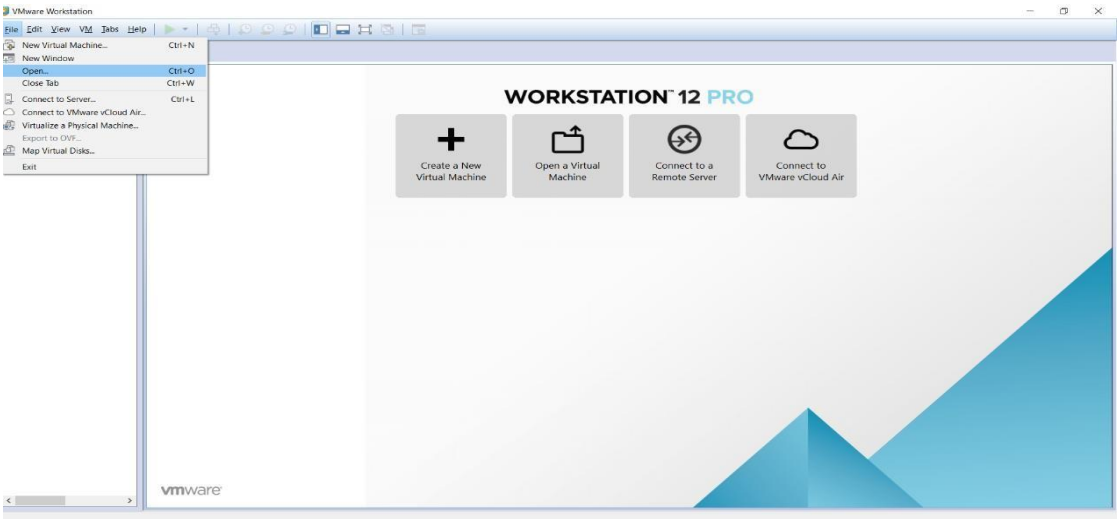


Figure 7.1: Import virtual machine

- Modify virtual controller machine settings to be compatible with our device
- Modify memory settings to 4 G RAM
- Modify processor settings preferred to use number of processors =1 & no. of cores per processor = 1

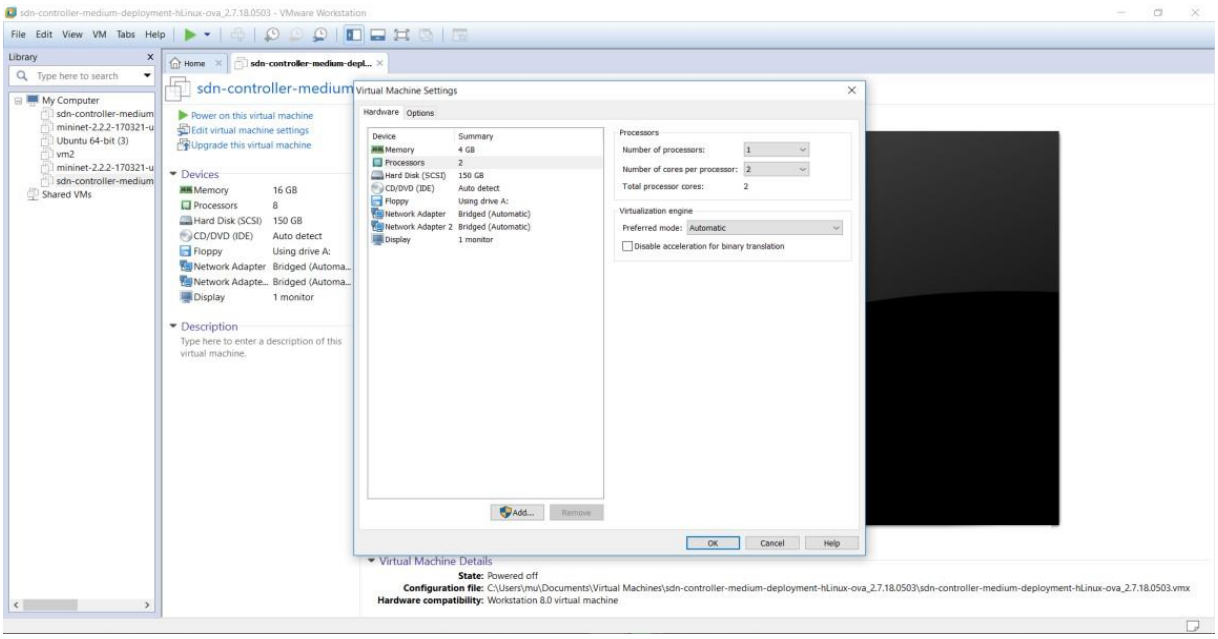


Figure 7.2: Edit a setting of Virtual Machine

- Modify network adapter settings to bridged networking to allow connect to internet with our private IP.
- We need only one network adapter card so we removed the second network card.

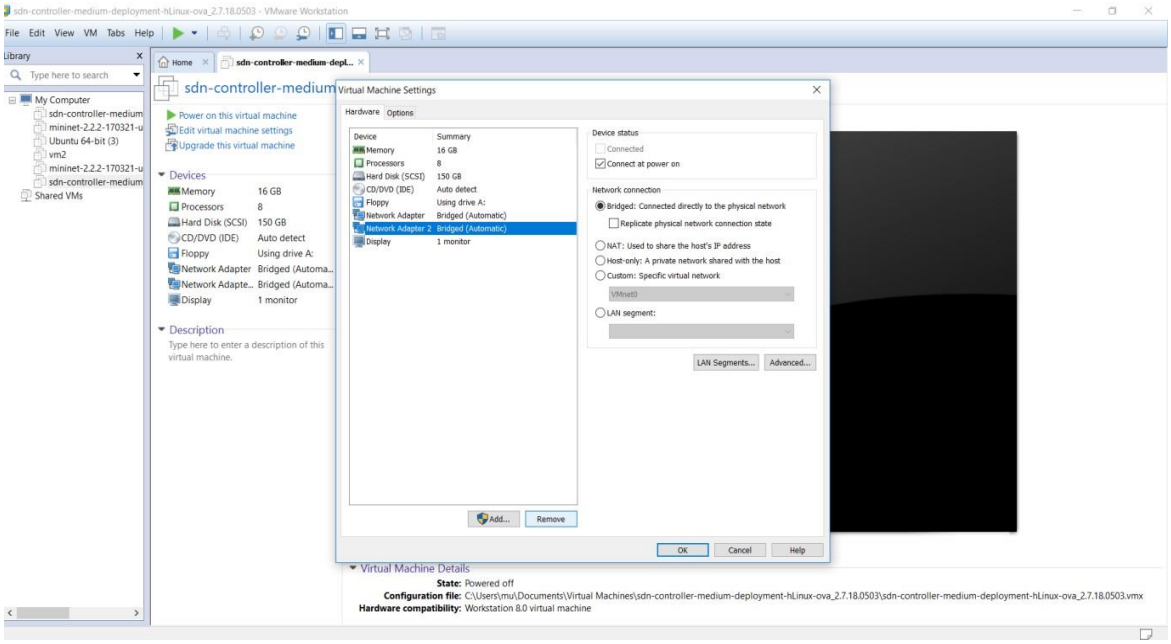


Figure 7.3: Network Setting of Virtual Machine

- **Installing Steps:**

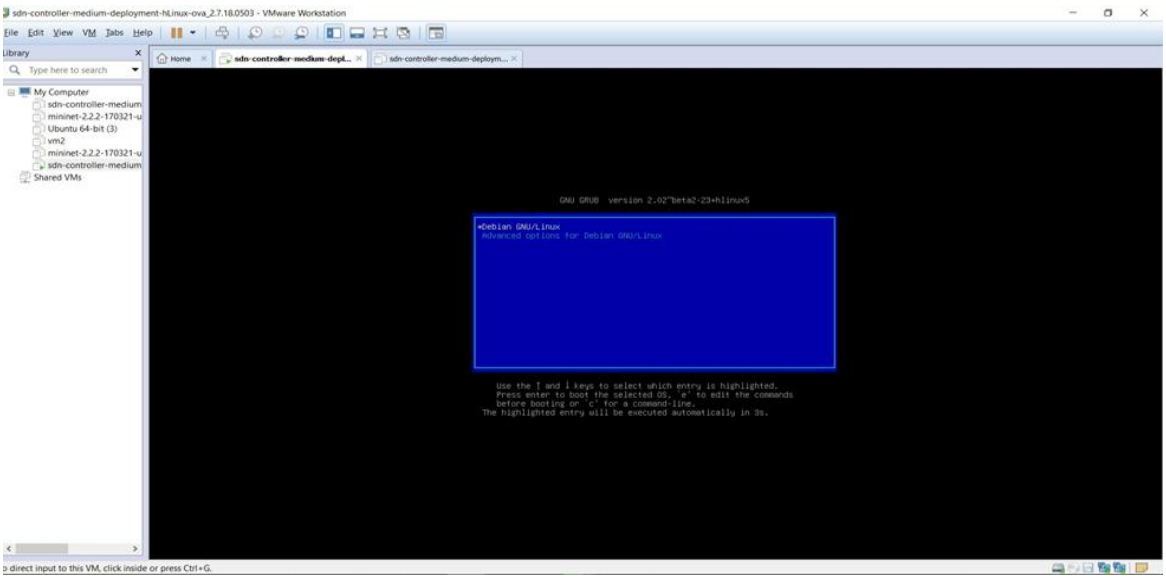


Figure 7.4: GUI / LINUX

- After powering on the controller virtual machine, we will log in using the default username and password:

  Login virtual machine with

Username	Sdn
Password	Skyline

Table 7.2: Password and Username of Virtual Machine

- It'll take a while till it assigns a static IP.

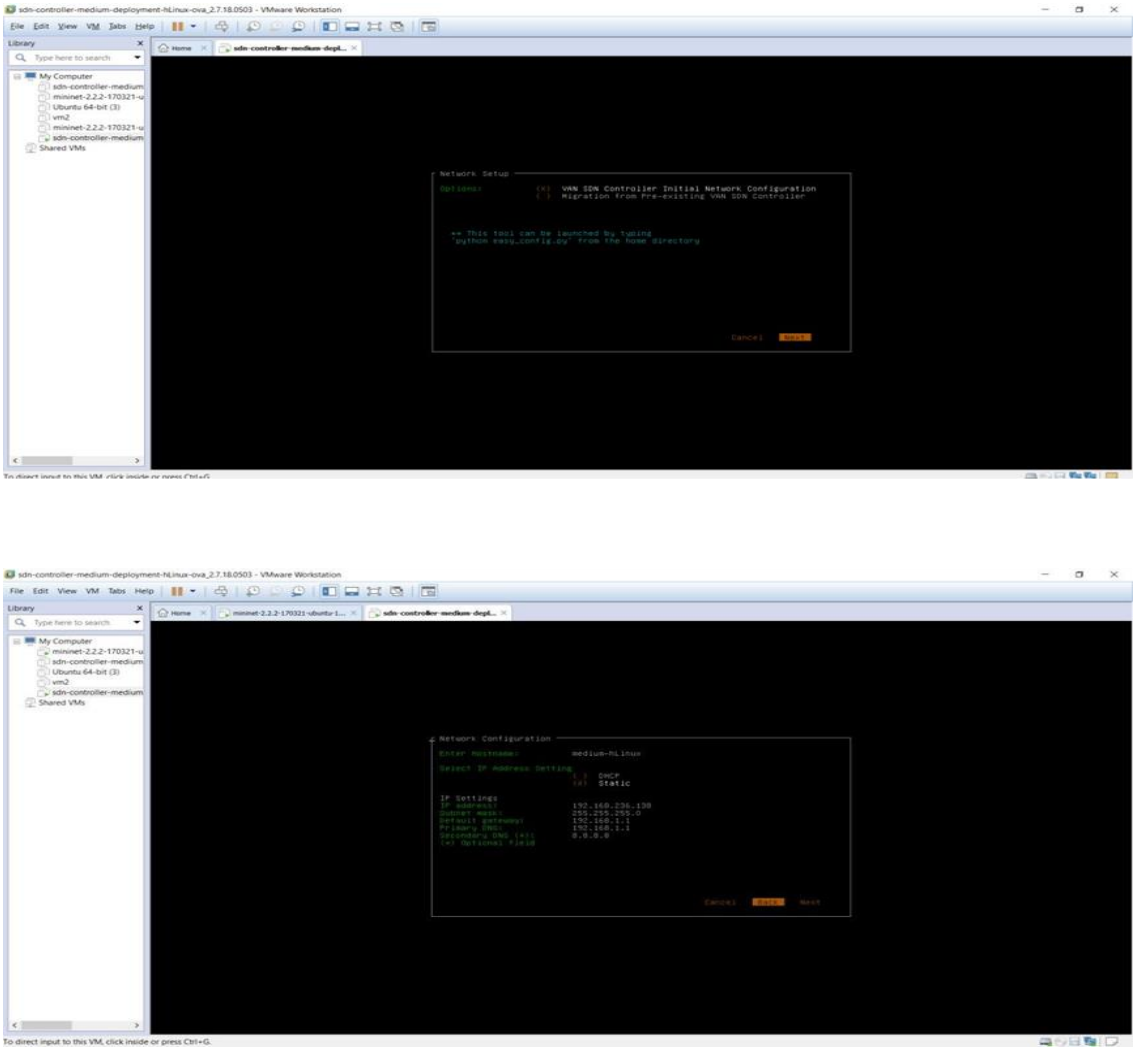


Figure 7.5: assigns a static IP

For remote connection there are many programs such as:

- Traterm
- Putty
- Secure CRT
- Super putty

We choose super putty from these programs as it's easy to deal with and enable us to open more than one session at the same time.

- **Create default topology on Mininet by command:**

```
$sudo mn
```

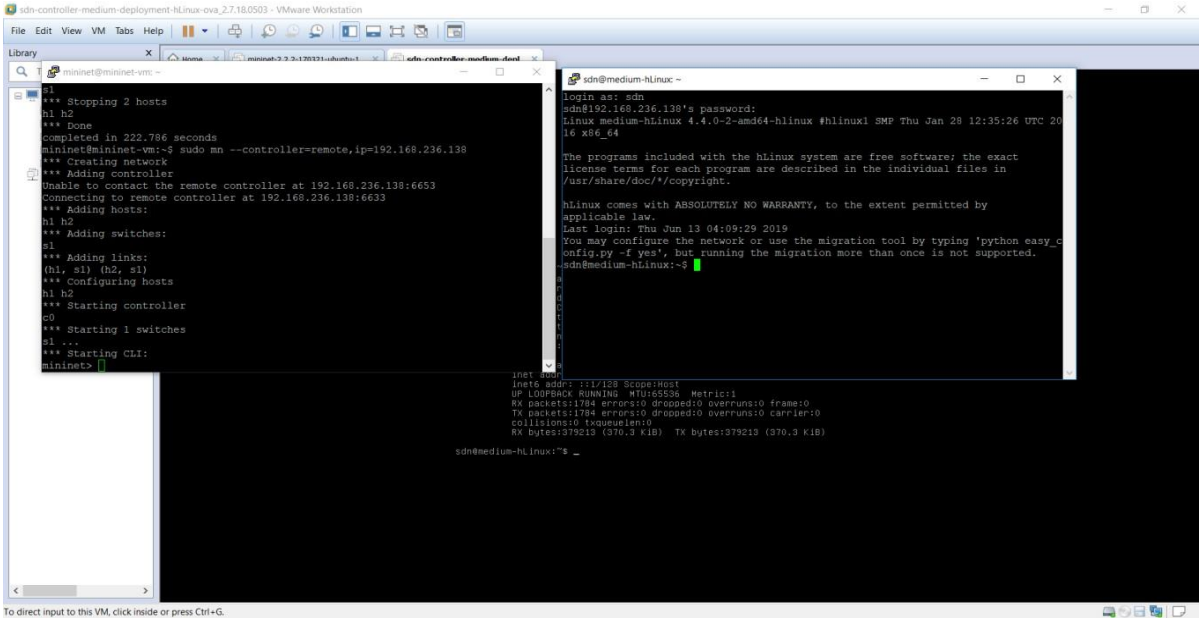


Figure 7.6: Sudo mn Command Output

1. Define controller machine to Mininet Machine by command:

```
$ sudo mn -- controller=remote,ip=192.168.236.138
```

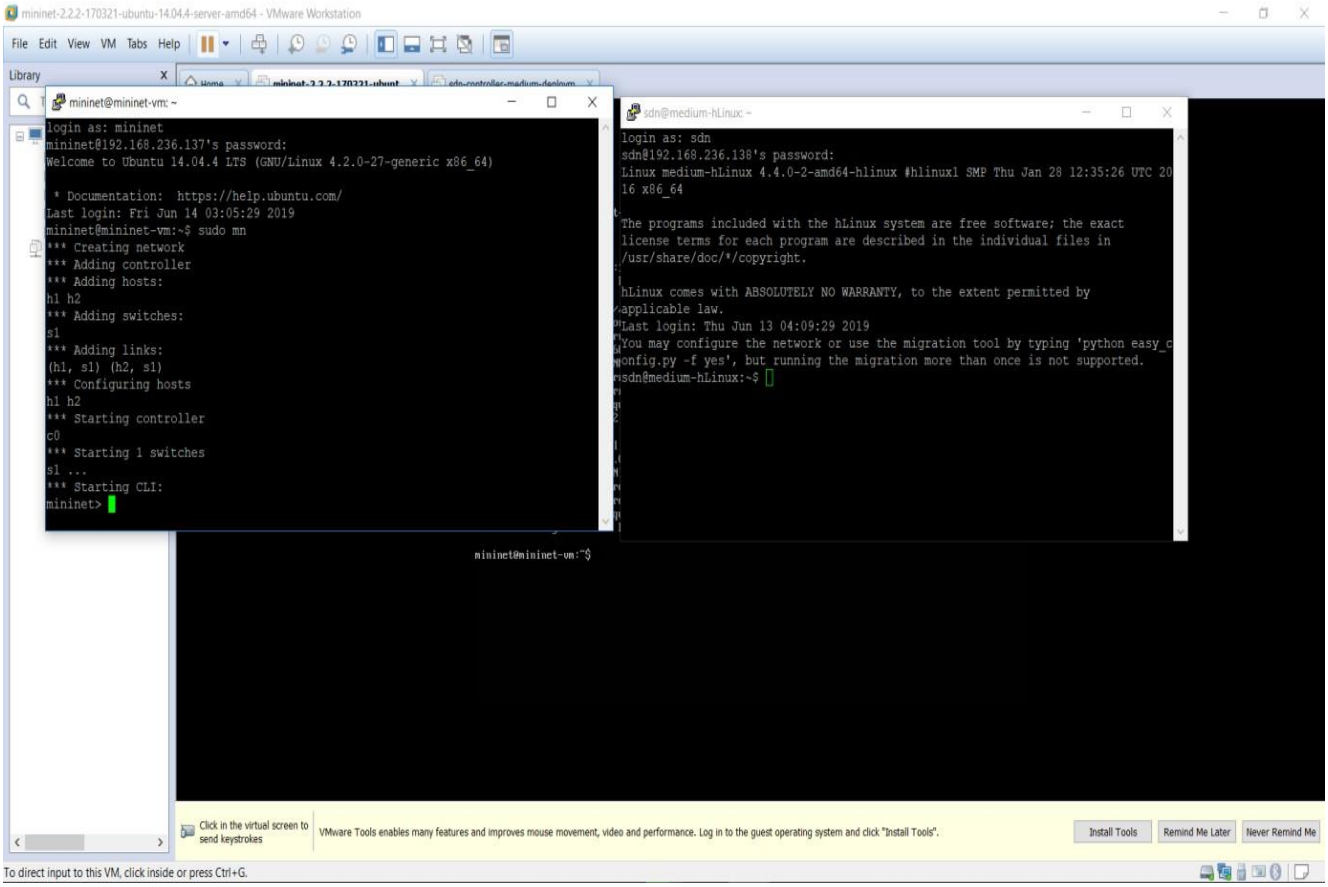


Figure 7.7: Sudo mn With Options Command Output

$ Ping all

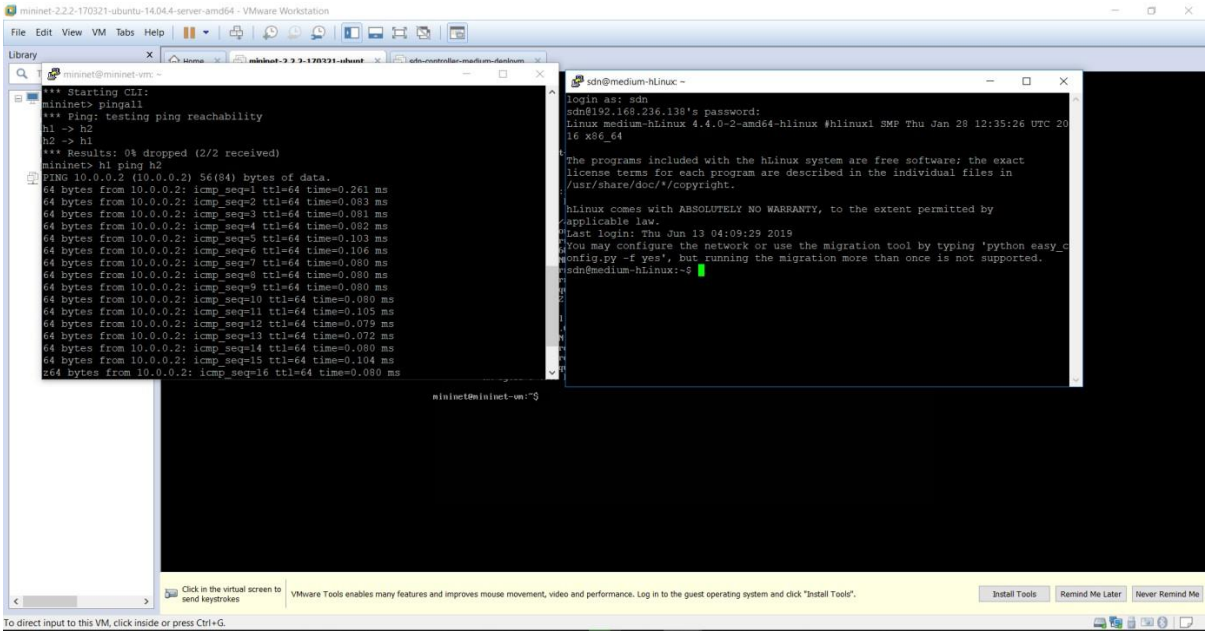


Figure 7.8: Ping all Command Output

2. Connect to: https://192.168.236.138:8443 to deal with the controller GUI.

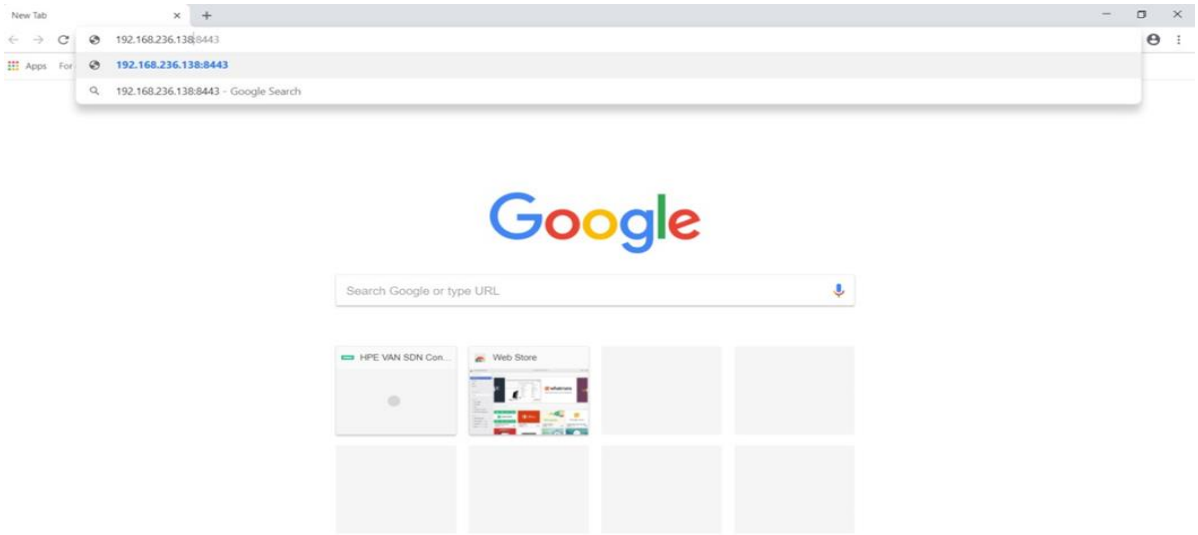


Figure 7.9: Connect to https://192.168.236.138:8443

3. Login to controller page with (username: sdn & password: skyline)

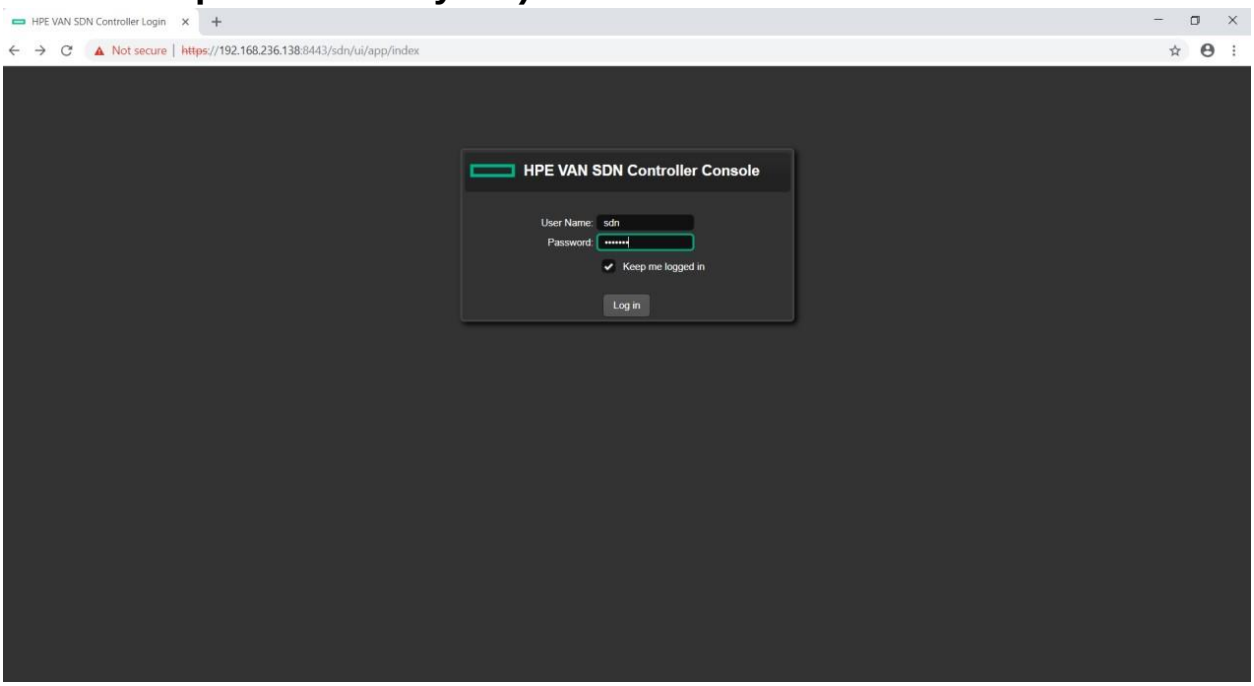


Figure 7.10: Enter to HPE VAN Console

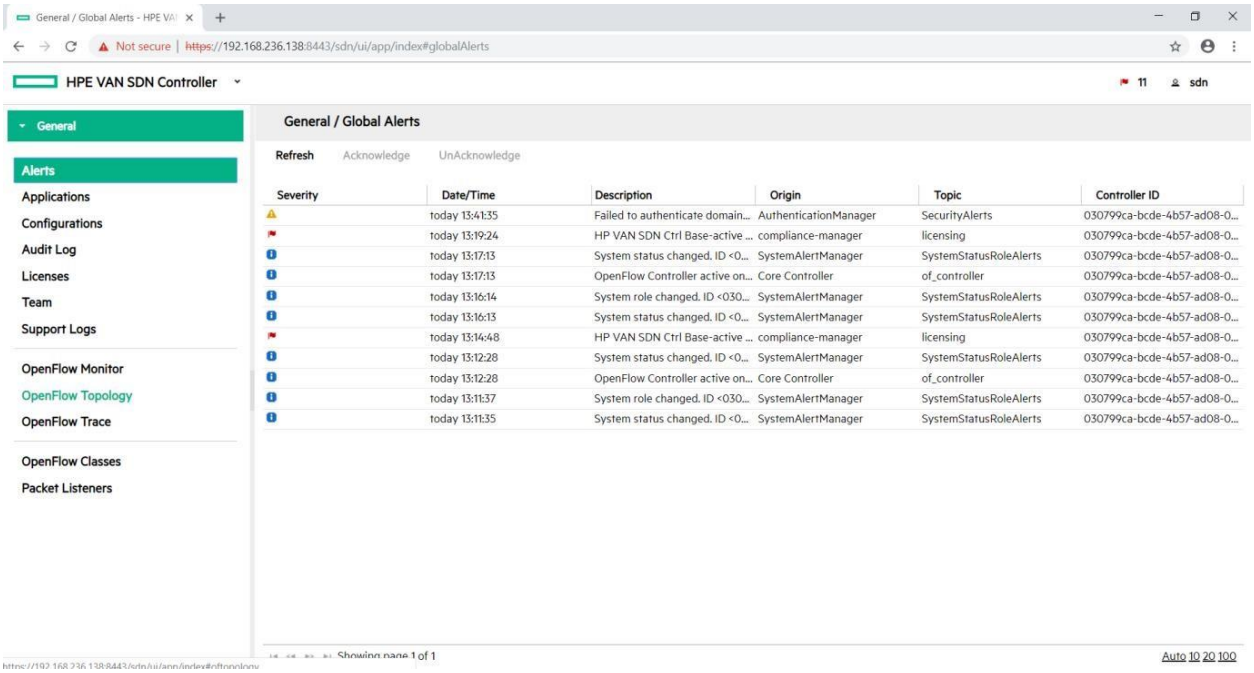


Figure 7.11: GUI HPE VAN Console

4. After reload to show result of command (sudo mn)

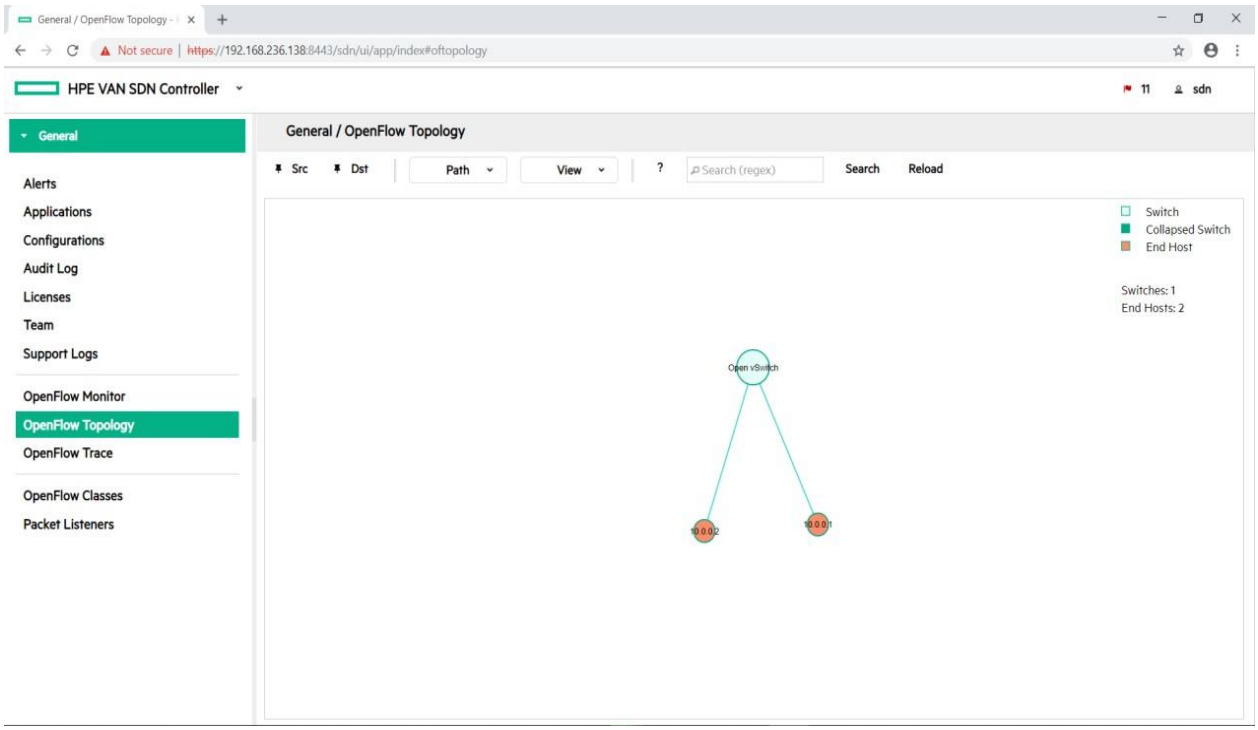


Figure 7.12: Result of sudo mn Command in HPE VAN

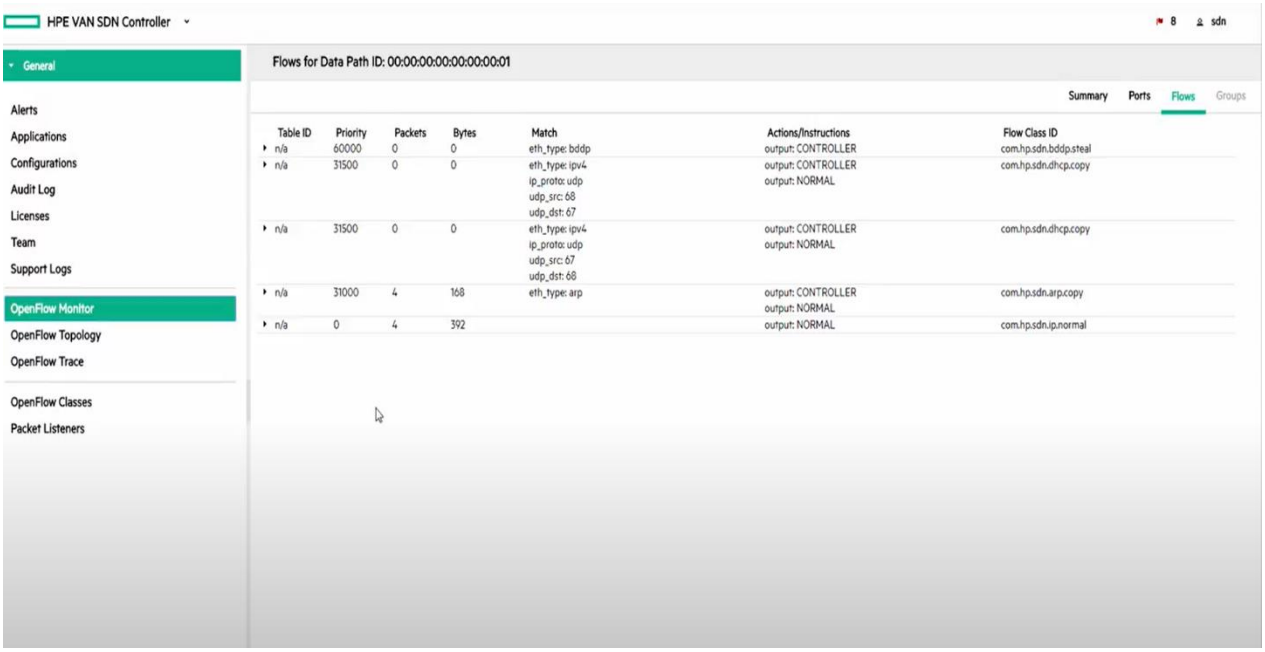


Figure 7.13: Flow Table HPE VAN

7.7 RYU

7.7.1 Introduction

1. What is RYU?

Ryu Controller is an open, software-defined networking (SDN) Controller designed to increase the flexibility of the network by making it easy to manage and adapt how traffic is handled. Ryu supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-Config, as well as OpenFlow, which Ryu using it to interact with the forwarding plane (switches and routers) to modify how the network will handle traffic flows. All of the code is open source on GitHub. Ryu is fully written in Python.

Ryu is a component-based software defined networking framework. Ryu provides software components with well-defined API that make it easy for developers to create new network management and control applications;

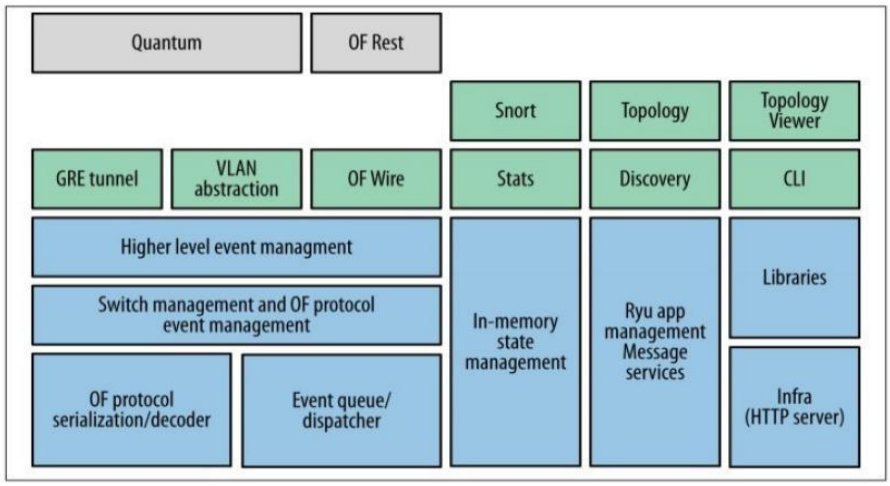


Figure 7.14: RYU ARCHITECTURE, APPLICATIONS (NON-EXHAUSTIVE), AND APIS

2. Components

• Provides interface for control and state and generates events

• *Communicates* using message passing

3. Libraries

• Functions called by components Eg: OF-Config, Netflow, sFlow, Netconf, OVSDB.

7.7.2 Getting Started:

• install Ryu from the source code:

```
$ git clone git://github.com/osrg/ryu.git
$ cd ryu
$ pip install
```

7.7.3 Ryu viewer

This is a topology viewer which enable you to see your topology which was created on Mininet and see the packets flow, This depends on following Ryu applications as shown in the following table.

ryu.app.rest topology	**Get node and link data**
ryu.app.ws topology	**Being notified change of link up down**
ryu.app.ofctl rest	**Get flows of data paths**

Table 7.3: Ryu Applications

- **Run Mininet (or join your real environment):**

```
$ sudo mn --controller remote --topo tree,depth=3
```

- **Run GUI application:**

```
$ PYTHONPATH=. ./bin/ryu run --observe-links
ryu/app/gui_topology/gui_topology.p
```

- **Access http://"ip address of ryu host":8080 with your Web browser**

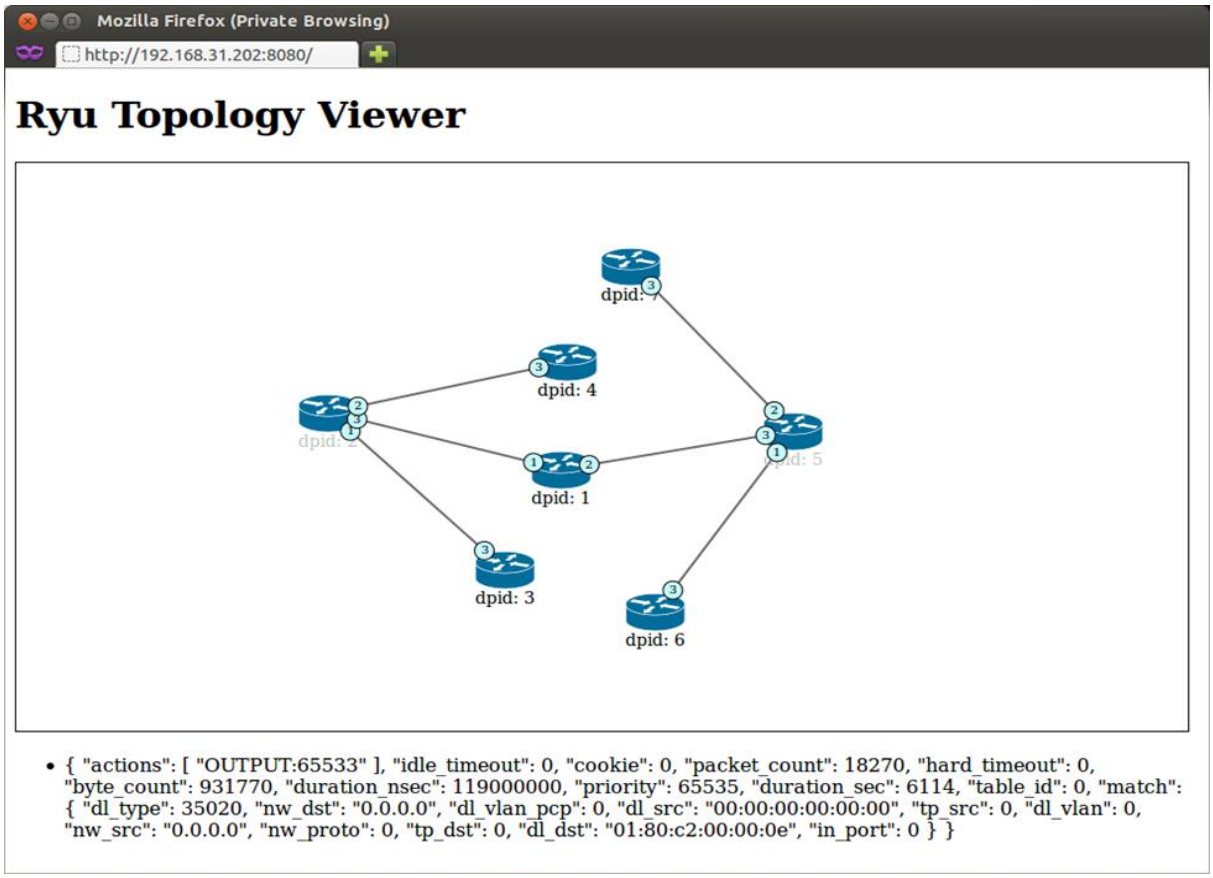


Figure 7.15: Ryu Topology Viewer

7.7.4 Ryu Applications

7.7.4.1 Layer 2 switch

forwards packets based on mac address.

In this example, we will connect 3 client hosts to software switch bridge ports. You can use 3 physical hosts as client hosts. If you have a Linux host (VM) with 3 or more interfaces, you can either create 3 VMs or create network namespaces to simulate physical hosts.

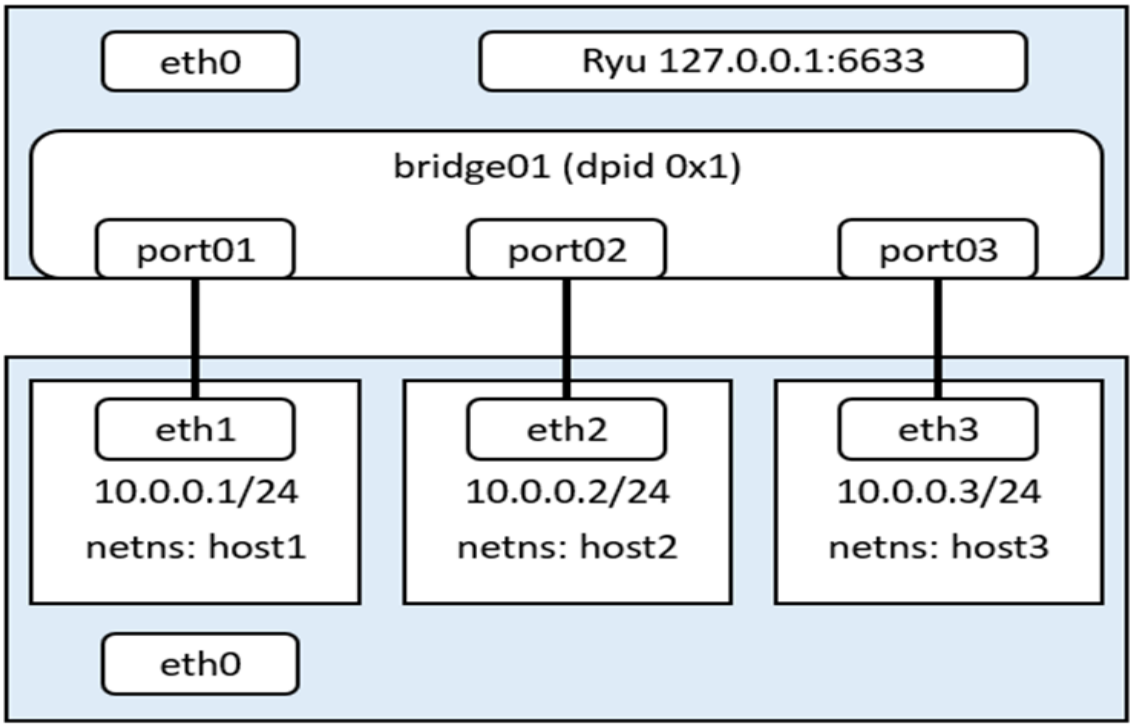


Figure 7.16: Hierarchy of forward packets based on mac address

- **Practical Lap Steps:**

1. Make a Topology in Mininet:

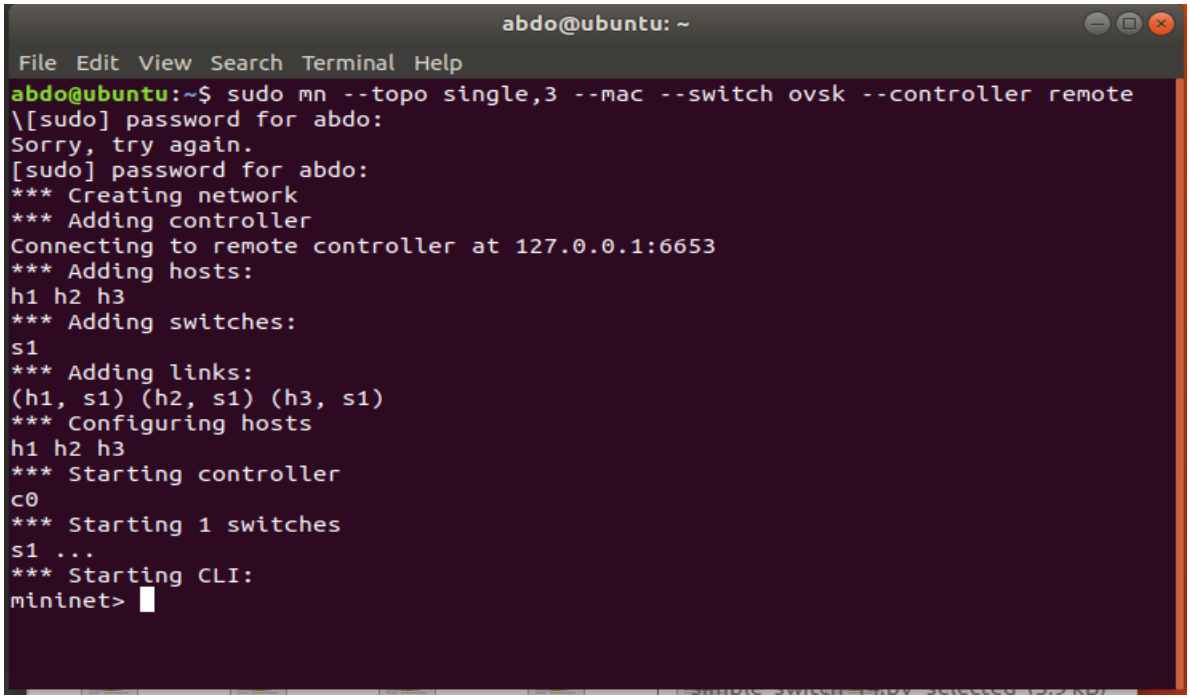


Figure 7.17: Creation of Topology

2. After pingall Command:

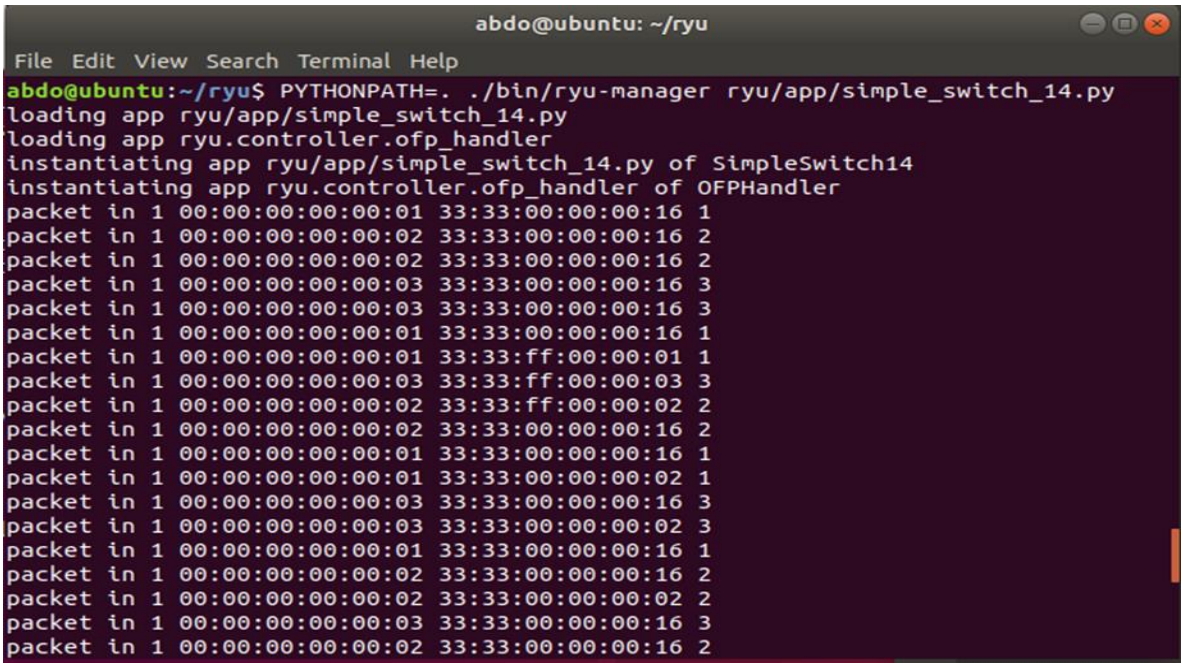


Figure 7.18: Pingall Command Output in Ryu

7.7.4.2 Spanning Tree Protocol

Spanning tree is a function that suppresses occurrence of broadcast streams in a network having a loop structure. applying the original function that is preventing the loop. it is used as a means to secure network redundancy to automatically switch the path in case of a network failure. Spanning Tree Protocol (STP: IEEE 802.1D) handles a network as a logical tree and by setting the ports of each switch (sometimes called a bridge in this section) to transfer frame or not it suppresses occurrence of broadcast storms in a network having a loop structure.

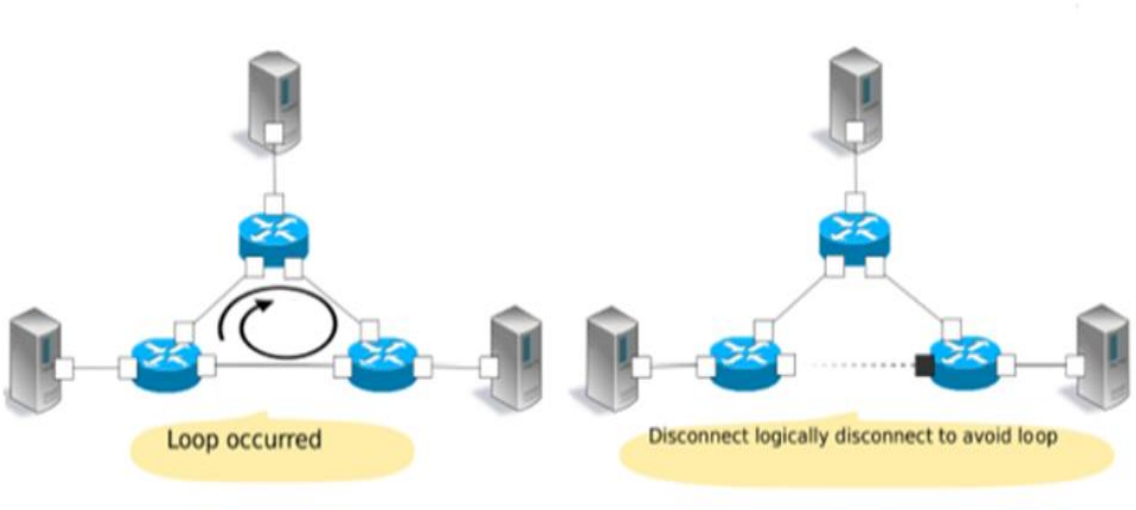


Figure 7.19: Spanning Tree Protocol

- **Spanning Tree Protocol Mechanism:**

  1. **Selecting the root bridge:**

     sends the original BPDU packet and other bridges transfer BPDU packets received from the root bridge. Deciding the role of port Based on the cost

of each port to reach the root bridge, decide the role of the ports.

2. **Root port:**

   The port having the smallest cost among bridges to reach the root bridge.

   This port receives BPDU packets from the root bridge.

3. **Designated ports:**

   Ports at the side having the small cost to reach the root bridge of each link. These ports, sends BPDU packets received from the root bridge. Root bridge ports are all designated ports.

4. **Non designated ports:**

   Ports other than the root port and designated port. These ports suppress frame transfer.

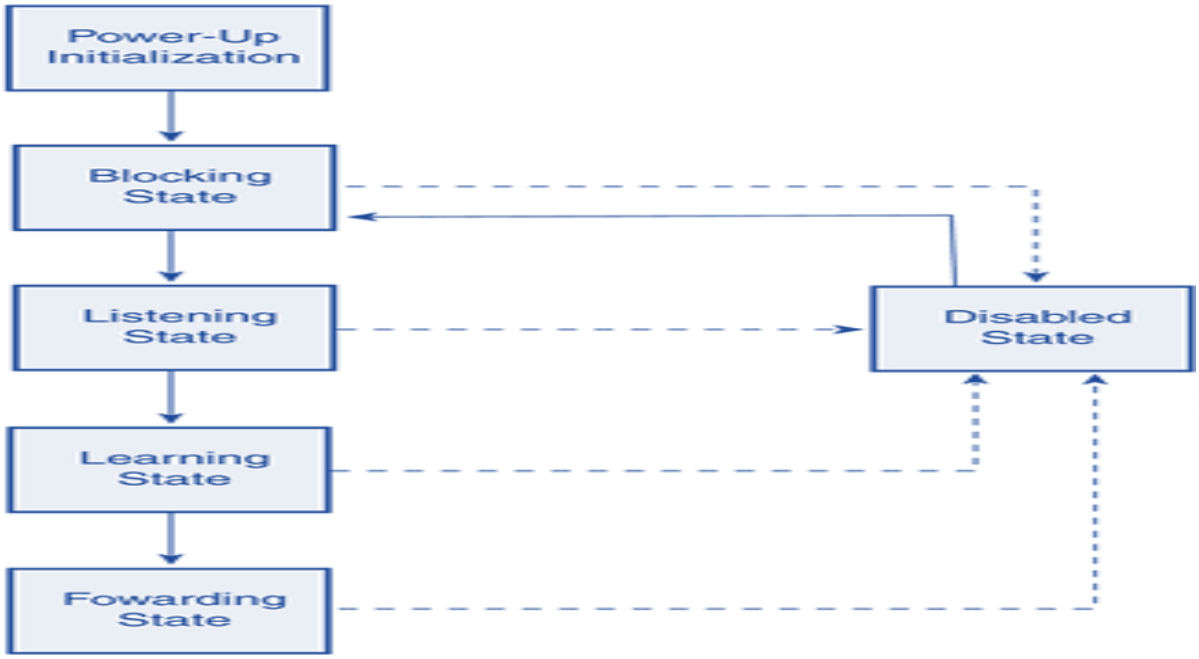


Figure 7.20: Spanning Tree Protocol Mechanism

- **Executing STP Ryu Application**

  First open VM and open 2 terminal one for creating topology and the other for mention the Spanning tree code to Controller. Here we create 3 switches and 3 hosts.

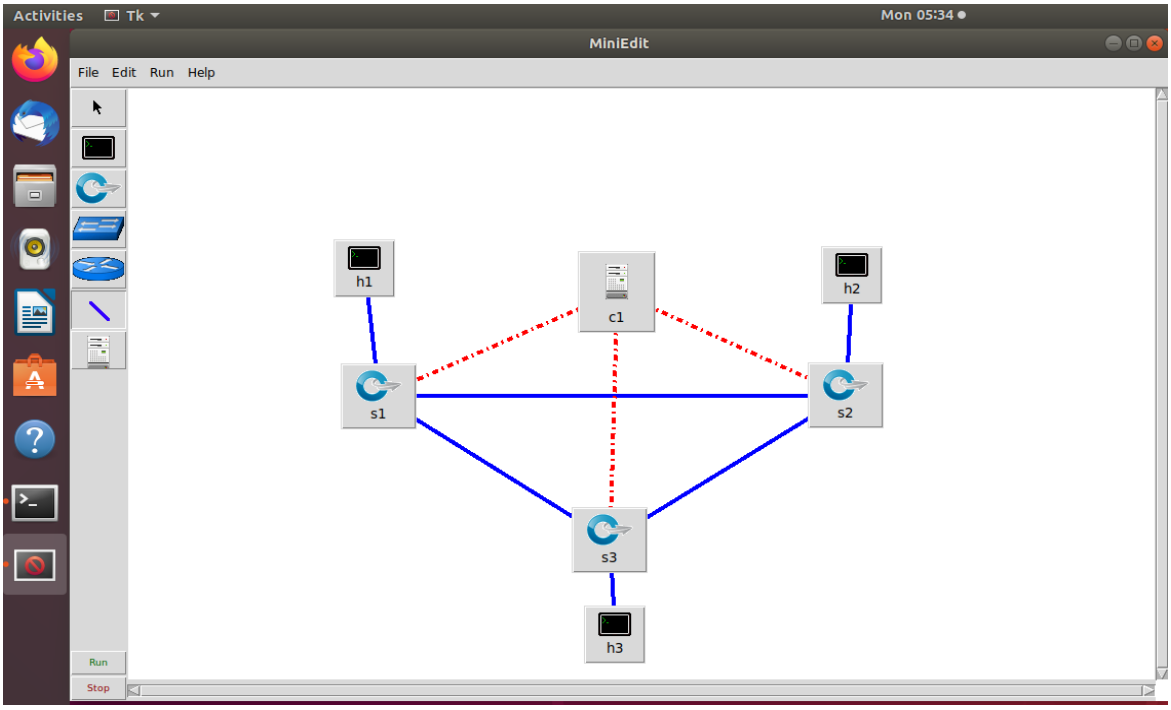


Figure 7.21: Spanning Tree Protocol Topology

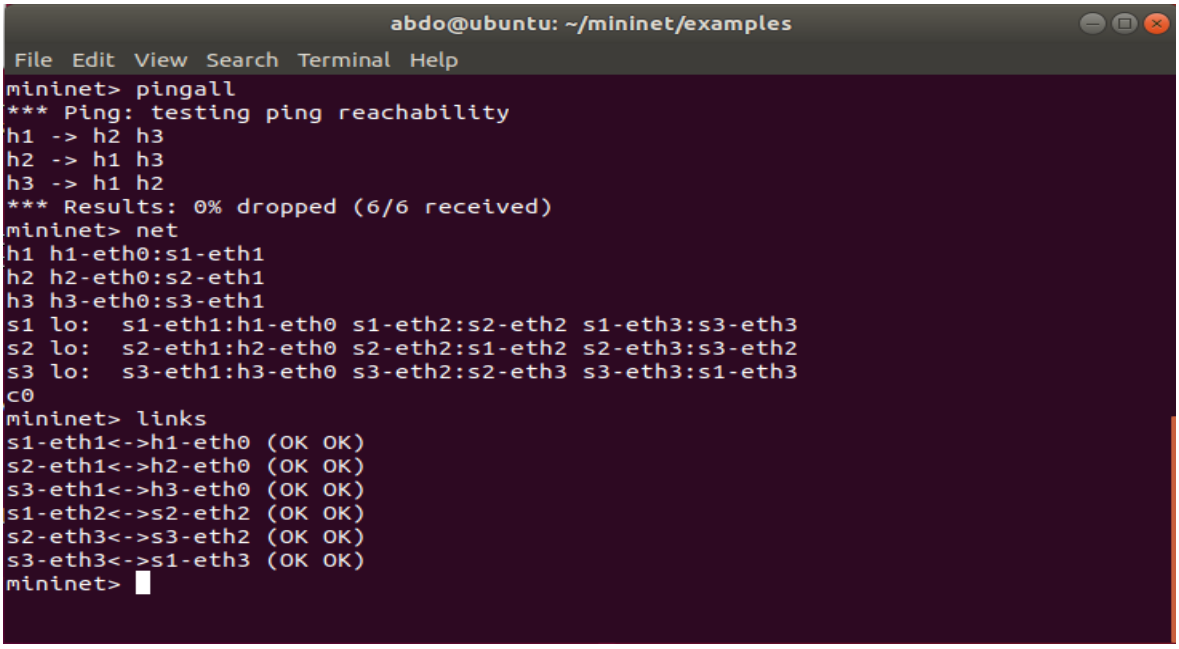


Figure 7.22: Results of Net, Link and Pingall Commands

- **Calculating STP Upon OpenFlow Swtich Starts:** When connection between each OpenFlow switch and the controller is completed, exchange of BPDU packets starts and root bridge selection, port role setting, and port state change takes place. As a result, each port eventually becomes FORWARD state or BLOCK state.

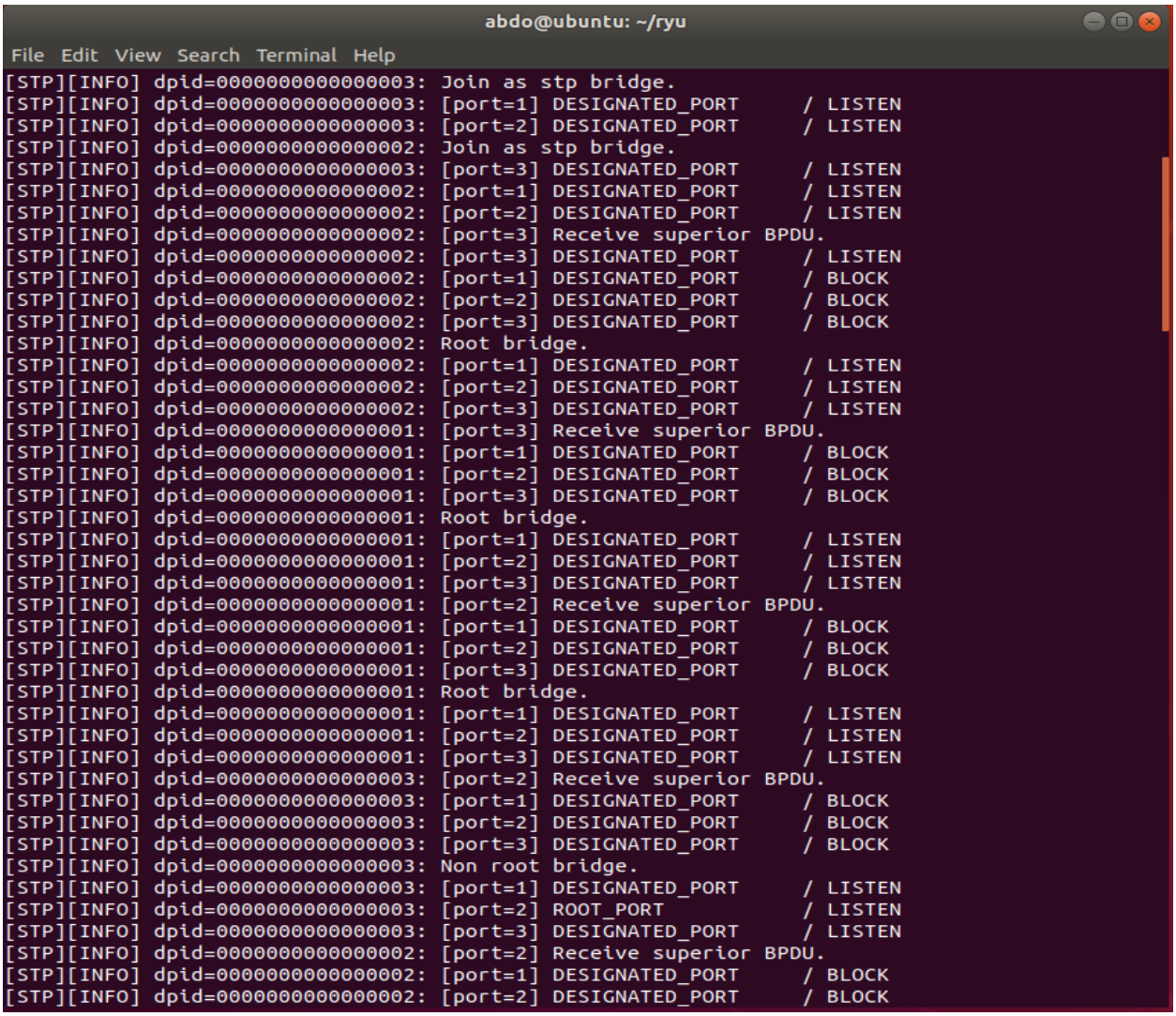


Figure 7.23: Exchange BPDU Messages Part 1

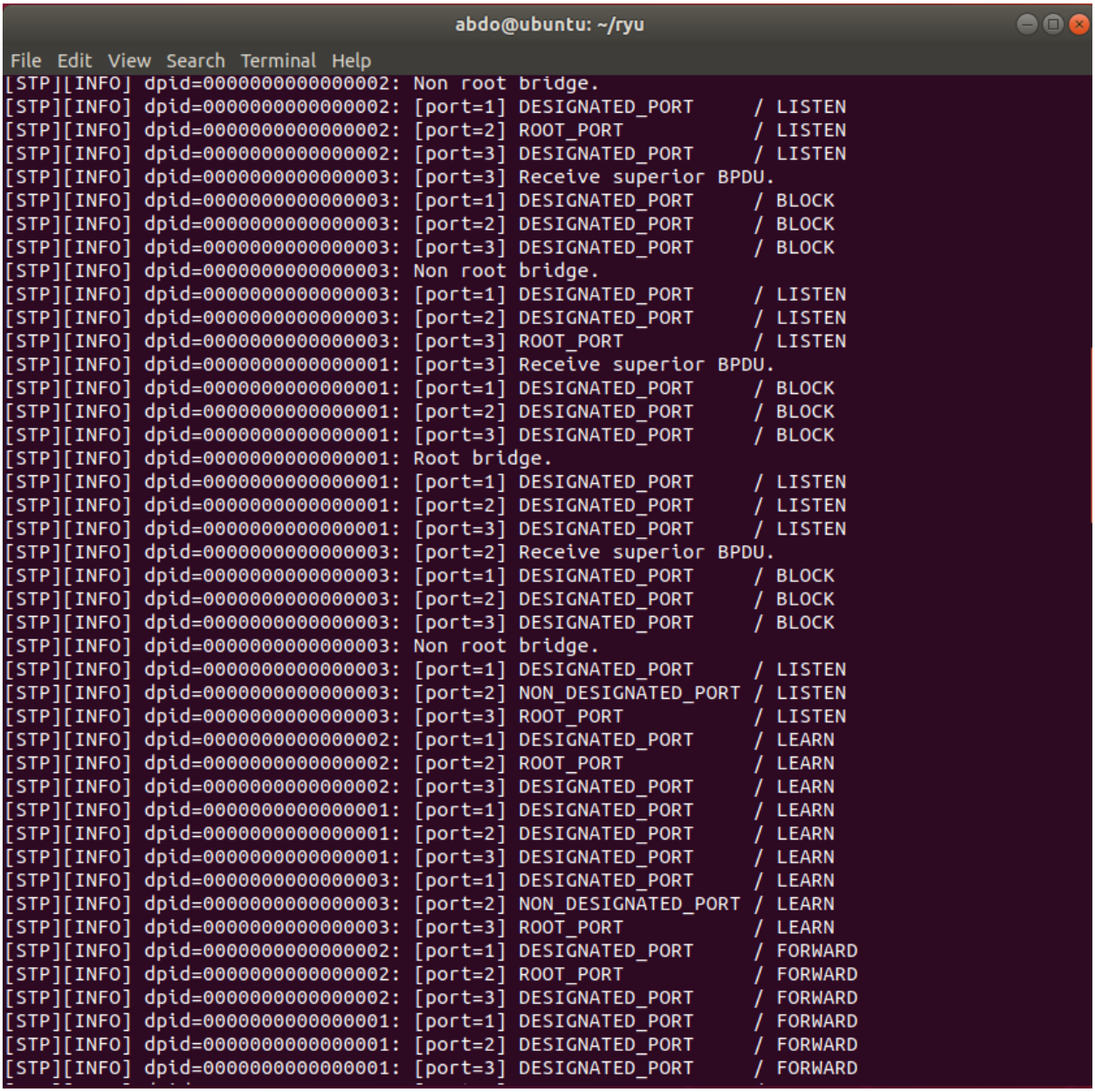


Figure 7.24: Exchange BPDU Messages Part 2

- **Results:**

  in order to confirm that the packets are not looped execute ping command and **tcpdump** command.

  as a result of **tcpdump** output, you can confirm that ARP is not looped.

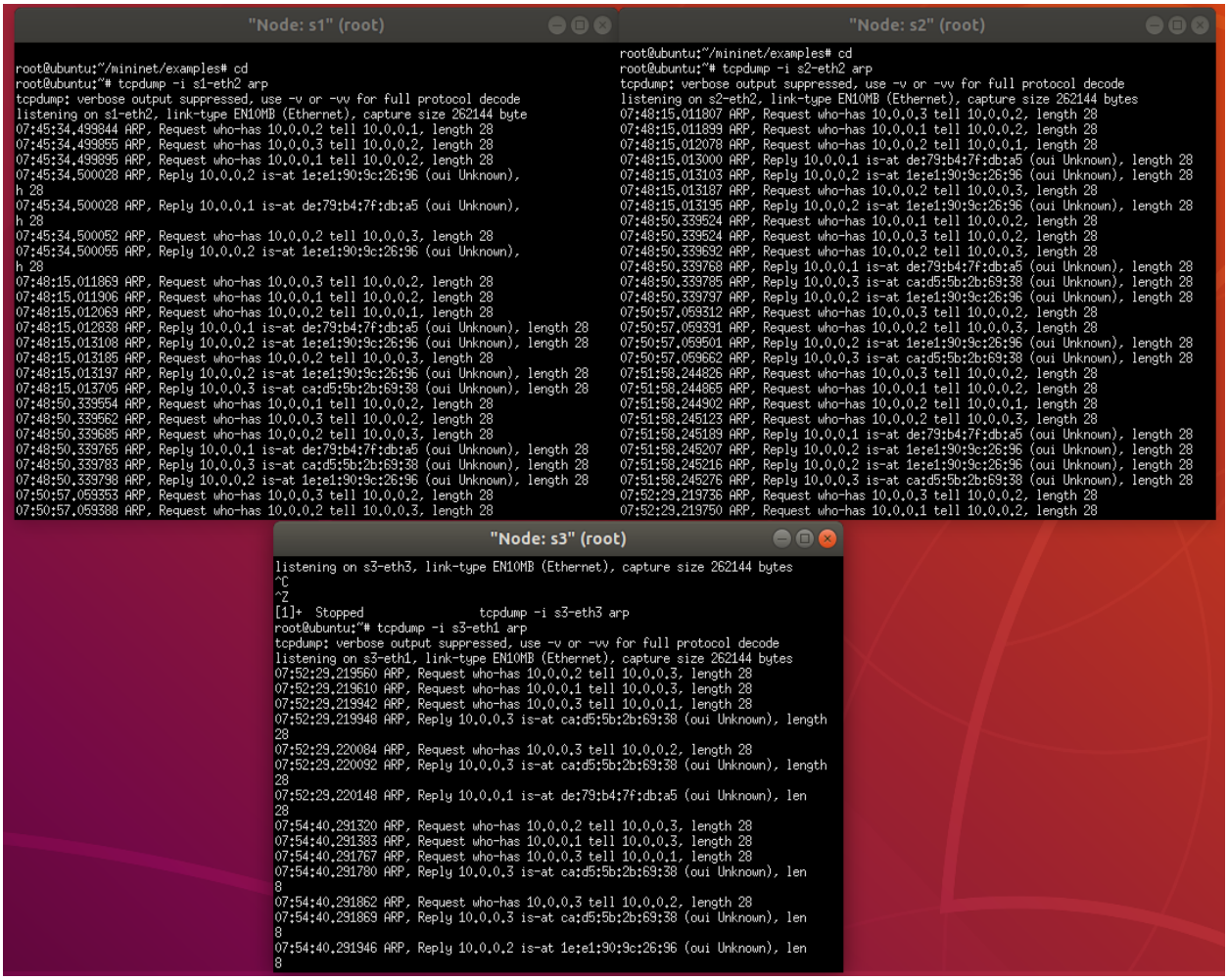


Figure 7.25: Tcpdump Command Output

- **Re-Calculation When a Failure is Detected:**
  Next, let's check re-calculation operation of STP in case of link down. In the state in which STP calculation has been completed after each OpenFlow switch starts, execute the following commands to make the port down. Link down is detected and recalculation of STP is executed.

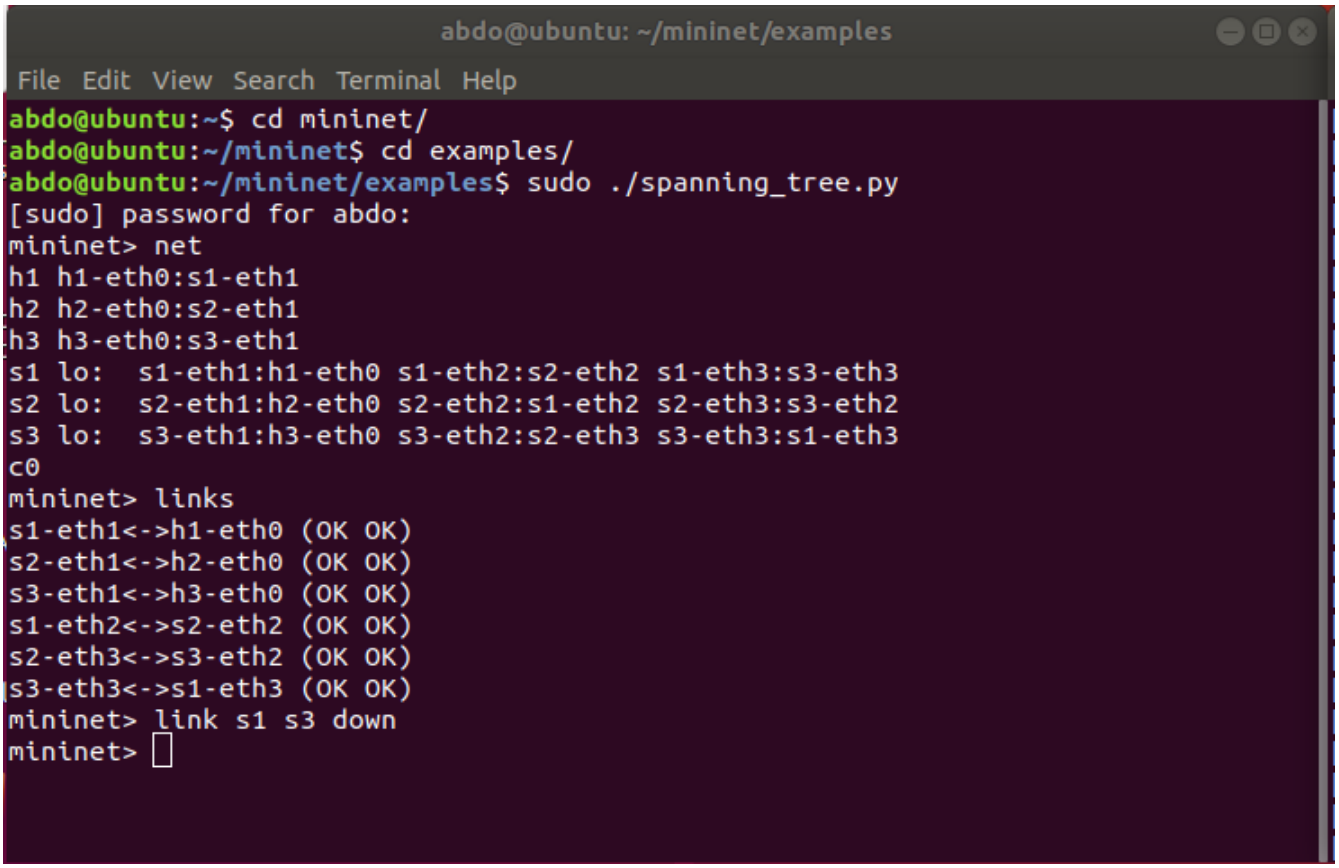


Figure 7.26: Link s1- s3 down Command

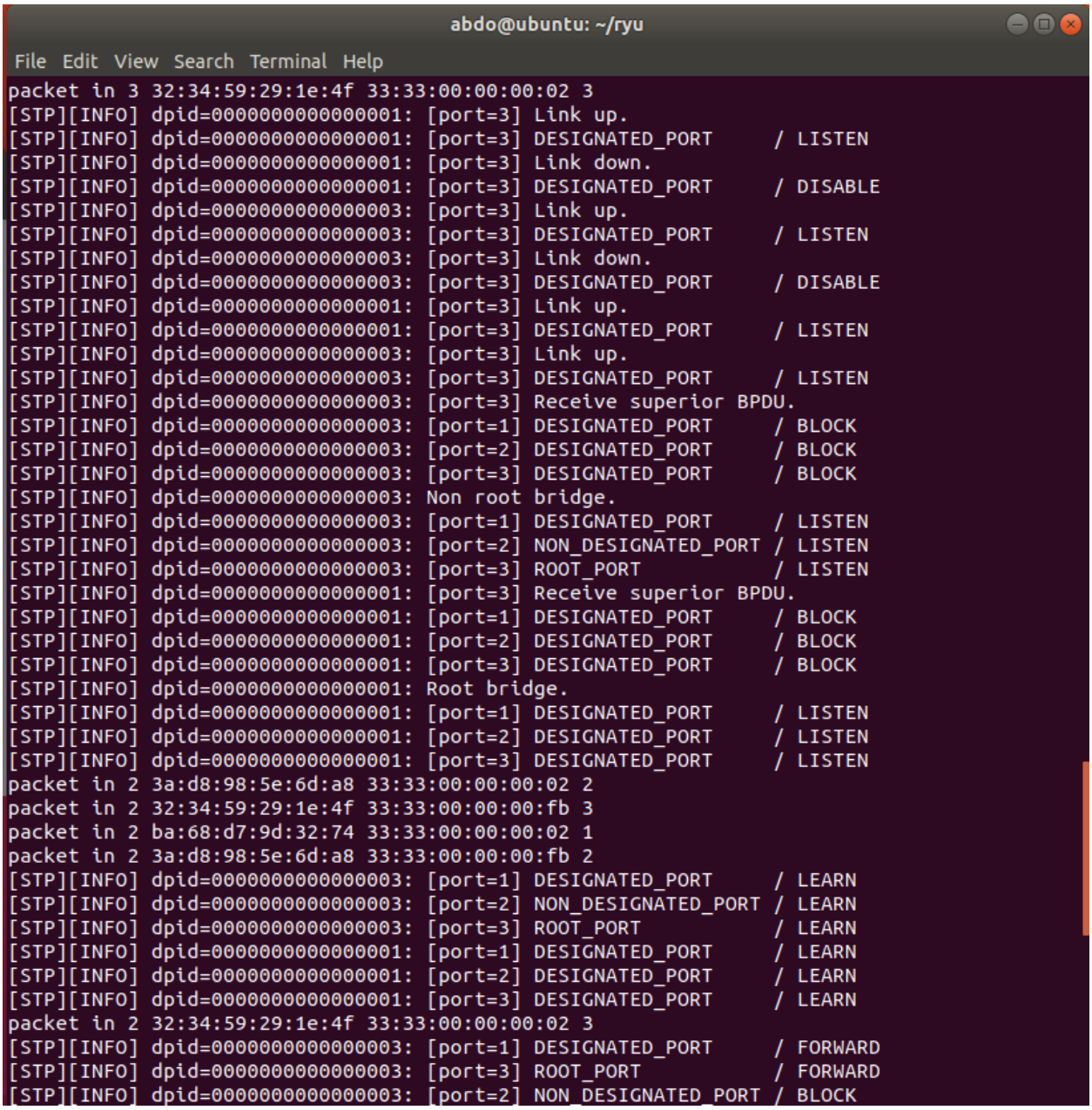


Figure 7.27: Detect Failure, Recalculate

Bridge class instance. Timeout of BPDU packet receive waiting is detected by the BPDU packet receive waiting thread (Port wait BPDU thread) of the Port class. When BPDU packets from the root bridge cannot be received for the maximum age (default: 20 seconds), an indirect failure is determined and is notified to the Bridge class instance.

7.8 POX

7.8.1 Introduction

1. POX Definition

POX is an open source development platform for Python-based software-defined networking (SDN) control applications, such as OpenFlow SDN controllers.

POX provides a framework for communicating with SDN switches using either the OpenFlow or OVSDB protocol. Developers can use POX to create an SDN controller using the Python programming language. It is a popular tool for teaching about and researching software defined networks and network applications programming.

2. POX Features

- Pythonic OpenFlow interface.
- Reusable sample components for path selection, topology discovery, etc.
- Runs anywhere can bundle with install-free PyPy runtime for easy deployment.
- Specifically targets Linux, Mac OS, and Windows.
- Topology discovery.
- Supports the same GUI and visualization tools as NOX.
- Performs well compared to NOX applications written in Python.

3. POX Components

POX components are additional Python programs that can be invoked when POX is started from the command line. These components implement the network functionality in the software defined network.

POX functionality is provided by its *components,* it can be immediately used as a basic SDN controller by using the stock components that come bundled with it. the **following** are examples of these components:

- **forwarding.hub**

The hub example just installs wildcarded flood rules on every switch, essentially turning them all into $10 ethernet hubs.

- **forwarding.l2_learning**

This component makes OpenFlow switches act as a type of L2 learning switch. This one operates much like NOX's "pyswitch" example, although the implementation is quite different. While this component learns L2 addresses, the flows it installs are exact-matches on as many fields as possible.

- **forwarding.l3_learning**

This component is not quite a router, but it's also definitely not an L2 switch. It's an L3-learning-switchy-thing. Perhaps the most useful aspect of it is that it serves as a pretty good example of using POX's packet library to examine and construct ARP requests and replies.

$./pox.py forwarding.l3_learning –fakeways=10.0.0.1,192.168.0.1

- **forwarding.l2_pairs**

Like l2_learning, this component also makes OpenFlow switches act like a type of L2 learning switch. However, this one is probably just about the simplest possible way to do it correctly. Unlike l2_learning, l2_pairs install rules based purely on MAC addresses.

- **forwarding.l2_multi**

This component can still be seen as a learning switch, but It has a twist compared to the others. The other learning switches “learn” on a switch-by-switch basis, making decisions as if each switch only had local information. l2_multi uses openflow.discovery to learn the topology of the entire network as soon as one switch learns where a MAC address is, they all do. Note that this means you must include OpenFlow.Discovery on the command line.

- **Openflow.spanning_tree**

This component uses the discovery component to build a view of the network topology, constructs a spanning tree, and then disables flooding on switch ports that aren’t on the tree. The result is that topologies with loops no longer turn your network into useless hot packet soup.

Note that this does not have much of a relationship to Spanning Tree Protocol. They have similar purposes, but this is a rather different way of going about it.

The **samples.spanning_tree** component demonstrates this module by loading it and one of several forwarding components. This component has two options which alter the startup behavior:

- **--no-flood** disables flooding on all ports as soon as a switch connects; on some ports, it will be enabled later.
- **--hold-down** prevents altering of flood control until a complete discovery cycle has completed (and thus, all links have had an opportunity to be discovered).
  Thus, the safest (and probably the most sensible) invocation is **openflow.spanning_tree --no-flood --hold-down**.

7.8.2 Install/Run POX

1. Installing POX

The best way to work with POX is as a git repository. You can also grab it as a tar ball or zip ball, but source control is generally a good thing. POX is hosted on **GitHub**; you can simply create a local clone:

```
$ git clone http://github.com/noxrepo/pox
$ cd pox
```

2. Running POX

- Run Mininet on a terminal window using the following command. This starts a network emulation environment to emulate 1 switch with 3 hosts. The switch tries to connect to

port 6633 on localhost. Initially, there is no controller listening.

```
$ sudo mn --topo single,3 --mac --controller remote --switch ovsk
```

- Since POX does not support OpenFlow 1.3 yet, it is best to statically set the OpenFlow version of the OVS

```
$ sudo ovs-vsctl set bridge s1 protocols=OpenFlow10
```

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

```
$ sudo~/pox/pox.pyforwarding.l2_learning
```

7.8.3 POX controller applications

7.8.3.1 POX as a hub

1. First the controller should act as a hub

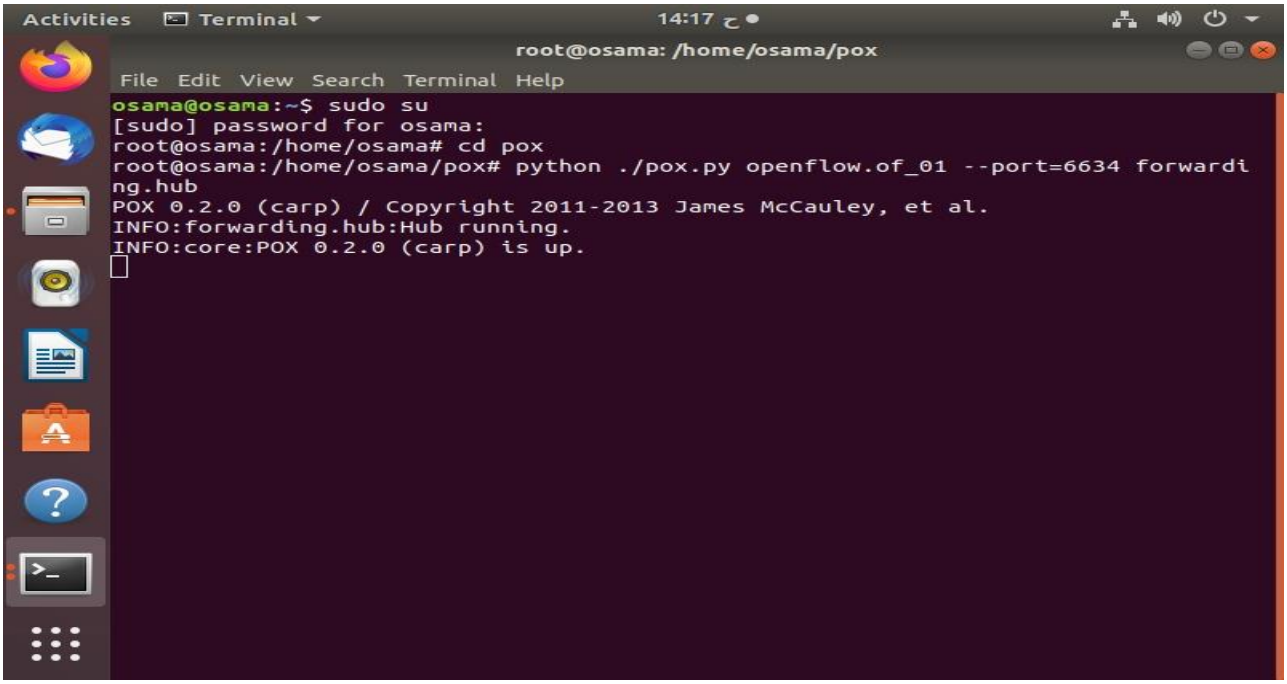


Figure 7.28: Controller as a Hub

2. Create the topology

```
osama@osama:~$ sudo su
[sudo] password for osama:
root@osama:/home/osama# sudo mn --topo single,6 --controller=remote,port=6633
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6
h2 -> h1 h3 h4 h5 h6
h3 -> h1 h2 h4 h5 h6
h4 -> h1 h2 h3 h5 h6
h5 -> h1 h2 h3 h4 h6
h6 -> h1 h2 h3 h4 h5
*** Results: 0% dropped (30/30 received)
mininet>
```

Figure 7.29: Create a Topology With Hub Controller

3. Start Tcpdump and Ping h2 from h1 will notice that h3 also will receive those packets

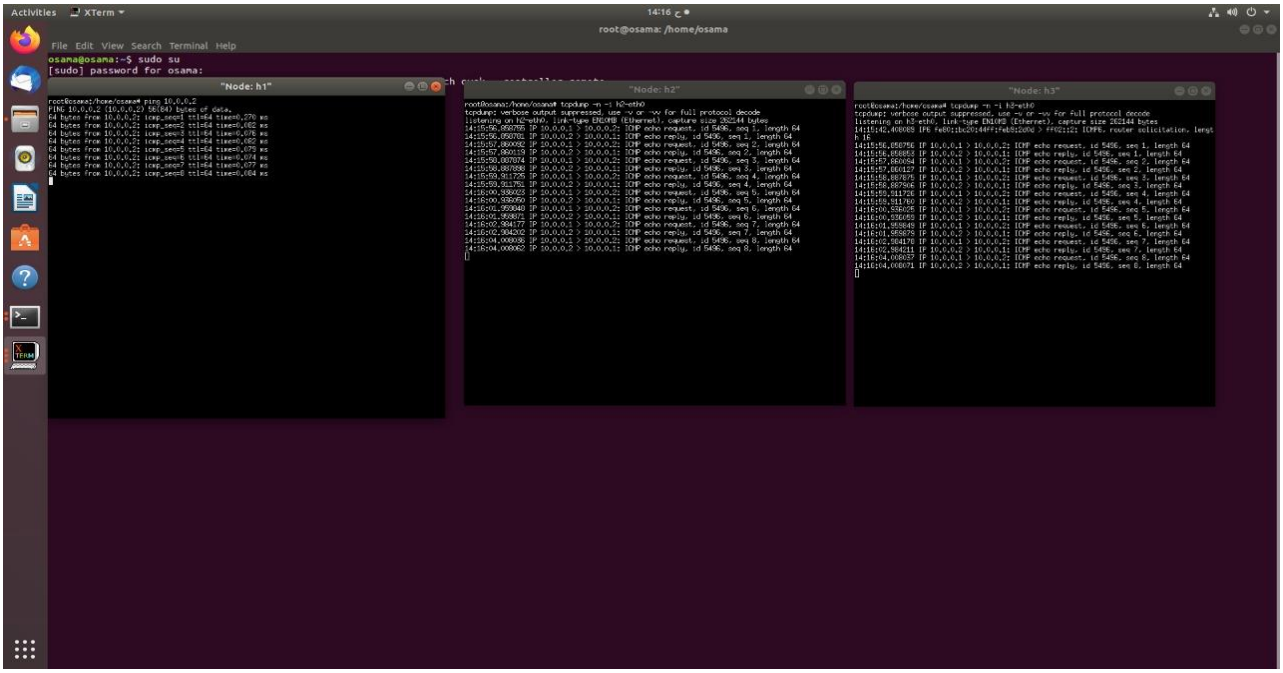


Figure 7.30: Checking Traffic

7.8.3.2 Pox as a forwarding.l2_learning

1. Run a Controller as a L2 Switch

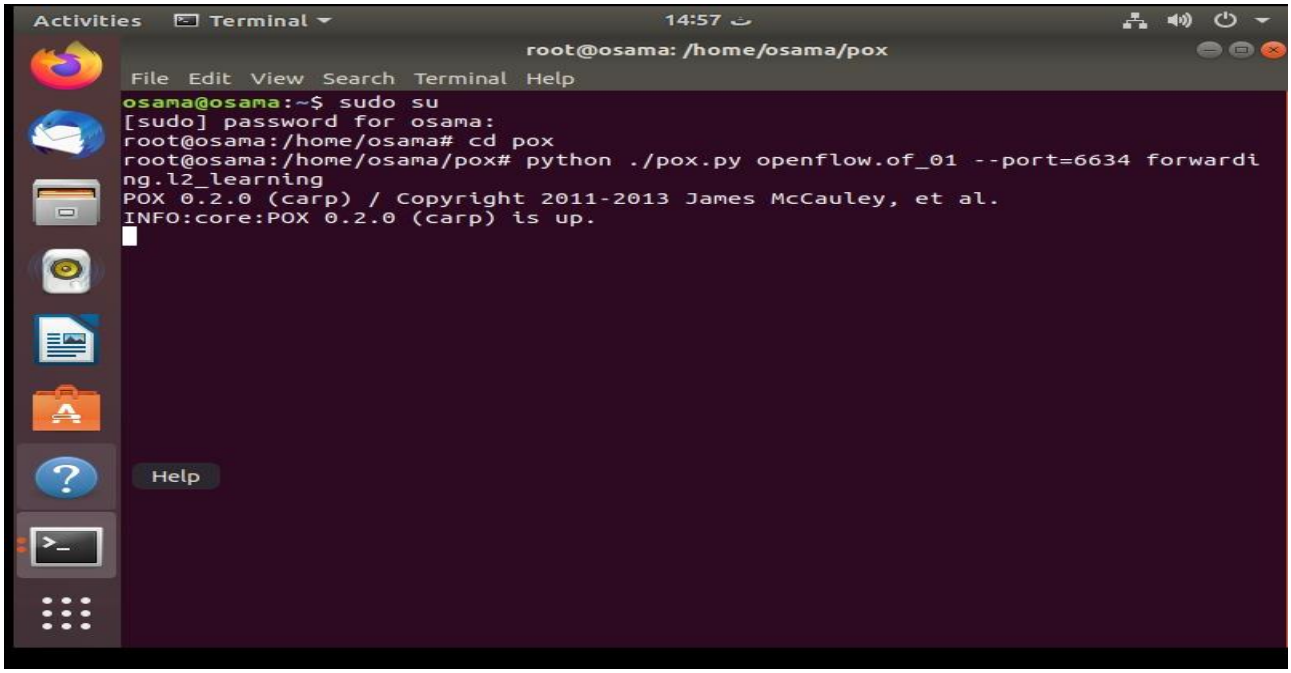


Figure 7.31: Controller as L2 Switch

2. Create a Topology

```
osama@osama:~$ sudo su
[sudo] password for osama:
root@osama:/home/osama# sudo mn --topo single,6 --controller=remote,port=6633
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6
h2 -> h1 h3 h4 h5 h6
h3 -> h1 h2 h4 h5 h6
h4 -> h1 h2 h3 h5 h6
h5 -> h1 h2 h3 h4 h6
h6 -> h1 h2 h3 h4 h5
*** Results: 0% dropped (30/30 received)
mininet>
```

Figure 7.32: Pingall Checking for Test the Controller

7.8.3.3 POX as a load balancer

1. First making a topology of 6 hosts that later 3 of them will be used as servers.

```
osama@osama:~$ sudo su
[sudo] password for osama:
root@osama:/home/osama# sudo mn --topo single,6 --controller=remote,port=6633
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6
h2 -> h1 h3 h4 h5 h6
h3 -> h1 h2 h4 h5 h6
h4 -> h1 h2 h3 h5 h6
h5 -> h1 h2 h3 h4 h6
h6 -> h1 h2 h3 h4 h5
*** Results: 0% dropped (30/30 received)
mininet>
```

Figure 7.33: making a topology with 3 hosts using as servers

2. Then you make sure the controller is up.

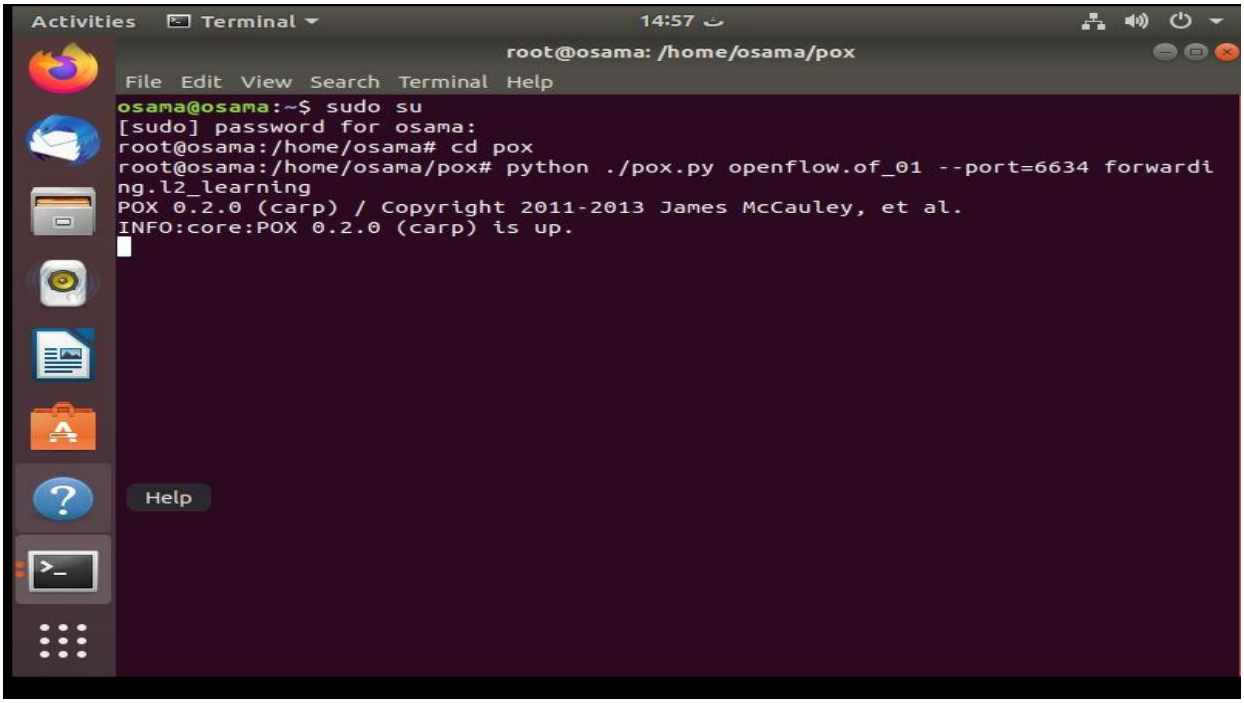


Figure 7.34: Controller is up

3. Then add the load balancing command:

```
pox/pox.py log.level --DEBUG misc.ip_loadbalancer --ip=10.0.1.1
--servers=10.0.0.1,10.0.0.2,10.0.0.3
```

4. now we use 3 hosts as servers.

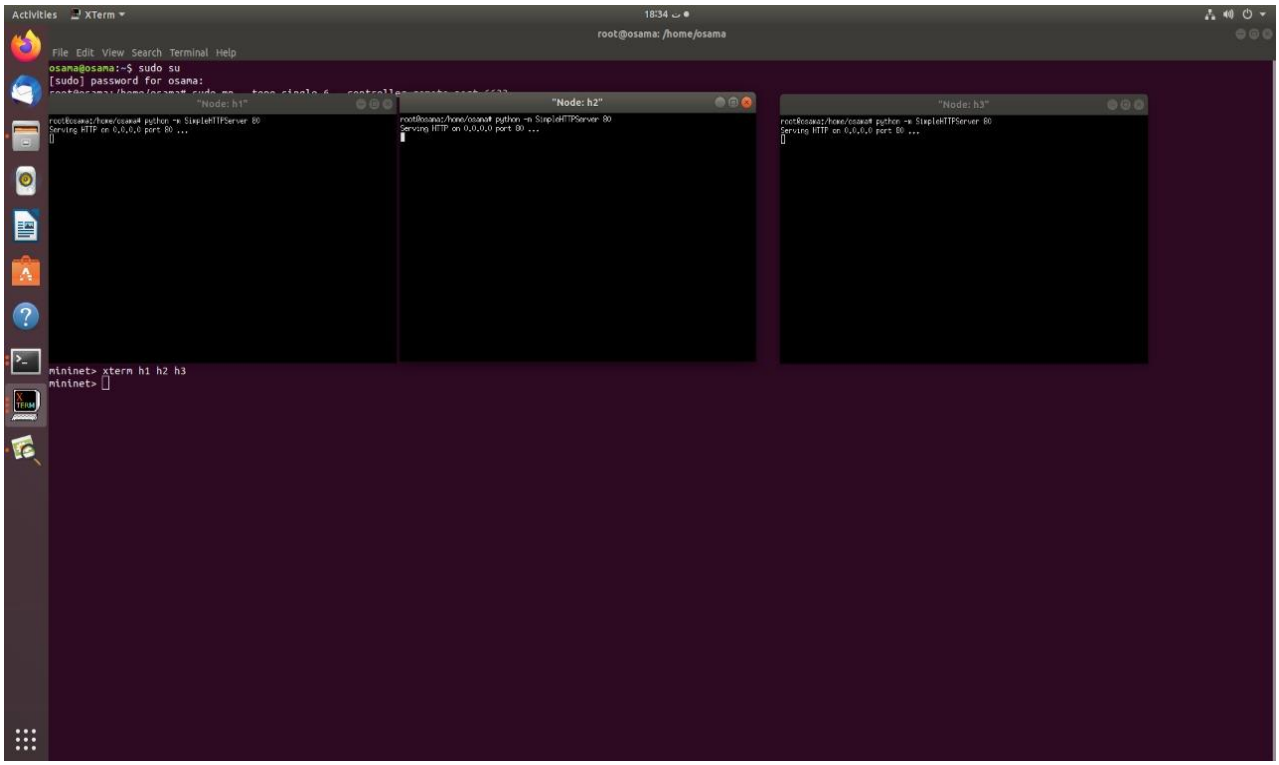


Figure 7.35: Hosts as a server

5. And start curling with the other hosts

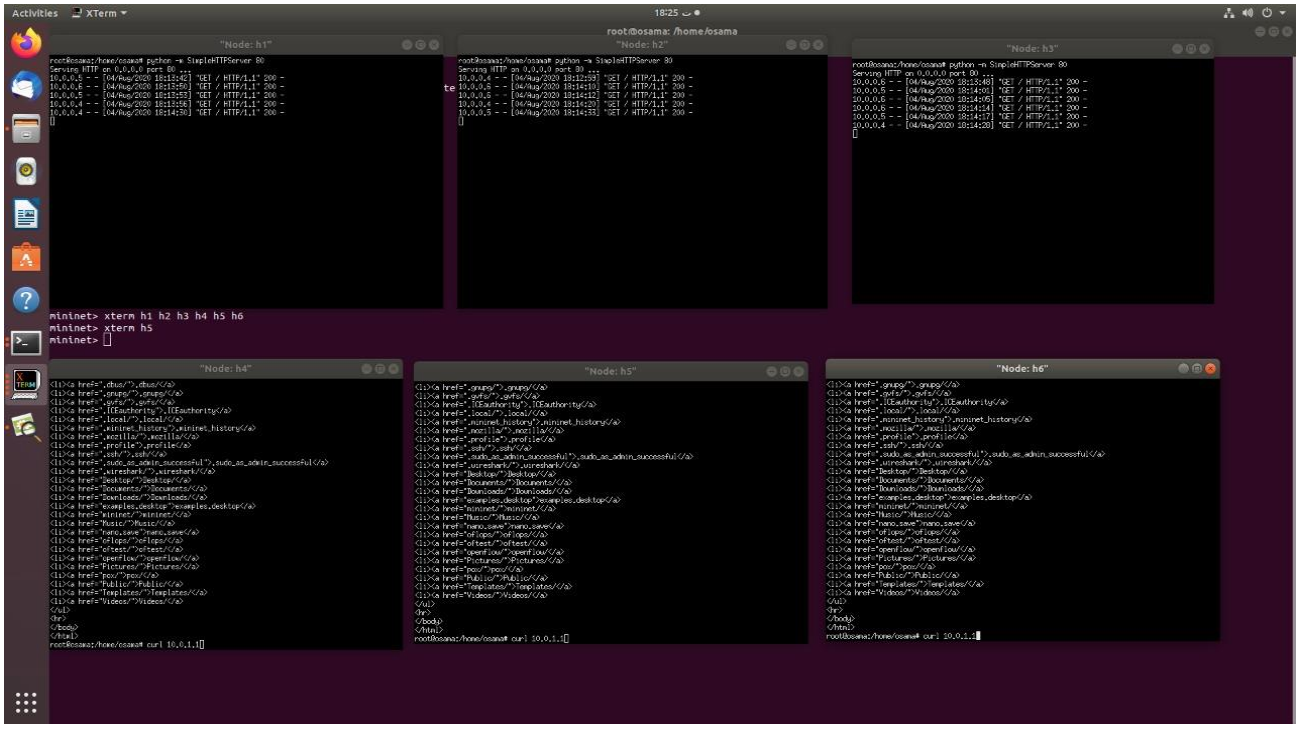


Figure 7.36: Curl Command Output

6. This figure shows the controller dividing packages randomly over the two servers ”2 hosts”

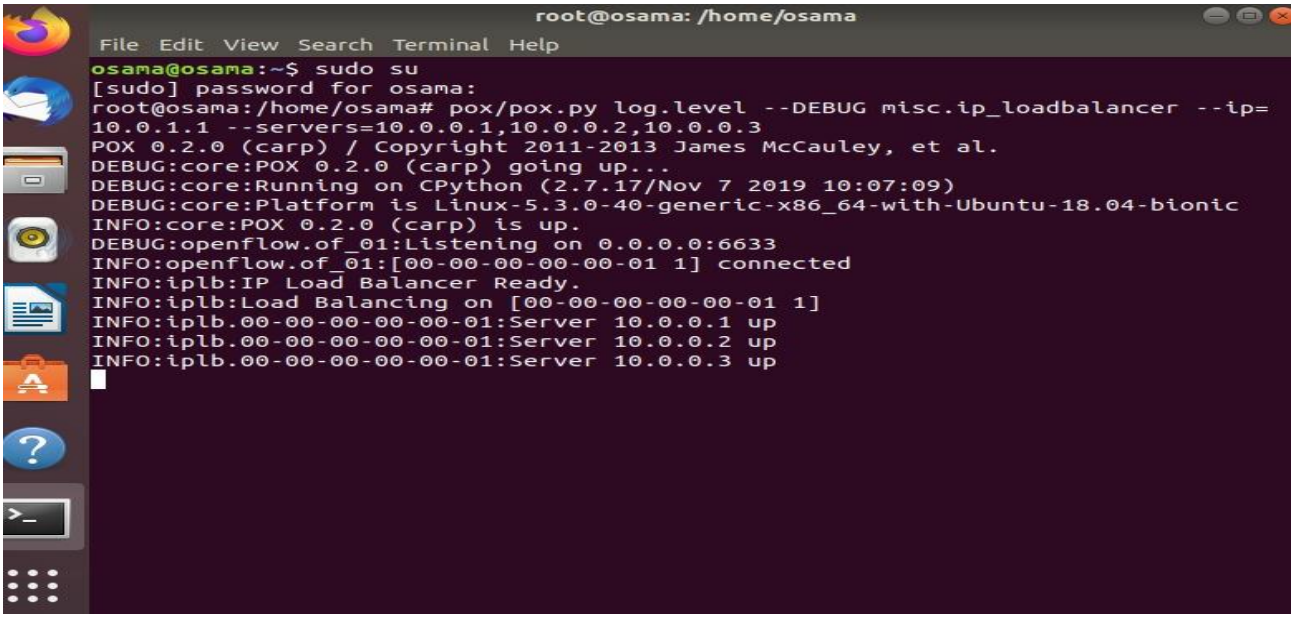


Figure 7.37: Packets Divided Randomly

Chapter 8

Tools

8.1 Openload

Introduction:

OpenWebLoad is a tool for load testing web applications. It aims to be easy to use and providing near real-time performance measurements of the application under test. This is particularly useful when you are doing optimization as you can see the impact of your changes almost immediately.

Quick Install Steps:

- Step 1: Download tool from
  https://sourceforge.net/projects/openwebload/files/openload/
- Step 2: compile using the normal procedure:

```
./configure
make
make install
```

another method in ubuntu 18.04:

- Download the file tool from this version only – 0.0.0 to 0.1.1 -
- Extract this file in your user or root user.
- built in your topology and run it on host with command:

```
user@ubuntu# openlaod ( ip of server ) ( number of clients )
```

How to use it:

OpenWebLoad is (currently) a command line tool, that you execute from a prompt like this:

```
openload [options] http://[num of clients]
```

The 2 parameters are:

- The URL of the web page you want to test.
- Number of simultaneous clients to simulate. This is optional and defaults to 5.

Available options:

The syntax for OpenWebLoad is:

```
openload [options] url [clients]
```

The options are:

- **-t** Test mode. In this mode no throughput measurements are done. However, the full response from the webserver (including headers) is displayed. This is useful for verifying that you get the right content before executing the real test. In this mode the number of clients defaults to 1.
- **-h** header value Specifies a HTTP request header. You can use this option several times to specify several headers. E.g. to specify the User-Agent header you can do:

  openload -h User-Agent "MSIE 5.0" mysite.com/someapp
- **-l** seconds Time limit. The test will only run for the specified number of seconds.

- **-o** output mode Output mode. Currently only csv (comma separated values) is supported.
- This is useful for importing the result in a spreadsheet. The fields are:
    - URL
    - Number of clients
    - TPS
    - Average response time (seconds)
    - Maximum response time
    - Total number of requests

Output Results:

```
$ openload localhost 10
```

URL: http://localhost:80/

Clients: 10

MaTps 355.11, Tps 355.11, Resp Time 0.015, Err 0%, Count 511

MaTps 339.50, Tps 199.00, Resp Time 0.051, Err 0%, Count 711

MaTps 343.72, Tps 381.68, Resp Time 0.032, Err 0%, Count 1111

MaTps 382.04, Tps 727.00, Resp Time 0.020, Err 0%, Count 1838

MaTps 398.54, Tps 547.00, Resp Time 0.018, Err 0%, Count 2385

MaTps 425.78, Tps 670.90, Resp Time 0.014, Err 0%, Count 3072

Total TPS: 452.90

Avg. Response time: 0.021 sec.

Max Response time: 0.769 sec

Where:

- MaTps: a 20 second moving average of TPS.
- Tps: (Transactions Per Second) is the number of completed requests during that second.
- Resp Time: the average response time in seconds for the elapsed second.
- Err: the percentage of responses that was erronous, i.e. didn't return a HTTP 200 Ok staus.
- Count: the total number of completed requests.
- Total TPS is the average TPS for the whole run, i.e. (Total completed requests) / (Total elapsed time).
- Avg. Response time: the overall average response time in seconds.
- Max Response time: the highest response time during this run.

Note: you stop the run by pressing Enter.

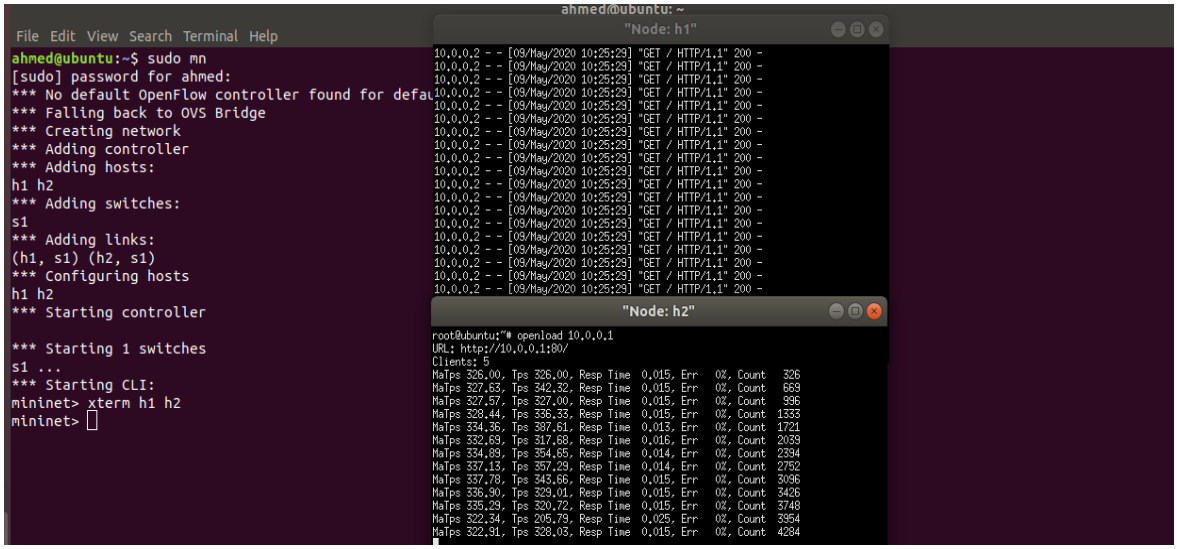


Figure 8.1: OpenLoad tool Output

8.2 Httpref

Introduction:

Httpref is a tool for measuring web server performance. It provides a flexible facility for generating various HTTP workloads and for measuring server performance.

The focus of Httpref is not on implementing one particular benchmark but on providing a robust, high-performance tool that facilitates the construction of both micro- and macro-level benchmarks. The three distinguishing characteristics of Httpref are its robustness, which includes the ability to generate and sustain server overload, support for the HTTP/1.1 and SSL protocols, and its extensibility to new workload generators and performance measurements.

Quick Install Steps:

- Step 1: sudo apt-get update –y
- Step 2: sudo apt-get install -y Httpref

How to use it :

- The simplest way to invoke Httpref is with a command line of the form:

```
httperf --server wailua --port 6800
```

- This command results in Httpref attempting to make one request for URL http://wailua:6800/. After the reply is received, performance statistics will be printed and the client exits (the statistics are explained below).

- A list of all available options can be obtained by specifying the --help option (all option names can be abbreviated as long as they remain unambiguous).
- A more realistic test case might be to issue 100 HTTP requests at a rate of 10 requests per second. This can be achieved by additionally specifying the --num-conns and --rate options. When specifying the --rate option, it's generally a good idea to also specify a timeout value using the --timeout option. In the example below, a timeout of one second is specified (the ramification of this option will be explained later):

  httperf --server wailua--port 6800 --num-conns100 --rate10--timeout 1

appropriate timeout values:

- Since the client machine has only a limited set of resource available, it cannot sustain arbitrarily high HTTP request rates. One limit is that there are only roughly 60,000 TCP port numbers that can be in use at any given time. Since, on HP-UX, it takes one minute for a TCP connection to be fully closed (leave the TIME_WAIT state), the maximum rate a client can sustain is about 1,000 requests per second.
- The actual sustainable rate is typically lower than this because before running out of TCP ports, a client is likely to run out of file descriptors (one file descriptor is required per open TCP connection). By default, HP-UX 10.20 allows 1024 file descriptors per process. Without a watchdog timer, httperf could potentially

quickly use up all available file descriptors, at which point it could not induce any new load on the server (this would primarily happen when the server is overloaded). To avoid this problem, Httpref requires that the web server must respond within the time specified by option --timeout. If it does not respond within that time, the client considers the connection to be "dead" and closes it (and increases the "client-timo" error count). The only exception to this rule is that after sending a request, Httpref allows the server to take some additional time before it starts responding (to accommodate HTTP requests that take a long time to complete on the server). This additional time is called the "server think time" and can be specified by option --think-timeout. By default, this additional think time is zero, so by default the server has to be able to respond within the time allowed by the --timeout option.

- In practice, we found that with a --timeout value of 1 second, an HP 9000/735 machine running HP-UX 10.20 can sustain a rate of about 700 connections per second before it starts to run out of file descriptor (the exact rate depends, of course, on a number of factors). To achieve web server loads bigger than that, it is necessary to employ several independent machines, each running one copy of Httpref. A timeout of one second effectively means that "slow" connections will typically timeout before TCP even gets a chance to retransmit (the initial retransmission timeout is on the order of 3 seconds). This is usually OK, except that one should keep in mind that it has the effect of truncating the connection life time distribution.

Total Results:

- The "Total" line summarizes how many TCP connections were initiated by the client, how many requests it sent, how many replies it received, and what the total test duration was. The line below shows that 100 connections were initiated, 100 requests were performed and 100 replies were received. It also shows that total test-duration was 9.905 seconds meaning that the average request rate was almost exactly 10 request per second.

Total: connections 100 requests 100 replies 100 test-duration 9.905 s

Connection Results

- These results convey information related to the TCP connections that are used to communicate with the web server.

  Specifically, the line below show that new connections were initiated at a rate of 10.1 connections per second. This rate corresponds to a period of 99.1 milliseconds per connection. Finally, the last number shows that at most one connection was open to the server at any given time.

  Connection rate: 10.1 conn/s (99.1 ms/conn, <=1 concurrent connections)

- The next line in the output gives lifetime statistics for successful connections. The lifetime of a connection is the time between a TCP connection was initiated and the time the connection was closed. A connection is considered successful if it had at least one request that resulted in a reply from the server. The line shown below indicates that the minimum ("min") connection lifetime was 4.6 milliseconds, the average ("avg") lifetime was 5.6 milliseconds,

the maximum ("max") was 19.9 milliseconds, the median ("median") lifetime was 4.5 milliseconds, and that the standard deviation of the lifetimes was 2.0 milliseconds.

Connection time [ms]: min4.6 avg5.6 max 19.9 median 4.5 stddev 2.0

- To compute the median time, Httpref collects a histogram of connection lifetimes. The granularity of this histogram is currently 1 millisecond and the maximum connection lifetime that can be accommodated with the histogram is 100 seconds (these numbers can be changed by editing macros BIN_WIDTH and MAX_LIFETIME in stat/basic). This implies that the granularity of the median time is 1 millisecond and that at least 50% of the lifetime samples must have a lifetime of less than 100 seconds.
- The next statistic in this section is the average time it took to establish a TCP connection to the server (all successful TCP connections establishments are counted, even connections that may have failed eventually). The line below shows that, on average, it took 1.4 milliseconds to establish a connection.

  Connection time [ms]: connect 1.4

- The final line in this section gives the average number of replies that were received per connection. With regular HTTP/1.0, this value is at most 1.0 (when there are no failures), but with HTTP Keep-Alive or HTTP/1.1 persistent connections, this value can be arbitrarily high, indicating that the same connection was used to receive multiple responses.

  Connection length [replies/conn]: 1.000

Request Results:

- The first line in the “Request”-related results give the rate at which HTTP requests were issued and the period-length that the rate corresponds to. In the example below, the request rate was 10.1 requests per second, which corresponds to 99.1 milliseconds per request.

  Request rate: 10.1 req/s (99.1 ms/req)

  As long as no persistent connections are employed, the “Request” results are typically very similar or identical to the “Connection” results. However, when persistent connections are used, several requests can be issued on a single connection in which case the results would be different.
- The next line gives the average size of the HTTP request in bytes. In the line show below, the average request size was 57 bytes.

  Request size [B]: 57.0

Reply Results:

- For simple measurements, the section with the "Reply" results is probably the most interesting one. The first line gives statistics on the reply rate:

  Reply rate [replies/s]: min 10.0 avg 10.0 max 10.0 stddev 0.0 (1 samples)

  The line above indicates that the minimum ("min"), average ("avg"), and maximum ("max") reply rate was ten replies per second. Given these numbers, the standard deviation is, of course, zero. The last number shows that only one reply rate sample was acquired. The present version of Httpref collects one

rate sample about once every five seconds. To obtain a meaningful standard deviation, it is recommended to run each test long enough so at least thirty samples are obtained---this would correspond to a test duration of at least 150 seconds, or two and a half minutes.

- The next line gives information on how long it took for the server to respond and how long it took to receive the reply. The line below shows that it took 4.1 milliseconds between sending the first byte of the request and receiving the first byte of the reply. The time to "transfer", or read, the reply was too short to be measured, so it shows up as zero (as we'll see below, the entire reply fit into a single TCP segment and that's why the transfer time was measured as zero).

Reply time [ms]: response 4.1 transfer 0.0

- Next follow some statistics on the size of the reply---all numbers are reported in bytes. Specifically, the average length of reply headers, the average length of the content, and the average length of reply footers are given (HTTP/1.1 uses footers to realize the "chunked" transfer encoding). For convenience, the average total number of bytes in the replies is also given. In the example below, the average header length ("header") was 219 bytes, the average content length ("content") was 204 bytes, and there were no footers ("footer"), yielding a total reply length of 423 bytes on average.

Reply size [B]: header 219.0 content 204.0 footer 0.0 (total 423.0)

- The final piece in this section is a histogram on the status codes received in the replies. The example below shows that all 100 replies were "successful" replies as they contained a status code of 200 (presumably):

  Reply status: 1xx=0 2xx=100 3xx=0 4xx=0 5xx=0

Miscellaneous Results:

- This section starts with a summary of the CPU time the client consumed. The line below shows that 2.71 seconds were spent executing in user mode ("user"), 7.08 seconds were spent executing in system mode ("system") and that this corresponds to 27.4% user mode execution and 71.5% system execution. The total utilization was almost exactly 100%, which is expected given that Httpref is a CPU hog:

  CPU time [s]: user 2.71 system 7.08 (user 27.4% system 71.5% total 98.8%)

  Note that any time the total CPU utilization is significantly less than 100%, some other processes must have been running on the client machine while Httpref was executing. This makes it likely that the results are "polluted" and the test should be rerun.
- The next line gives the average network throughput in kilobytes per second (where a kilobyte is 1024 bytes) and in megabits per second (where a megabit is 10^6 bit). The line below shows an average network bandwidth of about 4.7 kilobyte per second. The megabit per second number is zero due to rounding errors.

Net I/O: 4.7 KB/s (0.0*10^6 bps)

The network bandwidth is computed from the number of bytes sent and received on TCP connections. This means that it accounts for the network payload only (i.e., it doesn't account for protocol headers) and does not take into account retransmissions that may occur at the TCP level.

"Errors"

"Errors"

- The final section contains statistics on the errors that occurred during the test. The "total" figure shows the total number of errors that occurred. The two lines below show that in our example run there were no errors:

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

- The meaning of each error is described below:

1. total: The sum of all following error counts.

2. client-timo: Each time a request is made to the server, a watchdog timer is started. If no (partial) response is received by the time the watchdog timer expires, Httpref times out that request a increments this error counter. This is the most common error when driving a server into overload.

3. socket-timo: The number of times a TCP connection failed with a socket-level time out (ETIMEDOUT).

4. Connrefused: The number of times a TCP connection attempt failed with a "connection refused by server" error (ECONNREFUSED).

5. Connreset: The number of times a TCP connection failed due to a reset (close) by the server.

6. fd-unavail: The number of times the Httpref client was out of file descriptors. Whenever this count is bigger than zero, the test results are meaning less because the client was overloaded (see discussion on setting --timeout below).

7.Addrunavail: The number of times the client was out of TCP port numbers (EADDRNOTAVAIL). This error should never occur. If it does, the results should be discarded.

8. ftab-full: The number of times the system's file descriptor table was full. Again, this error should never occur. If it does, the results should be discarded.

9. Other: The number of times other errors occurred. Whenever this occurs, it is necessary to track down the actual error reason. This can be done by compiling Httpref with debug support and specifying option --debug 1.

8.3 Siege

Introduction:

Siege is a Hypertext Transfer Protocol (HTTP) and HTTPS load testing and web server benchmarking utility developed by Jeffrey Fulmer. It was designed to let web developers measure the performance of their code under stress, to see how it will stand up to load on the internet.

Siege can stress a single URL or it can read many URLs into memory and stress them simultaneously. It supports basic authentication, cookies, HTTP, HTTPS and FTP protocols.

Quick Install Steps:

Siege is multi-platform and can be installed under Ubuntu/Debian distribution using following commands:

```
$ sudo apt install siege
```

Then you can download Siege using wget command and install from sources as shown.

```
$ wget http://download.joedog.org/siege/siege-latest.tar.gz
$ tar -zxvf siege-latest.tar.gz
$ cd siege-*/
$ sudo ./configure --prefix=/usr/local --with-
ssl=/usr/bin/openssl
$ sudo make && make install
```

Configuring Siege HTTP Load Testing:

Once you have completed the installation, you can adjust your siege configuration file. It is located in **/etc/siege/siegerc**. In case you have decided to build the package from source, you will have to run:

```
$ sudo siege.config
```

This will generate siege.conf file located in your user's home **~/.siege/siege.conf.**

The contents of the file should look something like this. Note that I have uncomment the logfile and time directives:

```
$ cat siegerc |egrep -v "^$|#"
```

With the current configuration, siege will imitate 25 concurrent users over 1 minute. You are now ready to run your siege.

You can also use command line options, if you want to try different settings from the ones described in the configuration file.

- **C –** specify your own configuration file.
- **q –** suppresses siege's output.
- **g –** GET, pull down HTTP headers and display the transaction. Useful for debugging.
- **c –** the number of concurrent users, default is 10.
- **r –** how many times to run the test.
- **t –** how much time to run the test. You can specify S, M, or H ex: –time=10S for 10 seconds.
- **d –** random delay before each request.
- **b –** no delays between requests.

- **i –** user simulation. Uses to hit random URLs.
- **f –** test URLs from specified file.
- **L –** log file.
- **H –** Add a header to request.
- **A –** specify a user agent.
- **T** – Sets Content-Type in request.

no-parser – NO PARSER, turn off the HTML page parser.

no-follow – do not follow HTTP redirects.

use the -f option like this:

```
$ siege -f /usr/local/etc/urls.txt
```

Output Results:

Performance measures include elapsed time of the test, the amount of data transferred (including headers), the response time of the server, its transaction rate, its throughput, its concurrency and the number of times it returned OK. These measures are quantified and reported at the end of each run.

This is a sample of siege output:

```
shrouk@ubuntu:~$ siege -C
New configuration template added to /home/shrouk/.siege
Run siege -C to view the current settings in that file
CURRENT  SIEGE  CONFIGURATION
Mozilla/5.0 (pc-x86_64-linux-gnu) Siege/4.0.4
Edit the resource file to change the settings.
------------------------------------------------
version:                        4.0.4
verbose:                        false
color:                          true
quiet:                          false
debug:                          false
protocol:                       HTTP/1.1
HTML parser:                    enabled
get method:                     HEAD
connection:                     close
concurrent users:               25
time to run:                    n/a
repetitions:                    n/a
socket timeout:                 30
cache enabled:                  false
accept-encoding:                gzip, deflate
delay:                          0.000 sec
internet simulation:            false
```

Figure 8.2: Siege tool Options Output part 1

```
internet simulation:            false
benchmark mode:                 false
failures until abort:           1024
named URL:                      none
URLs file:                      /etc/siege/urls.txt
thread limit:                   255
logging:                        false
log file:                       /var/log/log/siege.log
resource file:                  /home/shrouk/.siege/siege.conf
timestamped output:             false
comma separated output:         false
allow redirects:                true
allow zero byte data:           true
allow chunked encoding:         true
upload unique files:            true
no-follow:
 - ad.doubleclick.net
 - pagead2.googlesyndication.com
 - ads.pubsqrd.com
 - ib.adnxs.com
```

Figure 8.3: Siege tool Options Output part 2

Testing Website Load with Siege Benchmarking Utility:

Running siege is quite easy, you only need to specify the website you wish to test like this:

```
$ # siege domain.name
```

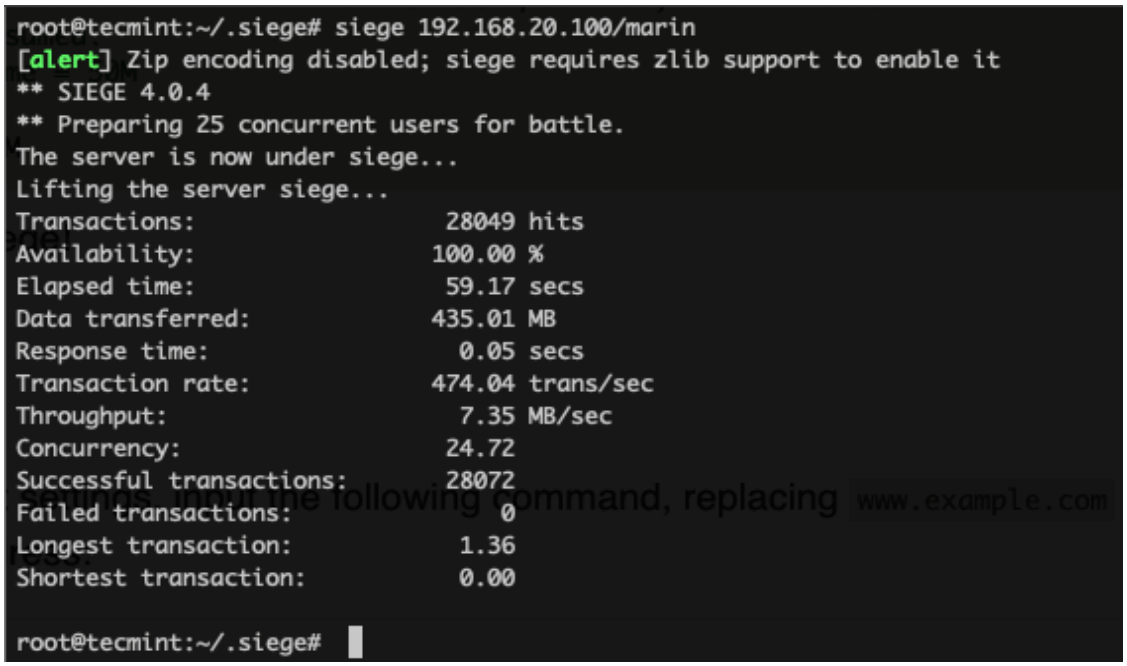


Figure 8.4: Siege tool Output By Domain

Run Siege against Multiple Websites:

You can test multiple URLs, by setting siege to read them from file.

```
root@tecmint:/usr/local/etc# cat urls.txt
# URLS file for siege
# --
# Format the url entries in any of the following formats:
# http://www.whoohoo.com/index.html
# http://www/index.html
# www/index.html
# http://www.whoohoo.com/cgi-bin/howto/display.cgi?1013
# Use the POST directive for pages that require it:
# http://www.whoohoo.com/cgi-bin/haha.cgi POST ha=1&ho=2
#       or POST content from a file:
# http://www.whoohoo.com/melvin.jsp POST </home/jeff/haha
# http://www.whoohoo.com/melvin.jsp POST <./haha
# You may also set and reference variables inside this file,
# for more information, man urls_txt
# ----------------------------------------------------------
192.168.20.100/marin
192.168.20.100/admin
root@tecmint:/usr/local/etc#
```

You can describe the URLs in **/usr/local/etc/urls.txt** like this:

Figure 8.5: Siege tool Test Multiple URL

Chapter 9

Load balancing Algorithms

9.1 Introduction

The load balancing algorithm used determines the selection of backend servers to forward the traffic. This is based on:

- Server health
- Predefined conditions

Different types of load balancing algorithms are meant for different benefits and the choice depends on your needs. To implement load balancing, a load balancer coupled with at least one more additional server is required. Depending on the distribution of the load, whether it is on the network or application layer, algorithms widely vary. Some of the efficient, frequently used network layer load balancing algorithms

9.2 Traditional methods

- Random-based: The controller selects one server randomly from the available servers for each request.
- Round-robin: This technique distributes the requests among the servers in a sequential manner. In other words, the selected server is always the server of the next turn in the network. All servers using this algorithm almost have the same number of requests. This involves a cyclical sequential distribution of request to server and is ideal for a throng comprising of identical servers.

The two types in Round robin are:

1. **Weighted round robin:** It can be used effectively for a cluster of unidentical servers as each server is assigned a weight based on its structure and efficiency. It also follows a cyclical procedure but the server preference happens according to the preassigned efficiency. Thus, efficient servers are loaded more. Besides the server capacity, it is preferred if one server should get a substantially lower number of connections than an equally capable server, since the first server runs critical applications and it should not be easily overloaded.
2. **Dynamic round robin:** It forwards the requests to the associated server based on a real-time calculation of server weights.

- **Connections-based:** This algorithm chooses the server based on minimum active connections.

  The command “netstat” is used to verify how many connections the server has. This command outputs the machine’s current TCP connections. The program then counts how many lines this command outputs and sends a packet to the controller including this information.

9.3 Round Robin Methods:

- **Bandwidth-based (BWBLB):** It selects the server which consumes the least bandwidth for the last 14-seconds interval for processing the requests. “Bwmng” software tool is used to report a summary of the bandwidth consumed by the server at which data

is being transferred in and out of all available network interfaces on the system.

- **Server CPU-based (CBLB):** This algorithm selects the least-loaded
  server in terms of the CPU usage of the server machine.
- **Server CPU-Memory-based (CMBLB):** This scheme is quietly similar to the CBLB but it takes into consideration both the CPU usage and memory usage of the server machine.
- **Least connections:** Here the relative computing capacity of each server is monitored and the server with least active transactions is selected to distribute the load.
- **Weighted least connections:** The load distribution is based on both the number of active connections to each server and the relative server capacity.
- **Source IP hash:** The selection of back-end server is based on a hash of the source IP of the request, like the IP address of the visitor. The distribution is altered if a web node fails and becomes out of service. As long as all servers are running, a particular user is consistently connected to the same server.
- **URL hash:** Hashing is performed on the request URL. This avoids duplication cache, by storing the same object in several caches, thus the effective capacity of the backend caches is increased.
- **The least response time:** The backend server is selected in preference with the least response time, ensuring the end client receives an immediate response.

- **The least bandwidth and the least packets method:** The back-end server is decided by the load balancer depending on the bandwidth consumption for the last fourteen seconds. The associated server which consumes the least bandwidth is selected. Similarly, at least packets method, the server transmitting the least packets receive new requests from the load balancer.
- **The custom load method:** The servers are chosen depending on their load, which is calculated based on the CPU usage, memory and response time. The server load is evaluated by load monitor. Resource utilization can be established efficiently by this method.

9.4 How the system works

1. The clients request the service's virtual IP and forward this request to the OpenFlow switch.

2. The OpenFlow switch forwards the requests to the controller to decide what action should be done.

3. The controller has the updated states of the servers and decides where the packet should be forwarded according to the running application. Then, it adjusts the packet header which contains the destination IP and MAC addresses of the selected server.

4. After the controller chooses the server, it sends a packet including the appropriate action to tell the OpenFlow switch which port should the packet be forwarded to.

5. The OpenFlow switch performs the controller's rule and forwards the packet to the port of the selected server.

6. The selected server replies on the request to the switch.

7. The switch forwards the reply to the corresponding client immediately.

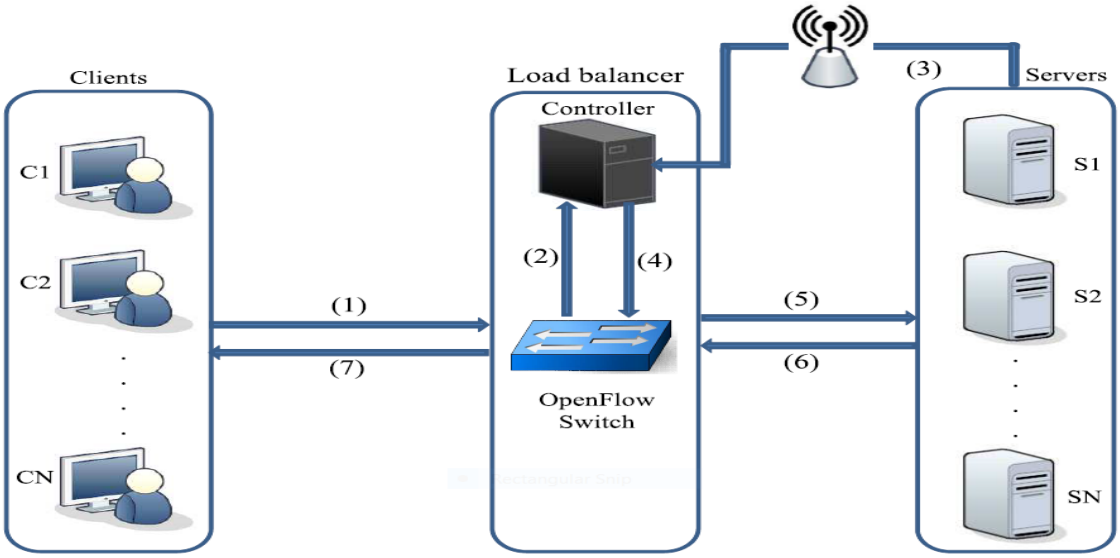


Figure 9.1: Load Balance Hierarchy

Conclusion and Future Work

In order to handle the growing data and make sustainable profits, the traditional networking infrastructure must adapt dynamically, but the limitations imposed by the complexity, vendor dependence, and QoS requirements have leaded to a static network. SDN overrides these limits and helps in making a dynamic network where the network control and management can be independent, programmable and less effortful.
This thesis proposed OpenFlow-based software solutions for servers' overload problem using SDN paradigm by implementing load balancing applications on an OpenFlow environment instead of the traditional load balancer which uses a more dedicated and expensive hardware. The load balancer aims to distribute the clients' requests among the available servers according to some specific criteria. In this thesis, the performance of the proposed load balancing schemes is compared and evaluated with Random-based, Round-robin, and Connections-based strategies in both mini-net emulation and real-time Raspberry Pi testbed environments. The benchmarking tools show that the proposed load
balancing algorithms cannot only improve the throughput and the response time of web servers but also achieving more servers' utilization. It is worth to state that the tests developed in this context can be used in a large scale real physical network. Considering the success of the SDN paradigm and all the identi-54 field challenges in this study, it is observed that there is a huge field for the development of new research projects focused on SDN. As a suggestion for future works, we wish to apply the use of SDN with OpenFlow technology adoption in the process of resource management in cloud computing. The concept of cloud computing is being

increasingly deployed in data centers. The process of resources virtualization brings additional needs for management and control of network elements and taking into accounts both the state of the servers and network links.

One controller is used in out implementation which may lead to a single point of failure.

Therefore, controllers must be configured in distributed mode rather than centralized mode where the single break down affects the whole network performance. Furthermore, load balancing and standby mechanism can also be applied to it.

References

1. http://mininet.org/walkthrough/
2. http://sdnhub.org/tutorials/pox/
3. T. Bourke. “server load balancing”, O’Reilly Media. 2001
4. https://support.hpe.com/hpesc/public/docDisplay?docId=c05028139
5. “POX OpenFlow Controller”, 2017, available at, https://OpenFlow.stanford.edu/display/ONL/POX+Wiki
6. http://www.brianlinkletter.com/how-to-use-miniedit-mininets-graphical-user-interface/
7. https://en.wikipedia.org/wiki/Software-defined_networking
8. Open Networking Foundation, ”Software-defined networking: The new norm for networks.” ONF White Paper, Apr. 2012.
9. “Ryu SDN Framework”, 2017, available at, https://osrg.github.io/ryu/
10. “SDN”, 2017, available at, http://www.cse.wustl.edu/%7Ejain/cse570-13/ftp/sdn/index.html
11. “Software-Defined Networking: The New Norm for Networks”, 2017, Available at:

https://www.opennetworking.org/images/stories/downloads/sdn-resources/whitepapers/wp-sdn-newnorm.pdf

12. "Ferro, G. OpenFlow and Software Defined Networking ipspace webinar", 2017,

Available at: http://demo.ipspace.net/get/OpenFlow

13. M. Nispel . "A Different Take on Software Defined Networks", 2013, Available at: http://www.extremenetworks.com/a-different-take-on-software-definednetworks.

14. M. B. Navarro, "Dynamic Load Balancing in Software-Defined Networks," M.S. thesis, Dept. Electronic Systems., Aalborg Univ., Tampere, Danmark, 2014.

15. N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-Serve: Load-balancing web traffic using OpenFlow," Demo at ACM SIGCOMM,Aug. 2009.

16. "Mininet", 2013, available at, http://mininet.org/.

17. "Raspberry Pi price", 2017, available at, https://www.raspberrypi.org/blog/raspberry-pi-3-on-sale/

18. "OpenFlow Switch Specification v1.3.1". 2016, Available at:

https://www.opennetworking.org/images/stories/downloads/sdn-resources/onfspecifications/openflow/openflow-spec-v1.3.1.pdf

19. "SDN Tutorial for Engineers".2012, available at: http://www.opennetsummit.org/archives/apr12/heller-mon-intro.pdf

20. https://github.com/ameyap13/SDN-LOAD-BALANCER/blob/master/our_topology.py

21. https://github.com/ameyap13/SDN-LOAD-BALANCER/blob/master/README.md

23. https://github.com/priyankvora/POX_Controller-Load_Balancing