TRAFFIC ENGINEERING WITH ROUTING PROTOCOLS USING SOFTWARE DEFINED NETWORKING AND NETWORK FUNCTIONS VIRTUALIZATION ARCHITECTURE

A PROJECT REPORT

Presented to the Department of Electrical Engineering

California State University, Long Beach

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

Committee Members:

Hen-Geul Yeh, Ph.D., P.E. (Chair)
Hengzhao Yang, Ph.D.
Walter Martinez, M.A.

College Designee:

Antonella Sciortino, Ph.D.

By Bhargava Bokkena

B.E., 2013, AIIT- Hyderabad, India

May 2017

ABSTRACT

TRAFFIC ENGINEERING WITH ROUTING PROTOCOLS USING SOFTWARE DEFINED NETWORKING AND NETWORK FUNCTIONS VIRTUALIZATION ARCHITECTURE

By

Bhargava Bokkena

May 2017

Software-Defined Networks (SDNs) have enforced greater flexibility in deploying and managing networks. Network virtualization is driving the Telecom Industry today. Open v Switch is an open source virtual switch that gives industry grade performance levels on various virtualized environments. This project mainly concentrates on the traffic engineering of the SDN networks and the performance it can provide compared to the legacy networks on Virtualized platforms. This project shows how the services can be chained, as in legacy network, in virtualized environments for enabling Network Functions Virtualization (NFV) for the future. Service Function Chaining (SFC) policies are designed and applied on the underlying switch for different Virtual Network Functions (VNFs). Performance analysis is performed using the ‘pktgen’ as the traffic generator and open v switch as the soft switch. Different use cases of porting in Network Functions as virtual elements and their performance metrics with reduced cost and operations are shown.

ACKNOWLEDGMENTS

I take immense pleasure in thanking the Electrical Engineering Chair, Hen-Geul Yeh, and other committee members, for having permitted me to carry out this project. I wish to express my gratitude to my project advisor, Dr. Boi Tran, for his guidance, encouragement, and suggestions, which helped me in completing the project on time. He has been a source of inspiration throughout the project, and his appropriate guidance in providing the artifacts is worth mentioning.

I sincerely thank the California State University, Long Beach Electrical Engineering Department for their encouragement and cooperation in carrying out the project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AMQP	Advanced Message Queuing Protocol
API	Application Program Interface
ARP	Address Resolution Protocol
CPU	Central Processing Unit
DNS	Domain-Name Service
FPGA	Field Programmable Gate Array
IP	Internet Protocol
IT	Information Technology
NAT	Network Address Translation
NFV	Network Functions Virtualization
NFVI	Network Functions Virtualization Infrastructure
ODL	OpenDay Light
OSGI	Open Service Gateway Initiative
OVDK	Open Virtual Development Kit
OVS	Open Virtual Switch
PC	Personal Computer
PIF	Physical Interface
RR-DNS	Round Robin-Domain Name Service
SDN	Software Defined Network
SF	Service Function
SFC	Service Function Chaining
VIF	Virtual Interface

VIP	Virtual Internet Protocol
VM	Virtual Machine
VNF	Virtualized Network Function

CHAPTER 1

INTRODUCTION

The constant development and rigorous application of cloud networking makes traditional network architecture unsuitable for present day networking. IT Industry and the Telecom Sector are mainly dependent on the Virtualization and Software-Defined Networking (SDN) for their growth. Numerous data centers are virtualized to provide faster provisioning overflow to the cloud, and enhance accessibility at times of disaster. This need is satisfied with the development of Network Functions Virtualization (NFV) and SDN ideas.

The main challenge for the organizations and data centers is to successfully transform from the existing traditional systems to SDN and NFV. This project discusses the basic concepts of SDN and NFV and helps in deploying them without any errors. NFV is a network design idea that uses the advances of IT virtualization in virtualizing the whole class of network hub functions into building objects that may associate or chain together to make administrations easy and simple. NFV enables organizations in replacing hardware systems with software-based Virtual Network Functions (VNFs). This creates a much more flexible system while reducing costs.

SDN is a way to cope with PC networking that permits network administrators to oversee network benefits through the reflection of lower-level usefulness. SDN is intended to address that the static engineering of conventional networks does not bolster the dynamic, versatile processing, and capacity needs of more present day figuring situations, for example server farms. Along these lines, both NFV and SDN have a solid potential to shape the future networks. The ascent of server virtualization has carried with it a central move in server farm networking.

Another network to layer has risen in which most network ports are virtual, which has brought about the improvement of software or virtual switches.

A virtual switch is a software program that basically encourages the correspondence between virtual machines or containers existing on the same or diverse physical machines. A virtual switch supplies the fundamental usefulness of a physical switch and provides different elements, for example GRE-burrowing, Multicast-snooping, and Deceivability into or between VM activities. They work considering the ‘control plane-information plane’ design of most SDN structures.

CHAPTER 2

SOFTWARE DEFINED NETWORKING

This chapter describes the SDN structure, differentiates it from the traditional architecture, and also states the advantages and impediments of the SDN. Figure 1 shows the basic SDN structure. SDN controller permits centralized control of the complete network. The switch and the SDN controller communicate through OpenFlow protocols. The switch running OpenFlow forwards incoming packets based on the data in the SDN controller.

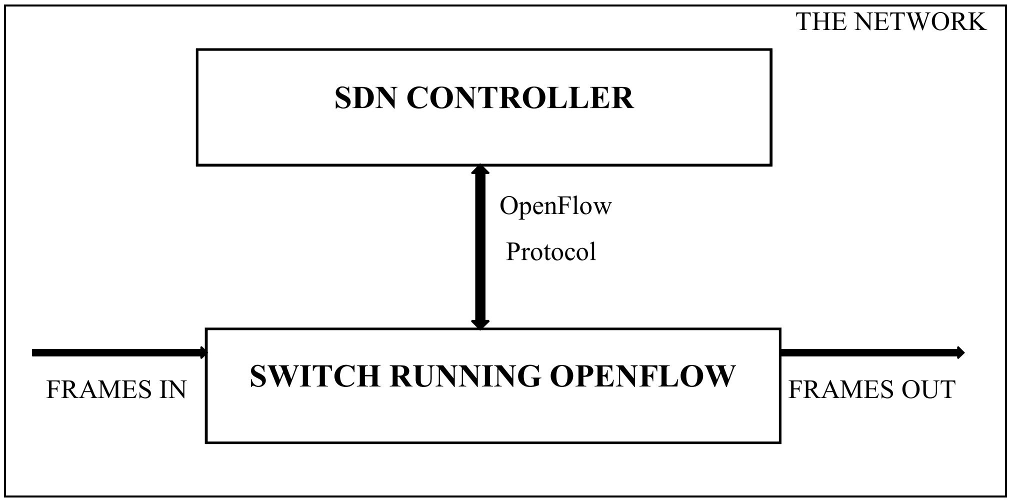


FIGURE 1. SDN design.

Figure 2 shows that the SDN layer has virtual servers and the physical layer has network switches. These virtual servers and the network switches are connected to a centralized controller. The virtual servers and the network switches communicate with the centralized controller through OpenFlow protocols.

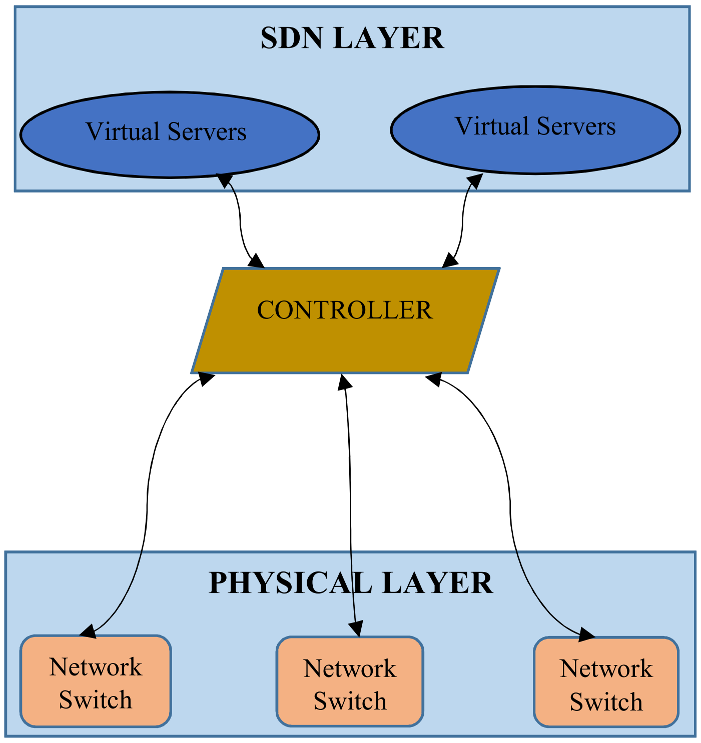


FIGURE 2. SDN architecture.

SDN is a network structure that settles on decisions about where the development is sent from the basic structures (the control plane) to the picked destination (the data plane). This consequently enables the framework control to wind up as the essential base for various applications and framework organizations. The Open Flow tradition is a foundational segment for buildingthe SDN course of action [1].

The SDN is programmable, centrally managed, and vendor neutral. This means it can be directly programmed to the device based on the needs of the user. The management of the network is achieved from a single controller, which can manage the whole network, and this controller can program the network depending on the user's requirements. The configuration of this network can be changed at any time based on the desired interests of the user. SDN enables innovation in designing and managing networks.

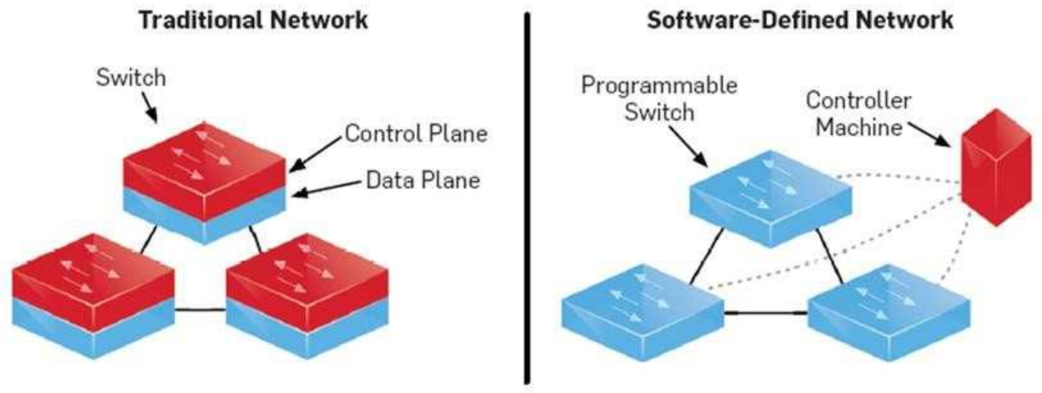


FIGURE 3. Traditional network versus SDN [1].

In traditional networks, the control plane and the data plane exist on the same device. The control planes communicate with each other and make decisions. The data planes interact with each other and send out the incoming packets based on the decisions made in the control plane. SDN decouples the control plane and the data plane [1]. Control planes of all the devices are put together in a single controller called the 'SDN controller' through which a centralized control is achieved. The data planes are replaced with programmable switches which communicate with the SDN controller through OpenFlow protocols [2]. This is shown in Figure 3.

The traditional load balancers are vendor specific, meaning that they function per the algorithm that is being specified by the vendor. The load balancers are completely vendor specified making the vendor the main controller for the functionality unlike SDN, where the user controls the functionality of the network. This makes SDN more favorable for controlling the traffic instead of traditional load balancers.

Even though SDN appears to be the ideal answer for today's gigantic information networking issues, it has impediments of its own. Moving from conventional appropriated networking to brought together approach, there is much burden on the controller itself. Any bundle misfortunes between the controller and switches may lead to negative results, dissimilar to in customary methodology where one single parcel misfortune does not influence the execution to such an extent. Making the entire SDN engineering measured and adaptable is a system configuration challenge when there are thick systems. Changing the system-building design from the current conveyed model to the SDN model inside of a datacenter, while achievable, is unrealistic past hierarchical limits.

SDN is creating development demonstrating that it is dynamic, sensible, fiscally shrewd, and flexible, making it ideal for the high-information transmission, and dynamic to present day's applications. When implemented on NFV platform (a use-case of SDN), SDN yields better results [3].

CHAPTER 3

NETWORK FUNCTIONS VIRTUALIZATION

In this chapter, the concept of NFV and its architecture are described. This chapter also includes the deployment of OpenStack system on a virtual platform as well as the management and orchestration of the virtual resources in the OpenStack.

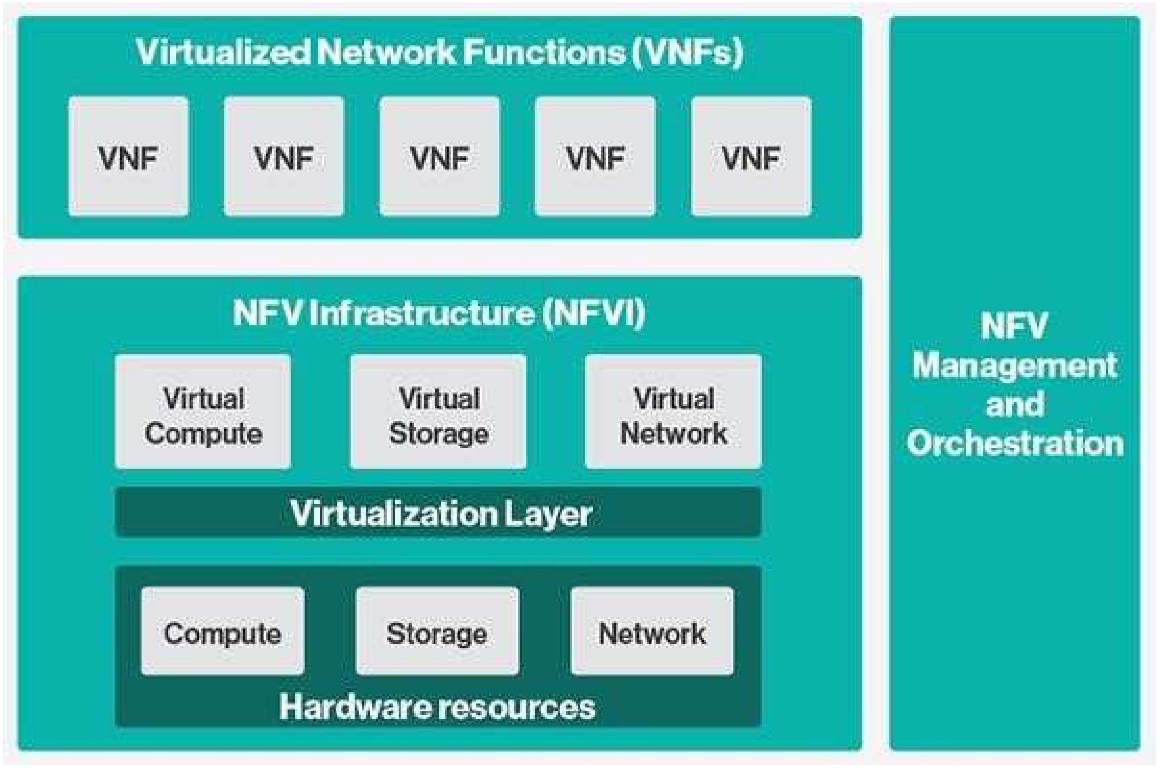


FIGURE 4. NFV structure [3].

Figure 4 shows that the physical layer has hardware resources such as compute, storage, and network. A virtualization layer (hypervisor) is created and virtual resources such as virtual compute, virtual storage, and virtual network exist on the hypervisor. This arrangement forms the Network Functions Virtualization Infrastructure (NFVI) and VNFs exist on top of the NFVI. Creating the virtual elements and managing them are performed in the NFV management and orchestration section. The focus lies on the NFVI that depicts the relation between the hardware resource emulation into virtual resources [3].

NFV is a system design architecture that uses virtualization techniques to oversee center network administration capacities by means of programming instead of relying on equipment to deal with these capacities. The NFV idea depends on building elements of virtualized functions, or VNFs, which can be joined to make full-scale organizing networking administrations. In NFV, software handle specific network functions that run in one or more virtual machines on top of the hardware networking infrastructure (both wired and wireless) such as routers, switches, servers, or cloud computing systems [4]. Examples of the types of networking functions that can be managed by NFV include network security and firewalls, Network Address Translation (NAT), Domain-Name Services (DNS), caching, intrusion detection, and others [4].

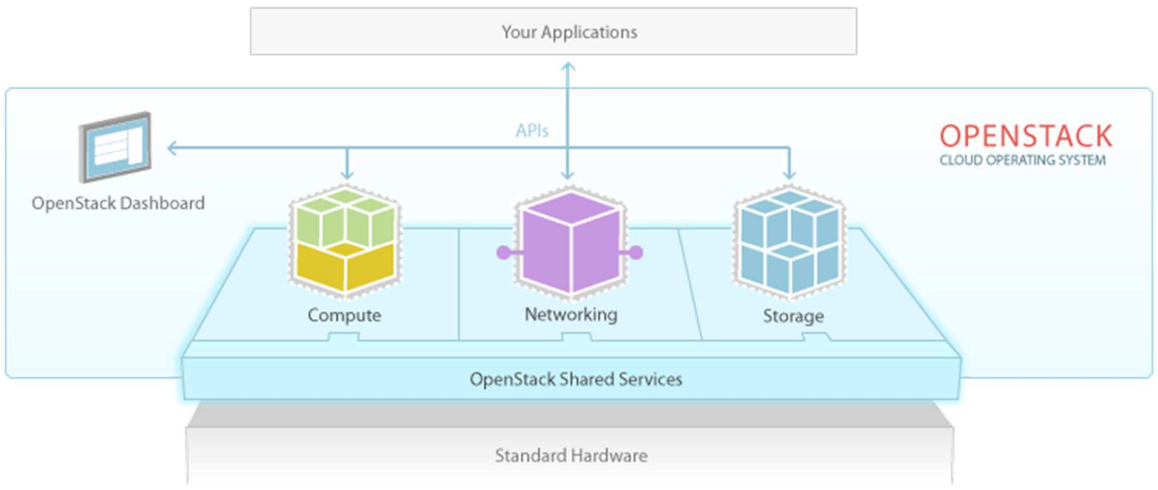


FIGURE 5. OpenStack system [5].

Figure 5 shows the main components of an OpenStack system: compute, networking, and storage. The OpenStack dashboard permits the users to create networks and routers. All the components and the user applications communicate using Application Programmable Interfaces (APIs).

OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that enables administrators control while empowering their users to provision resources through a web interface [5].

OpenStack consists of several independent parts named the OpenStack services. All services authenticate through a common Identity service. Individual services interact with each other through public APIs, except where privileged administrator commands are necessary. Internally, OpenStack services are composed of several processes. All services have at least one API process, which listens for API requests, pre-processes them, and passes them on to other parts of the service. Apart from the Identity service, distinct processes carry the actual work.

AMQP message broker enables the communication between the processes of one service. The service's state is stored in a database. When deploying, and configuring the OpenStack cloud, the user can choose among several message broker and database solutions such as RabbitMQ, MySQL, MariaDB, and SQLite.

Open v Switch (OvS) lies on top of the virtualization layer (hypervisor) and interfaces the VNFs. Each VNF has a Virtual Interface (VIF). VIF enables communication with other VNFs in the same virtual network and forwards traffic from the VNF to remote destinations. OvS enables bridging of VNFs in the same virtual network and communication with other virtual networks or physical networks via the Physical Interfaces (PIFs) [6]. OvS acts as a central component in the communication domain of virtual machines by interconnecting PIFs with the VIFs [6].

NFV allows flexibility in moving from specified equipment to general servers, which eases the burden on the service providers. It is secure, cost-effective, scalable, and easily adaptable.

CHAPTER 4

DESIGN OF SDN LOAD BALANCER AND FIREWALL

The importance of load balancer, the requirements for creating a SDN load balancer and the steps to deploy it on the network is discussed in this chapter.

Load Balancer

A load balancer is a device that acts as a reverse proxy and distributes network or application traffic across many servers. Load balancers are used to increase the capacity (concurrent users) and the reliability of applications. They improve the overall performance of applications by decreasing the burden on servers associated, by managing and maintaining application and network sessions as well as by performing application-specific tasks.

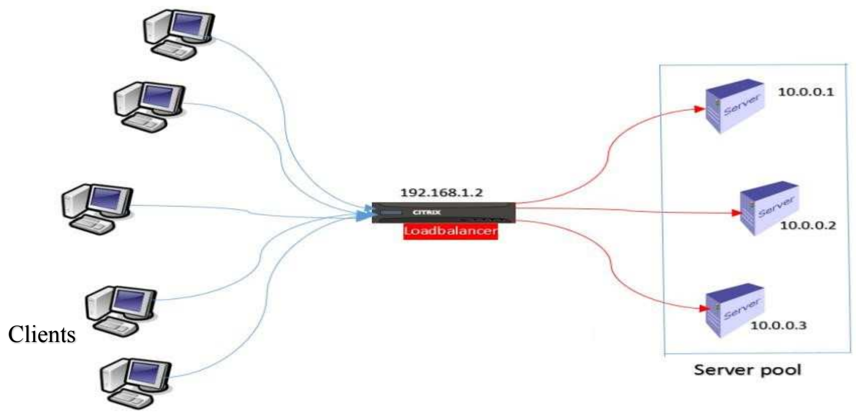


FIGURE 6. Load balancer [7].

Figure 6 shows that the communication between the clients and the servers is achieved through a load balancer (router), which has a unique IP address.

Load Balancing

Load Balancing is the process of distributing as well as redirecting traffic. The load balancing is highly reliable to use. Load balancing is accomplished not singularly but by bundle or convention, however, also by putting in a specific passageway. The load is out from 3 perspectives: system, purchaser, and server. In large-scale environment, the cloud data centers and end users are geographically distributed across the globe [7]. The biggest challenge for cloud data centers is how to handle and serve the millions of requests that are arriving very frequently from end users efficiently and correctly. In these cases, load balancing is required to distribute the dynamic workload evenly across all the nodes (servers). Therefore, load balancing can be used as a proxy server.

Load balancing helps to achieve a high user satisfaction and resource utilization ratio by ensuring an efficient and fair allocation of every computing resource. Proper load balancing aids in minimizing resource consumption, implementing fail-over, enabling scalability, avoiding bottlenecks, and over-provisioning.

Load Balancing on SDN

The load balancing with the utilization of SDN and the ODL Controller in a virtualized domain with Mininet is described and the outcomes are assessed. A “Central Controller” called “SDN Load Balancer” is proposed and it controls the load-balancing algorithm to balance the load among virtual machines in cloud datacenter. To enhance the parity, most servers receive the Round Robin-Domain Name Server (RR-DNS) arranging equation that maps the host name to a gaggle of science locations.

In Figure 7 the clients communicate with servers through a soft switch controlled by the user by using OpenFlow protocols. There are requirements that are needed for load balancing on SDN.

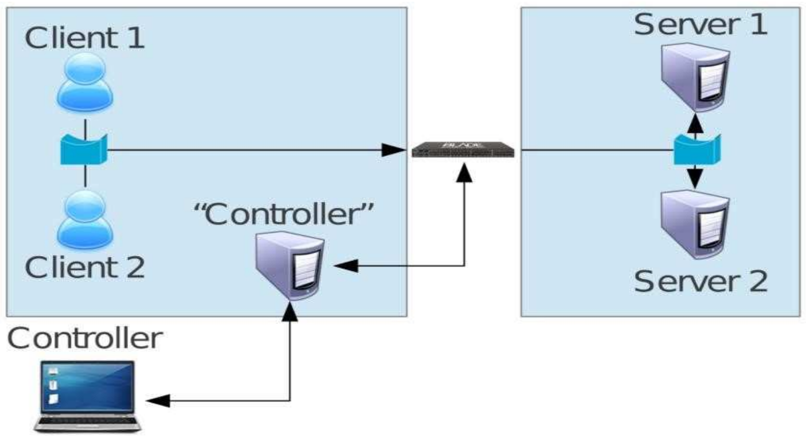


FIGURE 7. SDN load balancer [7].

The requirements are listed as follows:

- Load balancing on SDN requires the use of Mininet environment
- Load balancing on SDN requires the ODL controller

Mininet:

Mininet involves creating of topologies. The Mininet is a device to mimic the Software Defined Network that permits rapid proto-typing of a huge virtual foundation system with the utilization of standout PC. Mininet is a network emulation orchestration system that virtualizes a single system to look like a complete network, running the same kernel, system, and user code. Mininet empowers the user to make virtual models of versatile systems considering programming, for example OpenFlow utilizing primitive Virtualization Operating System. With these primitives, Mininet permits the user to make, associate and modify models for Software Defined Networks in a brisk manner.

Few features of the Mininet:

1. It gives a straightforward and modest path for testing systems for OpenFlow application improvement.
2. It permits that various scientists autonomously take a shot at the same system topology.

3. It permits the testing of a huge and complex topology, without even the need of a physical system.
4. It incorporates apparatus to troubleshoot and run tests over the system.
5. It underpins various topologies and incorporates an essential arrangement of topologies.
6. It provides straight forward Python APIs to make and test systems.
7. It, naturally, provides apparatus for creating learning systems in the territory.
8. Its interface empowers the utilization in examination and in classes for the common utilization of strategies and systems administration arrangements.

It is conceivable to perform with the Mininet troubleshooting tests and critical thinking, which can be a profit by having a complete trial system on a tablet or PC. The Mininet keeps running on virtual machine (VM Mininet) in VMware or Virtual Box for Windows working frameworks, Mac, and Linux with OpenFlow apparatus effectively introduced. Then again, the user can introduce VM Mininet on Ubuntu even though the engineers do not suggest.

Indeed, even with each of these qualities, the Mininet still does not show devoted execution and nature of a genuine system. This is because of assets that are multiplexed continuously by the part reproduction machine and the aggregate data transfer capacity constrained by limitations of the CPU (Central Processing Unit) and its own memory. Understanding the idea of POX and Mininet is conceivable to create important segments in POX and utilization in Mininet. To do as such, it is important to introduce them in their own virtual machine VM Mininet or on a machine which is facilitating the virtual machine, VM Mininet.

With created implementations, the scientist can better see how the structure of a SDN system can execute its own parts that meet the hunt pre-requisite. The codes are utilized as a part of the test system, Mininet; however, they could be connected in general.

OpenDay Light Controller:

To make SDN structural engineering, different segments are needed in its foundation. At present, there are a few SDN controllers such as OpenDay Light controller, POX controller. The ultimate goal for POX is to use it to create an archetypal, modern SDN controller. At first, a controller machine running a system-working framework, is required for instance ODL or POX controller. The primary segment of a Software Defined Networks is the SDN controller. The controller is the device that characterizes the way of the SDN worldview, where OpenDay Light (ODL) is used as a controller in this project. The OpenFlow protocol defines the open communications protocol that allows the SDN controller to work with the forwarding plane and make changes to the network. This provides the ability to adapt to the changing requirements and have greater control over the networks. ODL controller is the dependable segment for concentrating correspondence with every single programmable component of the system, providing a bound together perspective of the system.

Along these lines, forwarder components are also important that have isolated control arrangement with some programming interface, for this situation the OpenFlow. Besides, it is still obliged that a whole physical foundation connects these forwarder components and controller over a protected channel for the formation of a Software Defined Network.

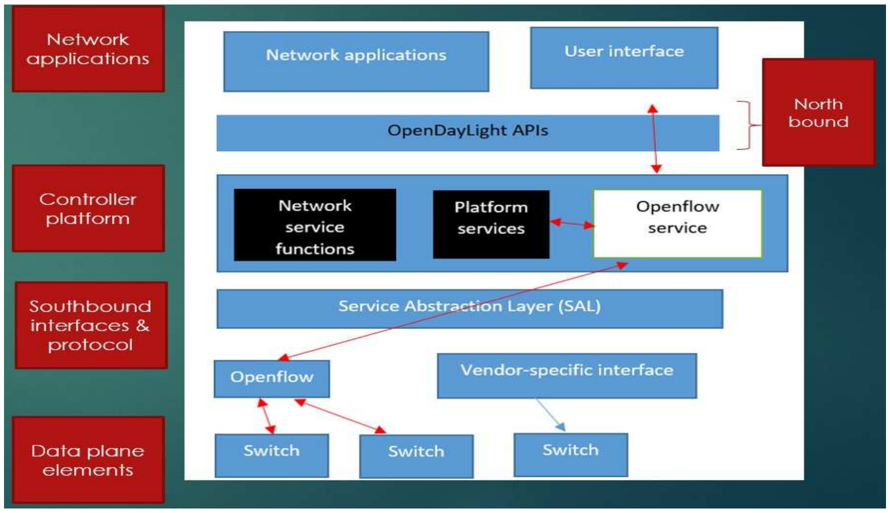


FIGURE 8. OpenDay Light controller [7].

Switches, data elements, network applications, platform services and the ODL APIs communicate using OpenFlow algorithms as shown in Figure 8.

Designing a SDN Load Balancer: A sample load balancer application that balances traffic to back-end servers based on the source address and source port on each incoming packet is utilized in this project. The simulation scenario consists of a single OpenFlow Switch (S) connected to four hosts (h1, h2, h3, h4) and to a controller ODL. This controller has two created components called 'Hub and Switch'. Four servers (S1, S2, S3, and S4) are created, and the load balancer is deployed as an application on the ODL controller.

The following steps are required to implement the designed Load Balancer functionality:

1. Start OpenDay Light controller: Make sure the sample load balancer and the sample modules are northbound, loaded by the controller (using ss on OSGi prompt).
2. Open terminal and input the following command: Sudo mn --custom ~/Mininet/custom/lb-4sw-5cli-3srv.py --topo mytopo: controller=remote, ip=10.10.55.59.
3. Then, input the following command: Pingall in Mininet.

4. Start browser and add default gateway in the ODL controller browser Firefox: http:// 10.10.55.59:8080, login: admin, password: admin, default gateway: 10.0.0.254/8.
5. Create load balancer pool VIP and attach the pool etc.
6. Install curl and run the load balancer design: Run ~ /loadbalancerdesign.sh 10.10.55.59 where ODL is running.
7. Generate xterms for hosts h1 to h4 from Mininet prompt to have xterms started on the machine. Open the terminals (xterm h1 S1 S2 S3 S4) on Mininet.
8. Resolve ARP for IP address of VIP; add static ARP entry for VIP's IP address in the host client h1: arp cache (arp -s 10.0.0.10 00:00:10:00:00:10) on h1 to h4 xterms.
9. On servers S1 to S4, start iperf server instances that listen on port 5550: (iperf -s -p 5550).
10. On h1, start iperf client that sends traffic to the virtual ip address: (iperf -c 10.0.0.10 -p 5550).
11. The user should see that iperf client connects to the iperf server running on server S1.
12. Once the previous test of iperf finishes, trigger the iperf client again and this time iperf client will connect to the iperf server running on server S2. Similarly, in the next iterations, iperf client will connect to the servers S3 and S4 in Round-Robin.

Results show that the proposed algorithm can achieve better load balancing in a largescale cloud computing environment when compared to previous load balancing algorithms.

CHAPTER 5

SERVICE FUNCTION CHAINING

In any network, there exists a path for the incoming packets or traffic. In SDN and NFV, the path for incoming traffic is achieved through Service Function Chaining (SFC). The operation of SFC, creating virtual instances, and setting rules for the firewall is explained in this chapter.

SFC provides the ability to define an ordered list of network services (e.g. firewalls, load balancers). In most networks, services are constructed as abstract sequences of Service Functions (SFs) that represent SFCs. These services are then "stitched" together in the network to create a service chain. This project provides the infrastructure (chaining logic, APIs) needed for ODL to provision a service chain in the network and an end-user application for defining such chains. At a high level, an SFC is an abstracted view of a service that specifies the set of required SFs as well as the order in which they must be executed.

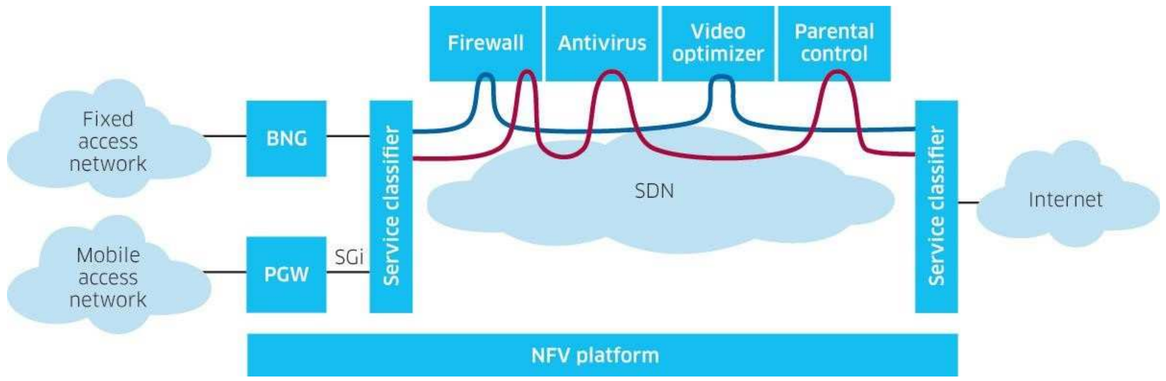


FIGURE 9. Service Function Chaining [8].

As in Figure 9, the incoming traffic is classified based on certain fields of the packets and the incoming traffic is appended with SFC tag, which is used to decide the path of the traffic.

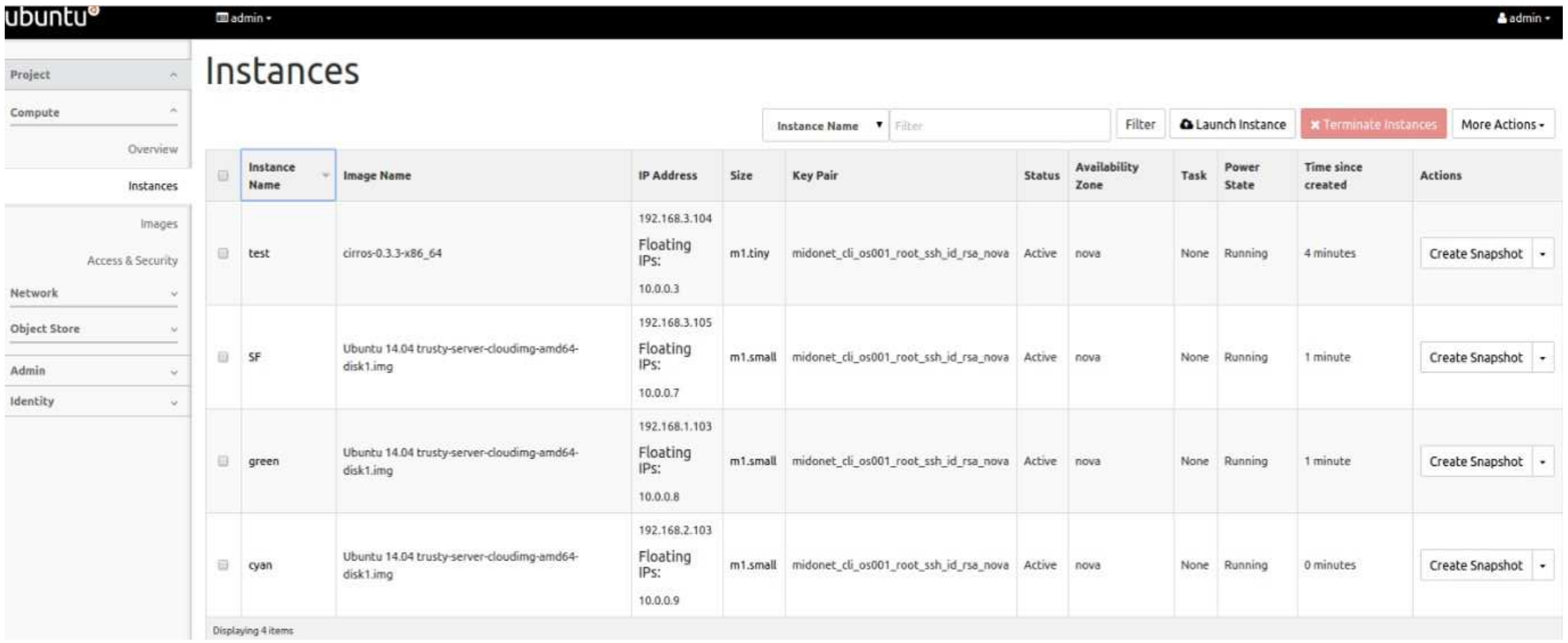
The traffic passes through different functions such as a firewall or an antivirus, and then go out by removing the appended SFC tag [8]. The SFC classifiers interface between the legacy traffic and the SFC traffic [8].

To enable SFC on top of OpenStack, first networks and routers are created in OpenStack using the dashboard ‘Horizon’. Three networks and two tenant routers are required and the second tenant router has no uplink configured. Four virtual instances namely Test, Green, Cyan and SF are created.

Figure 10 shows that the instances are created successfully. It also shows the details of the virtual instances such as IP addresses, running statuses, sizes, time they are created, power statuses, actions that can be performed and key pair names.

The public network and three virtual networks are connected through two tenant routers. The virtual instances SF and Test are created on network 3 with a virtual IP address 192.168.3.0/24. The virtual instance Cyan is created on network 2 with a virtual IP address 192.168.2.0/24. The virtual instance Green is created on network 1 with a virtual IP address 192.168.1.0/24. The network topology of these connections on cloud platform is shown in Figure 11.

To unblock all traffic, the user should enable the OpenStack security groups to allow all traffic by policy: allow any-any traffic, as shown in Figure 12. It provides the details of the ether types, IP protocols, port ranges, remote IP prefixes, and the directions of the created security rules. It also shows that the user can add, delete, and change/modify existing security rules as per the requirements.



Instances

Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
test	cirros-0.3.3-x86_64	192.168.3.104 Floating IPs: 10.0.0.3	m1.tiny	midonet_cli_os001_root_ssh_id_rsa_nova	Active	nova	None	Running	4 minutes	Create Snapshot
SF	Ubuntu 14.04 trusty-server-cloudimg-amd64-disk1.img	192.168.3.105 Floating IPs: 10.0.0.7	m1.small	midonet_cli_os001_root_ssh_id_rsa_nova	Active	nova	None	Running	1 minute	Create Snapshot
green	Ubuntu 14.04 trusty-server-cloudimg-amd64-disk1.img	192.168.1.103 Floating IPs: 10.0.0.8	m1.small	midonet_cli_os001_root_ssh_id_rsa_nova	Active	nova	None	Running	1 minute	Create Snapshot
cyan	Ubuntu 14.04 trusty-server-cloudimg-amd64-disk1.img	192.168.2.103 Floating IPs: 10.0.0.9	m1.small	midonet_cli_os001_root_ssh_id_rsa_nova	Active	nova	None	Running	0 minutes	Create Snapshot

Displaying 4 items

FIGURE 10. Instances.

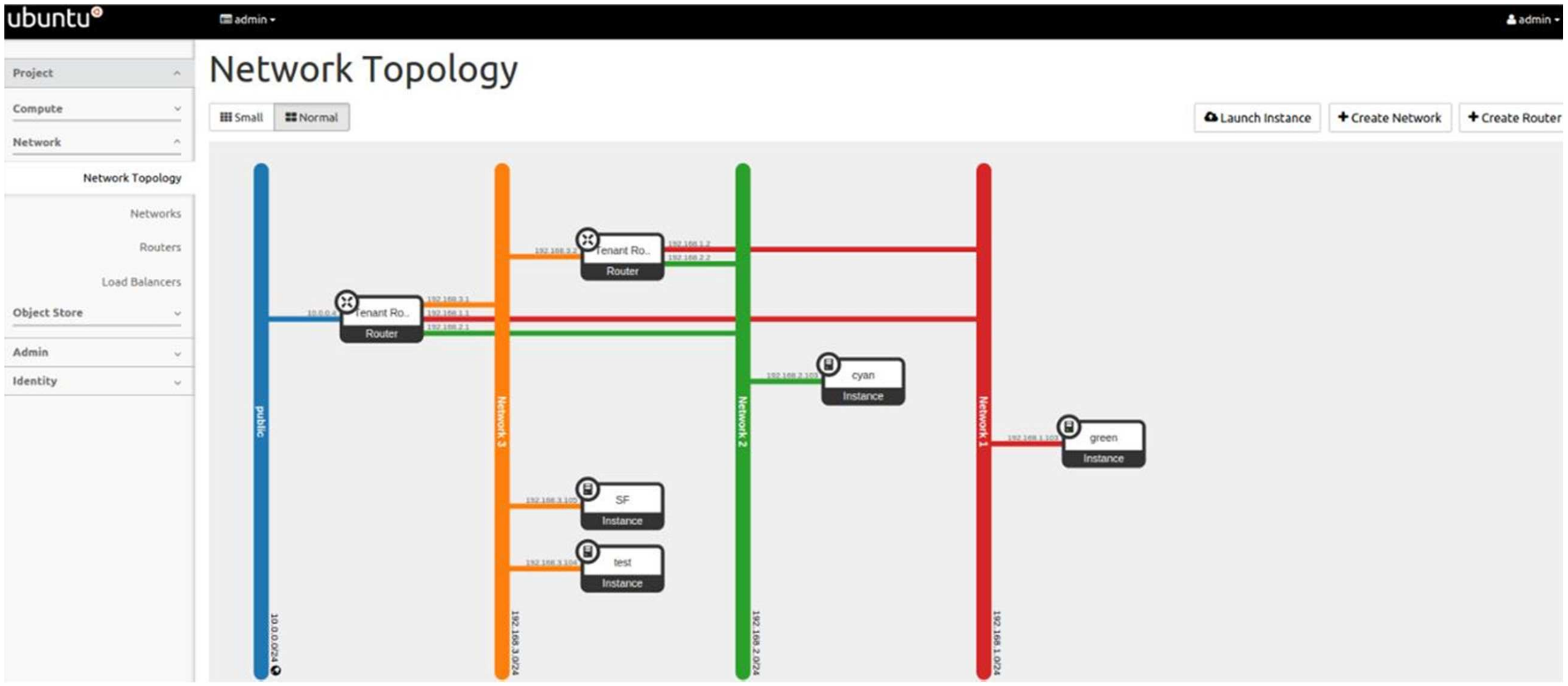
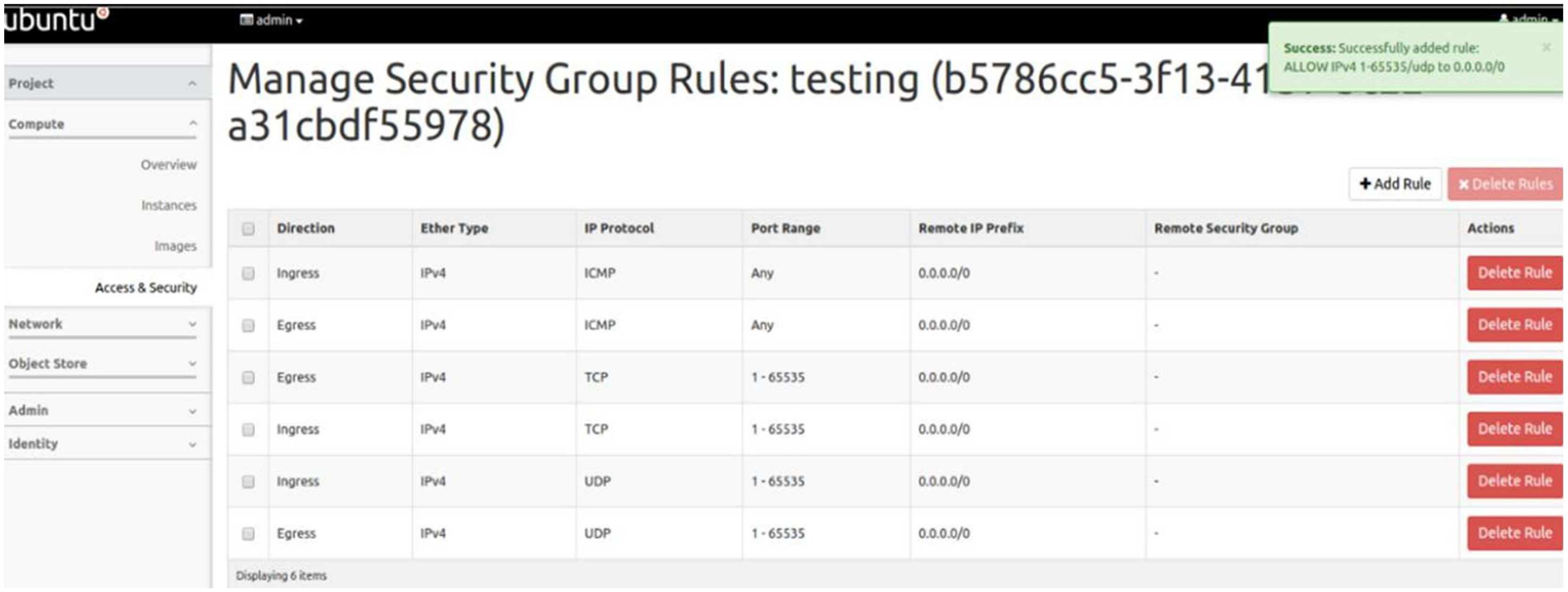


FIGURE 11. Network topology.



Manage Security Group Rules: testing (b5786cc5-3f13-41
a31cbdf55978)

Success: Successfully added rule: ALLOW IPv4 1-65535/udp to 0.0.0.0/0

Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Actions
Ingress	IPv4	ICMP	Any	0.0.0.0/0	-	Delete Rule
Egress	IPv4	ICMP	Any	0.0.0.0/0	-	Delete Rule
Egress	IPv4	TCP	1 - 65535	0.0.0.0/0	-	Delete Rule
Ingress	IPv4	TCP	1 - 65535	0.0.0.0/0	-	Delete Rule
Ingress	IPv4	UDP	1 - 65535	0.0.0.0/0	-	Delete Rule
Egress	IPv4	UDP	1 - 65535	0.0.0.0/0	-	Delete Rule

Displaying 6 items

FIGURE 12. Manage Security Group rules.

The management interface is connected to the bridge Branch Management (br- mgmt) and the other interfaces are connected to the bridge Branch Office (br O). SFC policy based flows for br O are configured as follows:

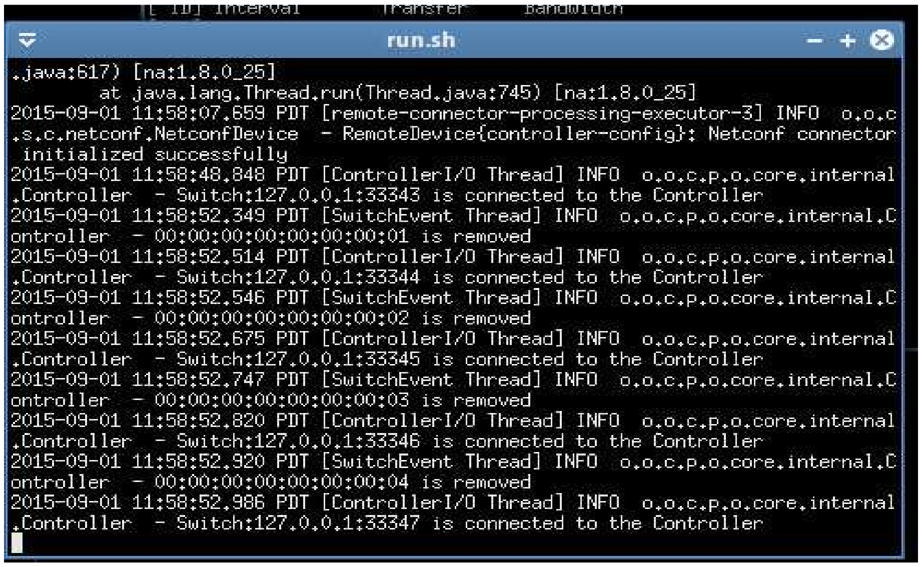
```
cookie=0x0, duration=0.121s, table=0, n_packets=0, n_bytes=0, idle_age=0, in_port=1
actions=output:80
cookie=0x0, duration=0.118s, table=0, n_packets=0, n_bytes=0, idle_age=0, in_port=80
actions=output:1
cookie=0x0, duration=0.114s, table=0, n_packets=0, n_bytes=0, idle_age=0, in_port=81
actions=output:82
cookie=0x0, duration=0.112s, table=0, n_packets=0, n_bytes=0, idle_age=0, in_port=82
actions=output:81
cookie=0x0, duration=0.108s, table=0, n_packets=0, n_bytes=0, idle_age=0, in_port=83
actions=output:84
cookie=0x0, duration=0.105s, table=0, n_packets=0, n_bytes=0, idle_age=0, in_port=84
actions=output:83
```

The flows are designed in a manner that the incoming packets from port 1 should give the output through port 80 and vice-versa. Similarly, incoming packets from ports 81, 83 should give output through ports 82, 84 respectively and vice-versa.

CHAPTER 6

RESULTS

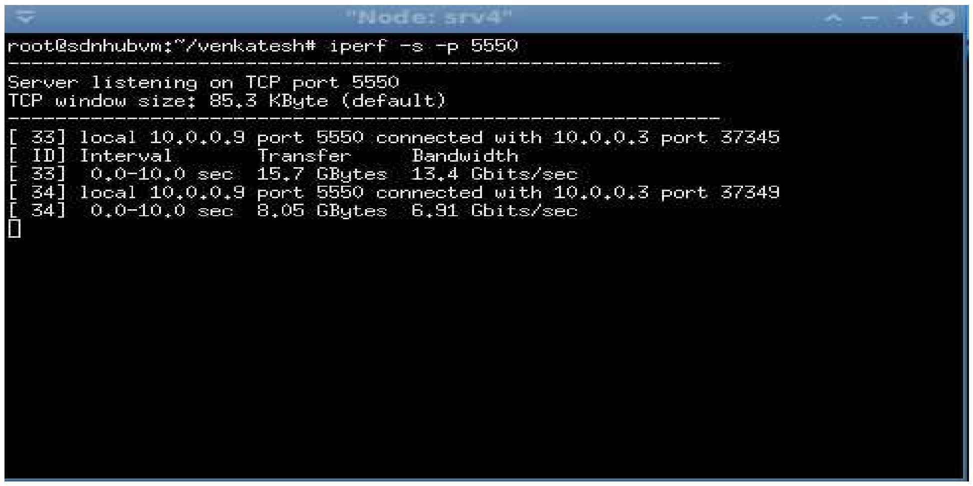
Load balancing implementation design and verification are performed on a single node server with all the components installed on the same machine with the help of virtualization. This project mainly requires SDN supported switches such as OVS or OVDK to act as a forwarding plane and a SDN controller as a control plane. Both switches and controllers are deployed as software demons running on any Linux machine. For this design, Ubuntu 14.04 virtual machine on top of Windows host is used.



```
.java:617) [na:1.8.0_25]
        at java.lang.Thread.run(Thread.java:745) [na:1.8.0_25]
2015-09-01 11:58:07.659 PDT [remote-connector-processing-executor-3] INFO  o.o.c
.s.c.netconf.NetconfDevice  - RemoteDevice{controller-config}: Netconf connector
 initialized successfully
2015-09-01 11:58:48.848 PDT [ControllerI/O Thread] INFO  o.o.c.p.o.core.internal
.Controller  - Switch:127.0.0.1:33343 is connected to the Controller
2015-09-01 11:58:52.349 PDT [SwitchEvent Thread] INFO  o.o.c.p.o.core.internal.C
ontroller  - 00:00:00:00:00:00:00:01 is removed
2015-09-01 11:58:52.514 PDT [ControllerI/O Thread] INFO  o.o.c.p.o.core.internal
.Controller  - Switch:127.0.0.1:33344 is connected to the Controller
2015-09-01 11:58:52.546 PDT [SwitchEvent Thread] INFO  o.o.c.p.o.core.internal.C
ontroller  - 00:00:00:00:00:00:00:02 is removed
2015-09-01 11:58:52.675 PDT [ControllerI/O Thread] INFO  o.o.c.p.o.core.internal
.Controller  - Switch:127.0.0.1:33345 is connected to the Controller
2015-09-01 11:58:52.747 PDT [SwitchEvent Thread] INFO  o.o.c.p.o.core.internal.C
ontroller  - 00:00:00:00:00:00:00:03 is removed
2015-09-01 11:58:52.820 PDT [ControllerI/O Thread] INFO  o.o.c.p.o.core.internal
.Controller  - Switch:127.0.0.1:33346 is connected to the Controller
2015-09-01 11:58:52.920 PDT [SwitchEvent Thread] INFO  o.o.c.p.o.core.internal.C
ontroller  - 00:00:00:00:00:00:00:04 is removed
2015-09-01 11:58:52.986 PDT [ControllerI/O Thread] INFO  o.o.c.p.o.core.internal
.Controller  - Switch:127.0.0.1:33347 is connected to the Controller
```

FIGURE 13. ODL controller running designed load balancer.

The designed load balancer switches which are deployed on the ODL controller and the OSGi window successfully, is shown in Figure 13.



```
root@sdnhubvm:~/venkatesh# iperf -s -p 5550
------------------------------------------------------------
Server listening on TCP port 5550
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 33] local 10.0.0.9 port 5550 connected with 10.0.0.3 port 37345
[ ID] Interval       Transfer     Bandwidth
[ 33]  0.0-10.0 sec  15.7 GBytes  13.4 Gbits/sec
[ 34] local 10.0.0.9 port 5550 connected with 10.0.0.3 port 37349
[ 34]  0.0-10.0 sec  8.05 GBytes  6.91 Gbits/sec
```

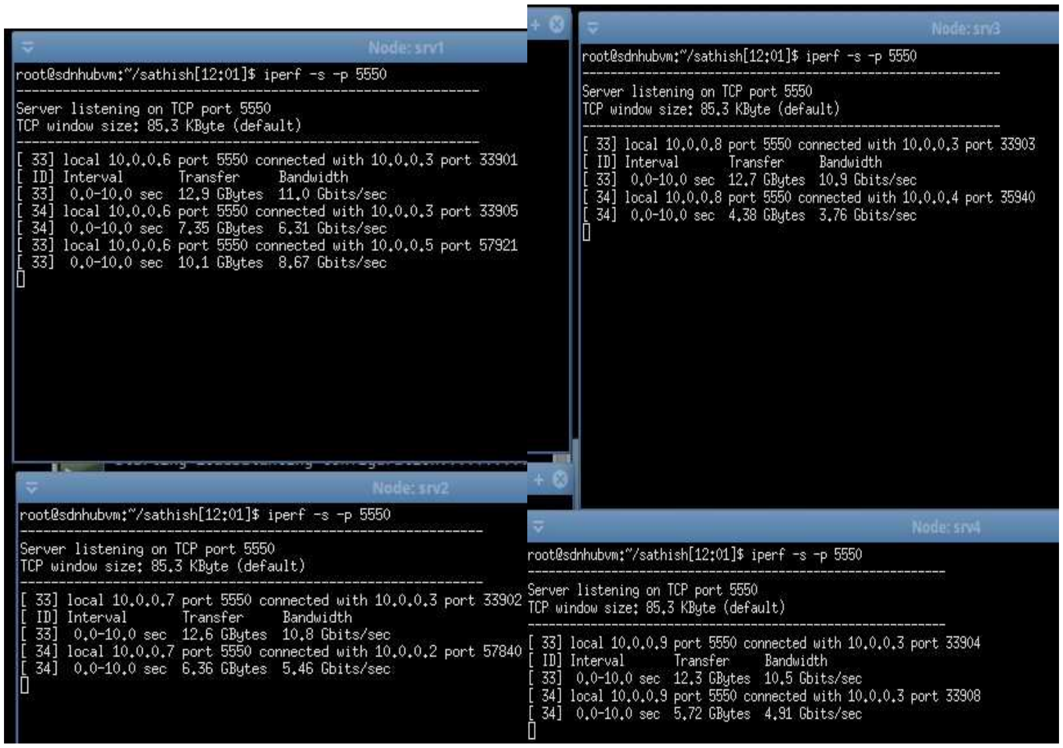
FIGURE 14. Iperf on servers.

Once the load balancing is started on the ODL, four virtual clients and four virtual servers are created. Now, the iperf for the analysis on the server is started and the same is shown in Figure 14. It can be observed that the local 10.0.0.9 port 5550 is connected with 10.0.0.3 port 37345 for the respective incoming request from the client.

After starting the iperf on the servers, the clients are connected in random order. Check that the connection establishments happen exactly based on the designed load balancer. For a Round-Robin load balancer, the connections are shown in Figure 15.

From Figure 15, it can be noticed that the first incoming packet from the client is sent to server node 1 through port 33901. Similarly, the second incoming packet from the same client is sent to server node 2 through port 33902 and the load balancing of the incoming packets follow this pattern verifying the RR-DNS algorithm.

Figure 15 provides the details of the size of the data transferred, time taken for the transfer, and the band width for each incoming request from the client. It also shows the interval time between each incoming request from the same client.



```
root@sdnhubvm:~/sathish[12:01]$ iperf -s -p 5550
------------------------------------------------------------
Server listening on TCP port 5550
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 33] local 10.0.0.6 port 5550 connected with 10.0.0.3 port 33901
[ ID] Interval       Transfer     Bandwidth
[ 33]  0.0-10.0 sec  12.9 GBytes  11.0 Gbits/sec
[ 34] local 10.0.0.6 port 5550 connected with 10.0.0.3 port 33905
[ 34]  0.0-10.0 sec  7.35 GBytes  6.31 Gbits/sec
[ 33] local 10.0.0.6 port 5550 connected with 10.0.0.5 port 57921
[ 33]  0.0-10.0 sec  10.1 GBytes  8.67 Gbits/sec
```

```
root@sdnhubvm:~/sathish[12:01]$ iperf -s -p 5550
------------------------------------------------------------
Server listening on TCP port 5550
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 33] local 10.0.0.8 port 5550 connected with 10.0.0.3 port 33903
[ ID] Interval       Transfer     Bandwidth
[ 33]  0.0-10.0 sec  12.7 GBytes  10.9 Gbits/sec
[ 34] local 10.0.0.8 port 5550 connected with 10.0.0.4 port 35940
[ 34]  0.0-10.0 sec  4.38 GBytes  3.76 Gbits/sec
```

```
root@sdnhubvm:~/sathish[12:01]$ iperf -s -p 5550
------------------------------------------------------------
Server listening on TCP port 5550
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 33] local 10.0.0.7 port 5550 connected with 10.0.0.3 port 33902
[ ID] Interval       Transfer     Bandwidth
[ 33]  0.0-10.0 sec  12.6 GBytes  10.8 Gbits/sec
[ 34] local 10.0.0.7 port 5550 connected with 10.0.0.2 port 57840
[ 34]  0.0-10.0 sec  6.36 GBytes  5.46 Gbits/sec
```

```
root@sdnhubvm:~/sathish[12:01]$ iperf -s -p 5550
------------------------------------------------------------
Server listening on TCP port 5550
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 33] local 10.0.0.9 port 5550 connected with 10.0.0.3 port 33904
[ ID] Interval       Transfer     Bandwidth
[ 33]  0.0-10.0 sec  12.3 GBytes  10.5 Gbits/sec
[ 34] local 10.0.0.9 port 5550 connected with 10.0.0.3 port 33908
[ 34]  0.0-10.0 sec  5.72 GBytes  4.91 Gbits/sec
```

FIGURE 15. Round Robin balancing connection between the clients and the servers.

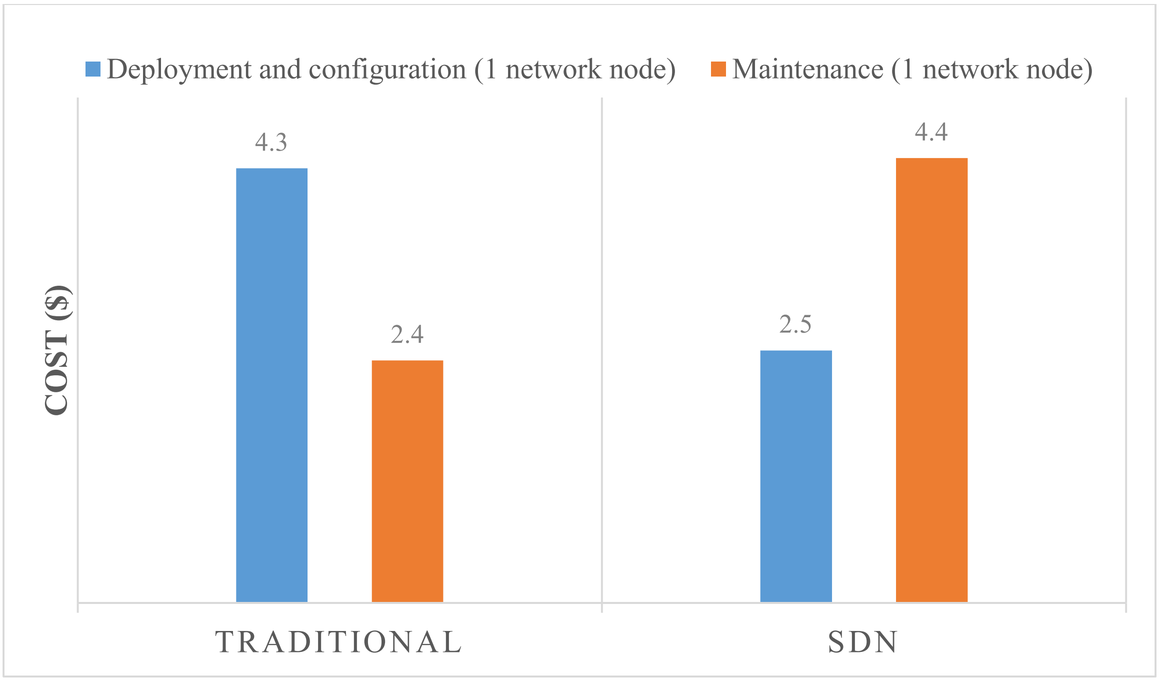


FIGURE 16. Cost comparison of traditional networks and SDN.

From Figure 16, it can be clearly seen that the cost of the networks when using SDN has decreased to nearly 50%. Thus, it can be assumed that if the traditional networks are replaced with SDN, the total cost of the project can be brought down.

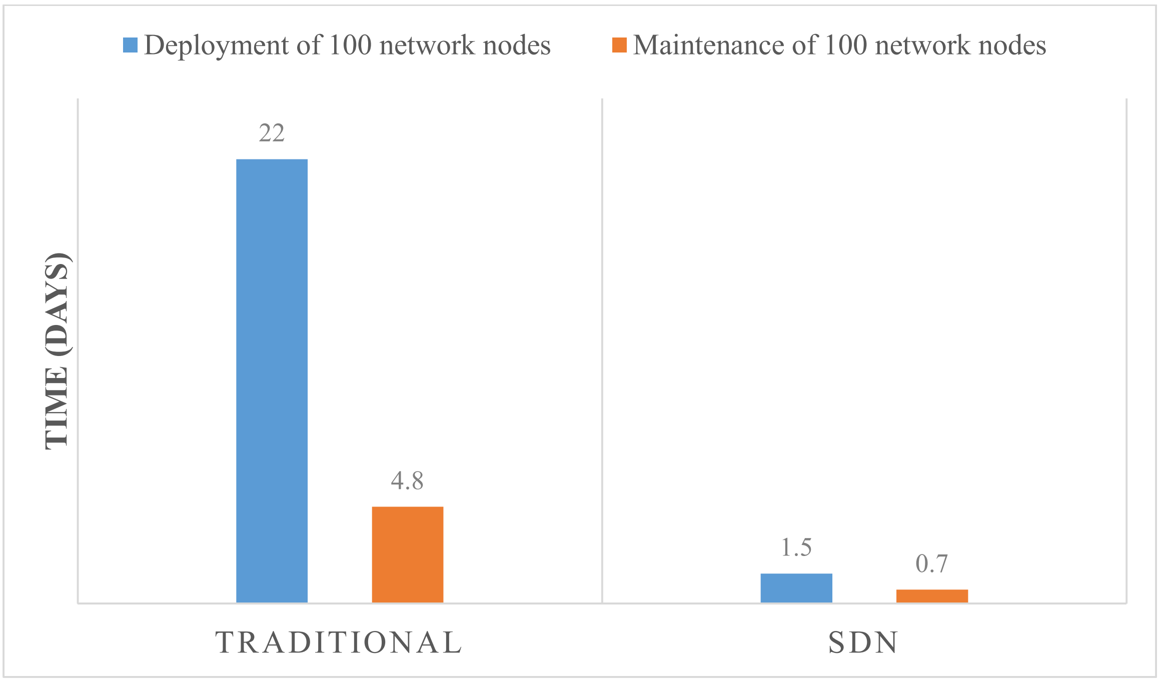


FIGURE 17. Time for deployment and maintenance of traditional networks and SDN.

Figure 17 shows that the load balancing using traditional networks is taking a huge time for deployment and maintenance when compared to Software Defined Networks. If SDN is used instead of traditional networks, a lot of time can be saved (which is elapsed in deployment and maintenance) whenever there is an issue with the network.

A traditional system and an SDN system with equally configured underlying hardware are utilized. A normal GNS3 switch is used for allowing traffic from both the systems. The image of the switching performance is emulated to obtain the network throughput.

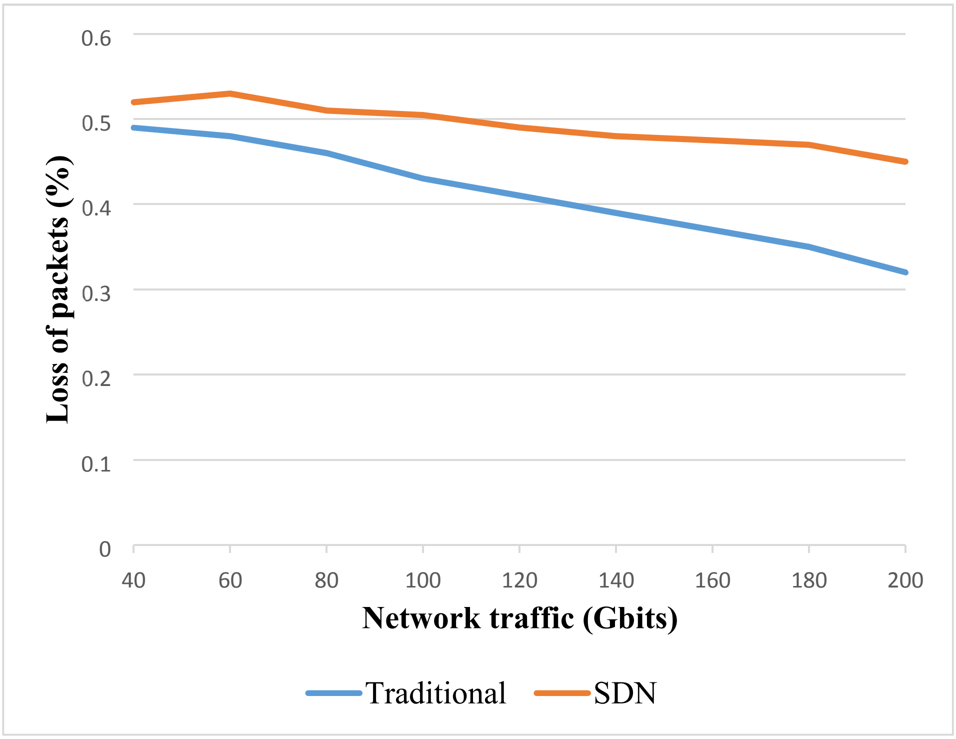
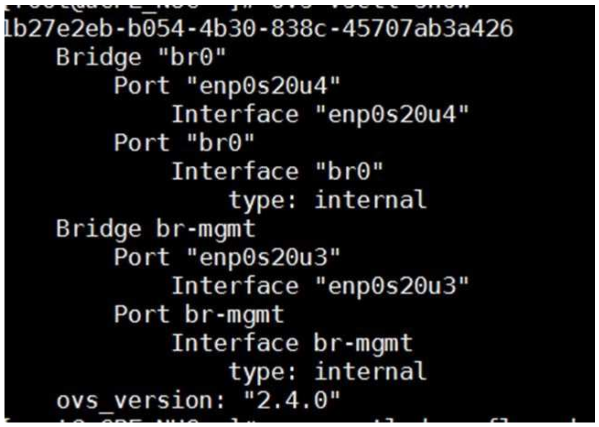


FIGURE 18. Network throughput.

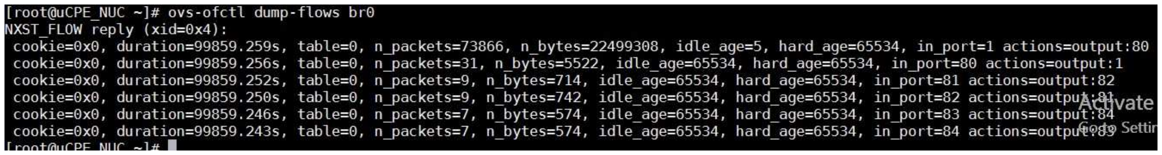
In Figure 18, network throughput increased when the load balancing is performed using SDN and therefore, the usage of SDN will be more advantageous when compared to the traditional load balancers.



```
1b27e2eb-b054-4b30-838c-45707ab3a426
    Bridge "br0"
        Port "enp0s20u4"
            Interface "enp0s20u4"
        Port "br0"
            Interface "br0"
                type: internal
    Bridge br-mgmt
        Port "enp0s20u3"
            Interface "enp0s20u3"
        Port br-mgmt
            Interface br-mgmt
                type: internal
    ovs_version: "2.4.0"
```

FIGURE. 19. Bridge connections of branch office and branch management interfaces.

Figure 19 shows that the branch office interface is connected to the Bridge br0 and management interface is connected to the Bridge br-mgmt successfully.



```
[root@uCPE_NUC ~]# ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=99859.259s, table=0, n_packets=73866, n_bytes=22499308, idle_age=5, hard_age=65534, in_port=1 actions=output:80
 cookie=0x0, duration=99859.256s, table=0, n_packets=31, n_bytes=5522, idle_age=65534, hard_age=65534, in_port=80 actions=output:1
 cookie=0x0, duration=99859.252s, table=0, n_packets=9, n_bytes=714, idle_age=65534, hard_age=65534, in_port=81 actions=output:82
 cookie=0x0, duration=99859.250s, table=0, n_packets=9, n_bytes=742, idle_age=65534, hard_age=65534, in_port=82 actions=output:81
 cookie=0x0, duration=99859.246s, table=0, n_packets=7, n_bytes=574, idle_age=65534, hard_age=65534, in_port=83 actions=output:84
 cookie=0x0, duration=99859.243s, table=0, n_packets=7, n_bytes=574, idle_age=65534, hard_age=65534, in_port=84 actions=output:83
[root@uCPE_NUC ~]#
```

FIGURE. 20. Output for SFC based flows.

In Figure 20, it can be noticed that the output is sent from the ports 80, 1, 82, …… as per the given input. This figure also provides the number of packets processed, time taken to process the packets and the idle time of the system during the process.

CHAPTER 7

CONCLUSION

Traffic Engineering is a study of measuring and managing network traffic, designing routing mechanisms to provide a path for the incoming network traffic while efficiently using the available network resources and better meet the quality of service.

This project demonstrates that using the “load balancer” as a use-case, how different network applications can be designed using “Software Defined Networking” techniques, which are increasingly popular. In addition, it is shown that these applications can be deployed at a faster pace when compared to traditional networking techniques and that it is simple to manage these applications using SDN. Also, the functional performance of the load balancer is better when implement using the SDN.

The goal of the project, which is to demystify designing of an efficient load balancer using SDN, is achieved.

REFERENCES

REFERENCES

[1] Q.Y. Zuo, M. Chen, G.S. Zhao, C.Y. Xing, G.M. Zhang, and P.C. Jiang, "OpenFlow- based SDN technologies," *Ruanjian Xuebao/Journal of Software*, vol. 24, pp. 1078-1097, 2013.

[2] H. Kim, A. Voellmy, S. Burnett, N. Feamster, and R. Clark, "Lithium: Event- driven network control," Georgia Institute of Technology, Atlanta, Tech. Rep., May 2013.

[3] M. Handley, O. Hodson, and E. Kohler, "Xorp: An Open platform for network research," paper presented at ACM SIGCOMM Hot Topics in Networking, 2002.

[4] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashock, "The Click modular router," *ACM Transactions on Computer Systems,* vol. 18, no.3, pp. 263-297, Aug. 2007.

[5] J. Turner, P. Crowley, J. Dehart, A. Freestone, B. Heller, F. Kuhms, S. Kumar, J. Lockwood Lu, M. Wilson, C. Wiseman, and D. Zar, "Supercharging Planetlab high performance, multi-application, overlay network platform," paper presented at ACM SIGCOMM, Aug 2007, Kyoto, Japan.

[6] MediaWiki and WordPress. (2011). The OpenFlow Switch Specification. [Online]. Available: http://OpenFlowSwitch.org.

[7] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," paper presented at ACM SIGCOMM, August 2007, Kyoto, Japan.

[8] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105-110, July 2008.