



**SYMBIOSIS**  
**INSTITUTE OF TECHNOLOGY, NAGPUR**

# Managing Network using NFV and SDN

## CAPSTONE PROJECT REPORT

Submitted in partial fulfillment of the requirements for the Capstone  
Project of B.Tech CSE

### Submitted by:

Swarali Limaye PRN: (22070521144)

Praveg Chitke PRN: (22070521148)

Saharsh Vaidya PRN: (22070521158)

### Under the Guidance of:

Dr. Pradnya Borkar

B.Tech Computer Science and Engineering  
Semester VI  
Session 2024–2025

# CERTIFICATE

This is to certify that the Capstone Project work titled “**Managing Network Using NFV and SDN**” that is being submitted by **Swarali Limaye, 22070521144** **Sa-harsh Vaidya, 22070521158** **Praveg Chikte, 22070521148** is in partial fulfillment of the requirements for the Capstone Project is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma, and the same is certified.

Name of PBL Guide & Signature

Verified by:  
Dr. Parul Dubey  
Capstone Project Coordinator

**The Report is satisfactory/unsatisfactory**

Approved by  
**Prof. (Dr.) Nitin Rakesh**  
Director, SIT Nagpur

# Abstract

The project expands existing ideas about uniting **SDN (Software Defined Networking)** functionalities with **NFV (Network Function Virtualization)** elements. **SDN** functions through **Data and Control plane separation** while **NFV** allows network appliances to be replaced by server-based software. The **NFV project** functions as an approach to modernize traditional network management through these techniques.

**OpenFlow protocol** and the **OpenStack framework** serve as implementation components for this project because they provide flexible management of **SDN** and **NFV** operations. The project demonstrates the use of **OpenStack**, **OpenDaylight**, and **Mininet** alongside existing features and additional enhancements built onto **OpenStack** and **OpenDaylight** to fulfill the given requirements.

This project creates a depiction of different realistic situations by using **OpenStack instances** for **Webserver** and **Fileserver** functions along with other service infrastructure implementations. **Mininet** serves as the virtualization platform for **ISP network topology**, while **OpenDaylight** controls packet routing through **Mininet** switches.

The integration of **SDN with NFV** provides benefits that set this project as a new approach for meeting modern challenges while being **reasonably affordable for academic research purposes**.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Objectives . . . . .	3
1.2	Existing System . . . . .	3
1.3	Proposed System . . . . .	4
1.4	Organization of the Report . . . . .	4
<b>2</b>	<b>Literature Review</b>	<b>5</b>
<b>3</b>	<b>System Design and Architecture</b>	<b>6</b>
3.1	System Overview . . . . .	6
3.2	Architecture Design . . . . .	6
3.3	Technology Stack . . . . .	7
3.4	Data Flow Theory . . . . .	8
<b>4</b>	<b>Implementation</b>	<b>9</b>
4.1	Install Mininet . . . . .	9
4.2	Install XTerm (for terminal windows): . . . . .	9
4.3	Install Open vSwitch (OVS) . . . . .	9
4.4	Install POX Controller . . . . .	9
4.5	Run the Project in Mininet . . . . .	10
4.6	Run the POX Controller . . . . .	10
4.7	Test Network Connectivity in Mininet . . . . .	10
4.8	Observe POX Controller Behavior . . . . .	10
4.9	Open XTerm for Hosts (Optional but Helpful) . . . . .	11
4.10	Run NFV Scripts (If Applicable) . . . . .	11
4.11	Analyze or Visualize Output . . . . .	11
4.12	Stop Everything Gracefully . . . . .	12
<b>5</b>	<b>Results, Overview &amp; Analysis</b>	<b>13</b>
5.1	Results . . . . .	13
5.2	Overview . . . . .	16
5.3	Analysis . . . . .	18
<b>6</b>	<b>Conclusion and Future Works</b>	<b>19</b>
6.1	Conclusion . . . . .	19
6.2	Future Works . . . . .	19
<b>7</b>	<b>References</b>	<b>21</b>

# Chapter 1

## Introduction

Modern times exhibit rapid development of Internet and communication technologies which produces substantial data traffic expansion. Network resource optimization requires new management solutions because the rapid growth keeps posing obstacles. The approach of traditional network management is losing value because its static nature and hardware focus. Network management requirements for dynamic scalable programable solutions have brought Software Defined Networking (SDN) and Network Function Virtualization (NFV) into being. SDN implements dynamic network management through centralized control platforms which use programmable interfaces but NFV separates network functions from hardware devices to run them as software.

The project implements SDN and NFV integration to optimize network administration through cloud-based VNF deployment. The deployment uses Mininet to create SDN topology simulations alongside Kubernetes for container management which provides flexible and scalable VNF management capabilities.

### 1.1 Objectives

We will create and implement an expandable network management system through the combination of SDN and NFV technologies. The implementation of VNFs through containers requires Kubernetes for orchestration. The performance evaluation and scalability assessment of the proposed architectural design will be conducted. The solution aims to deliver an economical flexible approach over conventional network management systems.

### 1.2 Existing System

The conventional network management system heavily depends on hardware-based devices together with manual configuration procedures. Every network device functions by itself while a strong link between control plane and data plane restricts both flexibility and programmability. Physical intervention is necessary to add new services or modify existing ones because this system lacks scalability. Network deployment of firewalls as well as load balancers and other functions requires dedicated hardware which leads to both higher costs and suboptimal resource allocation.

## 1.3 Proposed System

**Proposed System** The system implements contemporary network management through the combination of SDN and NFV technologies. The SDN controller operates the network flow rules dynamically while Kubernetes deploys virtualized network functions through containers. SDN and NFV technologies work together in this network architecture to provide unified control and automatic resources provisioning and optimized resource allocation. Network agility improves as the Virtual Network Functions (VNFs) can be dynamically scaled according to demand needs. Testing of the system involves Mininet for network emulation and Docker/Kubernetes for virtual network function deployment.

## 1.4 Organization of the Report

The initial chapter presents an introduction to the project that includes objectives and existing and proposed system information and report organization structure.

The second chapter presents a literature review which incorporates existing technologies and research in SDN and NFV domains.

The project implementation methodology with system analysis appears in Chapter 3.

The system design and implementation details are presented in Chapter 4 while discussing the architecture and tool utilization and configuration aspects.

The results and analytical findings from the proposed system assessment appear in Chapter 5.

The report ends with chapter 6 while proposing potential improvements for future development.

## Chapter 2

# Literature Review

Academic institutions together with industrial entities show significant attention toward the SDN and NFV integration because of its potential benefits for network flexibility and scalability and cost-efficiency. Many research works along with projects have focused on this domain.

SDN introduces a new control plane and data plane separation through which network administration becomes centralized. The control plane separation from data plane through this design allows organizations to create dynamic configurations while managing resources efficiently and programming networks.

Network Function Virtualization (NFV) functions along with Software Defined Networking (SDN) by virtualizing firewall systems along with infiltration security systems and forwarding services that normally function on dedicated appliances. Through NFV general-purpose servers can execute network functions which results in cost reductions and enhanced scalability benefits.

Multiple research works have developed architectures which unite SDN with NFV to create flexible programmable network systems. The deployment of VNFs through these architectures employs OpenDaylight and Ryu SDN controllers with OpenStack and Kubernetes virtualization platforms.

The implementation of Docker as a container technology and the Kubernetes orchestration system has been investigated for the deployment of VNFs that have both lightweight design and scalability features. Fast deployment combined with efficient resource allocation comes together with easier management through these technologies.

Research teams typically employ Mininet emulation tools to test SDN topologies because these tools let them verify network configurations and controller behavior within controlled environments.

## Chapter 3

# System Design and Architecture

### 3.1 System Overview

System overview Academic institutions and industrial entities display substantial interest in SDN and NFV integration because network flexibility and scalability alongside cost-efficiency benefits are possible. Multiple research studies and projects have concentrated on this field.

Centralized network administration emerges from SDN because it implements separate control and data planes. Network administration through this design achieves control plane separation from data plane which enables organizations to both manage resources effectively and program networks and create dynamic configurations.

NFV together with SDN provides virtualization capabilities to firewall systems and infiltration security systems and forwarding services that typically need dedicated appliances. NFV enables general-purpose servers to run network functions leading to decreased costs together with better scalability.

The combination of SDN with NFV has led multiple research projects to design flexible programmable network systems. The deployment of VNFs through these architectures employs OpenDaylight and Ryu SDN controllers with OpenStack and Kubernetes virtualization platforms.

The deployment of VNFs with lightweight design and scalability features uses Docker containers managed by the Kubernetes orchestration system. These technologies provide fast deployment and efficient resource allocation and easier management as separate capabilities.

Research groups use Mininet emulation tools to test SDN topologies since these tools enable them to validate network configurations and controller behaviors through controlled testing environments.

### 3.2 Architecture Design

The architecture consists of a layered design where SDN handles control and data flow while NFV provides service abstraction through virtualized functions.



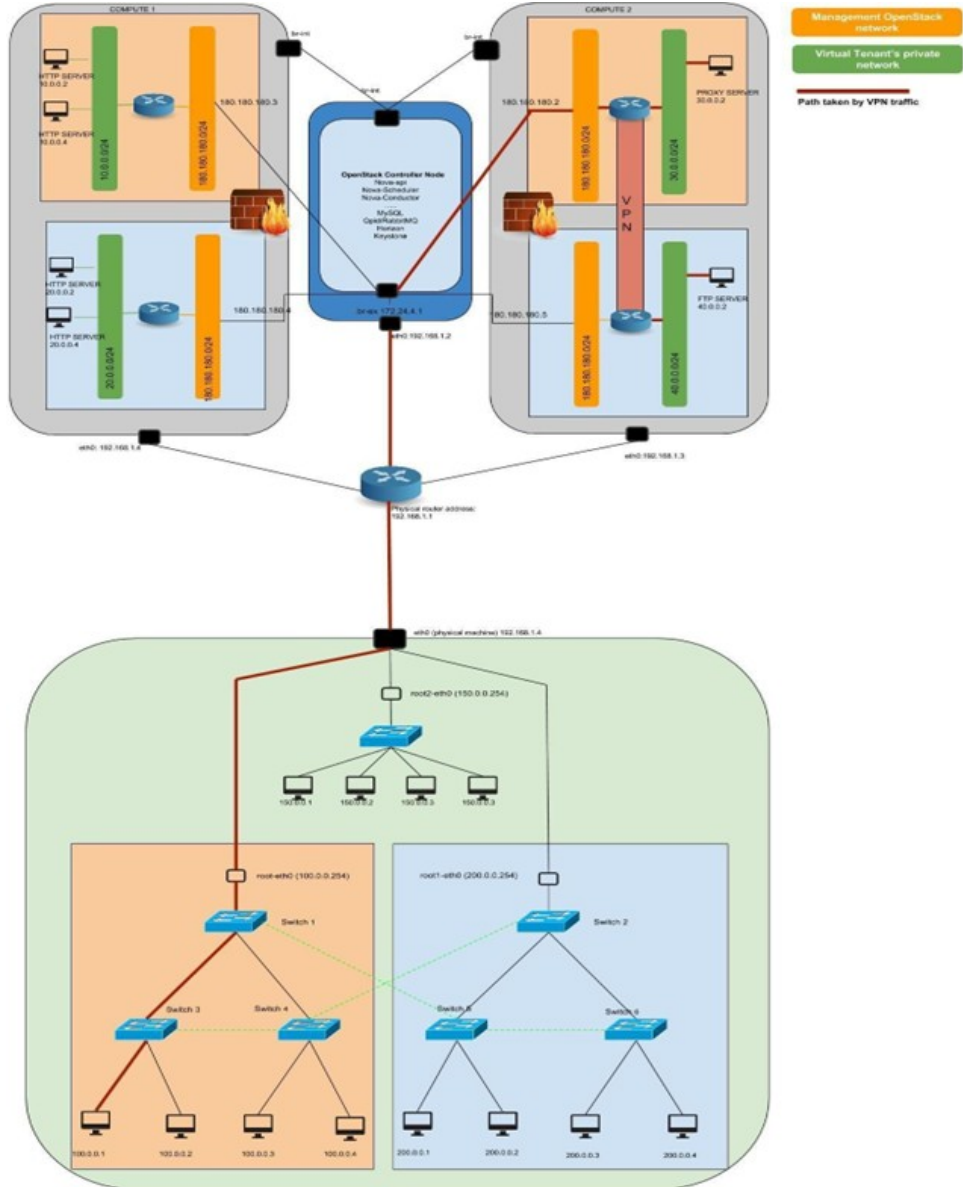


Figure 3.1: System Architecture Diagram

### 3.3 Technology Stack

The development of the proposed system depends on the following implementation technologies:

- **Mininet:** Enables users to create virtual networks with hosts, switches, controllers, and links on a single machine. It helps in testing SDN behavior in various network scenarios.
- **Ryu Controller:** A Python-based, component-driven SDN controller with OpenFlow support, allowing dynamic switch management.
- **Docker:** A containerization tool that creates lightweight containers for VNFs, ensuring isolated and repeatable deployment environments.

- **Kubernetes:** An open-source orchestration tool that automates deployment, scaling, and management of VNFs in containers.
- **Linux (Ubuntu):** The base operating system supporting open-source tools and offering easy configuration for system components.
- **Python:** Used for implementing Ryu controller logic, scripting automation, and monitoring.
- **OpenFlow:** A communication protocol enabling the SDN controller to manage switches and define packet routing paths.

## 3.4 Data Flow Theory

1. The system starts with users or simulated hosts in Mininet generating network traffic.
2. OpenFlow-enabled switches in the Mininet topology receive this traffic as part of the packet forwarding process.
3. Unrecognized packets are sent to the SDN controller (Ryu) for instructions through the Flow Request mechanism.
4. The SDN controller analyzes incoming requests, checks flow policies, and determines actions based on the network state.
5. The controller installs flow rules in switches to allow future packets of the same flow to be handled directly.
6. Flows requiring inspection or modification are redirected to Virtual Network Functions (VNFs) like firewalls or IDS, running in Docker containers.
7. Kubernetes manages the lifecycle of VNFs, balances traffic loads, and scales the VNFs to handle traffic efficiently.
8. The processed traffic is either returned to the originating host or forwarded to external networks via Root Gateway Nodes.
9. Python-based monitoring scripts collect data on flow behavior and VNF performance for analysis and optimization.

## Chapter 4

# Implementation

### 4.1 Install Mininet

- Mininet is a tool used to simulate SDN networks.
- Use this command to install it:

Terminal Command

```
sudo apt install mininet
```

### 4.2 Install XTerm (for terminal windows):

- It helps open multiple terminal windows for different devices.

Terminal Command

```
sudo apt install xterm
```

### 4.3 Install Open vSwitch (OVS)

- It is a virtual switch used in SDN.

Terminal Command

```
sudo apt install openvswitch-switch
```

### 4.4 Install POX Controller

- POX is the SDN controller used in this project.
- Go to the directory where you want POX installed.
- Clone the POX repository:

Terminal Command

```
git clone https://github.com/noxrepo/pox.git
```

## 4.5 Run the Project in Mininet

- Navigate to your project directory where the Mininet script is located.
- Run the topology file using the following command:

Terminal Command

```
sudo python3 <your_topology_file>.py
```

- Replace <your\_topology\_file>.py with the actual file name of your topology script.

## 4.6 Run the POX Controller

- Open a new terminal.
- Go to the POX directory.
- Run the controller using the following commands:

Terminal Commands

```
cd pox
./pox.py forwarding.l2_learning
```

## 4.7 Test Network Connectivity in Mininet

- After running your topology, you'll be inside the Mininet CLI.
- Use the following command to test if all hosts can communicate:

Mininet CLI Command

```
pingall
```

- This command sends ping messages from every host to every other host.
- If everything is working correctly, you should see **0% packet loss**.

## 4.8 Observe POX Controller Behavior

- Look at the terminal where you ran the POX controller using: `./pox.py forwarding.l2_learning`
- You'll see log messages showing how the controller is learning the network.
- These logs include information about:
  - MAC addresses
  - Switch connections
  - Flow installations

## 4.9 Open XTerm for Hosts (Optional but Helpful)

- In the Mininet CLI, you can open terminal windows for specific hosts using the command:

Mininet CLI Command

```
xterm h1 h2
```

- Replace `h1`, `h2` with actual host names from your topology.
- These terminals simulate separate computers.
- You can use them to run commands like:
  - `ping`
  - `iperf`
  - Any custom script for testing

## 4.10 Run NFV Scripts (If Applicable)

- If your project includes Network Function Virtualization (e.g., firewalls, IDS, load balancers), check if any scripts are provided to run these services.
- These are usually Python files that launch VNFs (Virtual Network Functions) on hosts.
- Example command to run a firewall VNF:

VNF Script Example

```
python3 nfv_firewall.py
```

## 4.11 Analyze or Visualize Output

- Some projects generate log files or performance data such as:
  - Throughput
  - Delay
  - Packet loss
- Check these output files in your project directory for analysis.
- You can also use visualization or monitoring tools such as:
  - **Wireshark** – for packet-level inspection
  - **iftop** – for real-time bandwidth monitoring
  - Python plotting libraries (e.g., **matplotlib**) – for graph-based results

## 4.12 Stop Everything Gracefully

- Once you're done testing and observing the network:
  - In the Mininet CLI, type the following command:

Mininet Command

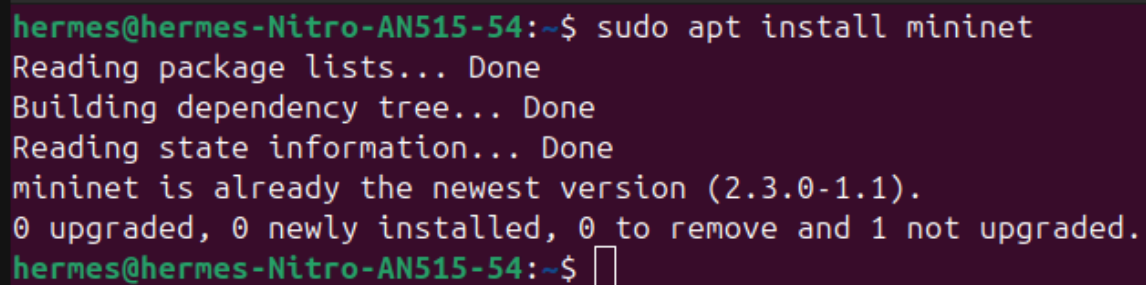
```
exit
```

- In the terminal running the POX controller, press **Ctrl + C** to stop it.

## Chapter 5

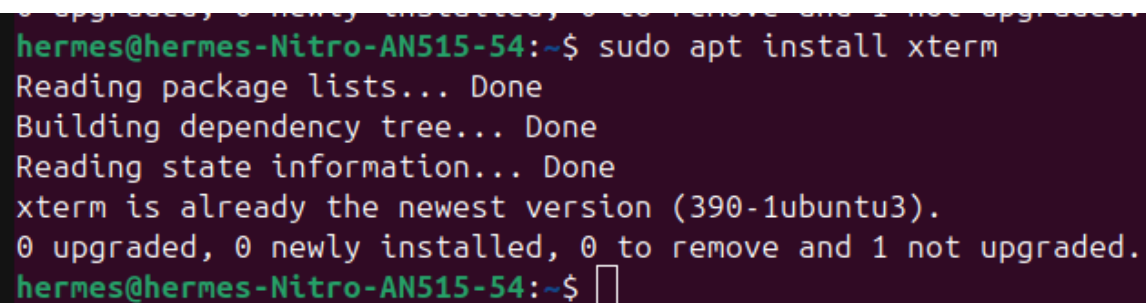
# Results, Overview & Analysis

### 5.1 Results

A terminal window with a dark purple background. The text is as follows:

```
hermes@hermes-Nitro-AN515-54:~$ sudo apt install mininet
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
mininet is already the newest version (2.3.0-1.1).
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
hermes@hermes-Nitro-AN515-54:~$
```

Figure 5.1: Mininet Installation

A terminal window with a dark purple background. The text is as follows:

```
hermes@hermes-Nitro-AN515-54:~$ sudo apt install xterm
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
xterm is already the newest version (390-1ubuntu3).
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
hermes@hermes-Nitro-AN515-54:~$
```

Figure 5.2: XTern Installation

```

hermes@hermes-Nitro-AN515-54:~$ sudo apt install openvswitch-switch
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openvswitch-switch is already the newest version (3.3.0-1ubuntu3.2).
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
hermes@hermes-Nitro-AN515-54:~$

```

Figure 5.3: Openswitch Installation

```

hermes@hermes-Nitro-AN515-54:~$ cd pox
./pox.py forwarding.l2_learning
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.12.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:openflow.of_01:[00-00-00-00-00-06 2] connected
INFO:openflow.of_01:[9a-7e-70-7c-69-4f 3] connected
INFO:openflow.of_01:[00-00-00-00-00-04 4] connected
INFO:openflow.of_01:[00-00-00-00-00-05 5] connected
INFO:openflow.of_01:[00-00-00-00-00-03 6] connected
INFO:openflow.of_01:[00-00-00-00-00-02 7] connected

```

Figure 5.4: Pox Installation

```

hermes@hermes-Nitro-AN515-54:~/Downloads/NETWORK-MANAGEMENT-USING-SDN-and-NFV-master/Minine
t$ sudo python3 FinalDemoTopo.py
*** Add switches
*** Add hosts
*** Add Host-Switch links
*** Add Switch-Switch links
*** Adding NAT and root nodes
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 root root1 root2
*** Adding switches:
s0 s1 s2 s3 s4 s5 s6
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s2) (h6, s2) (h7, s2) (h8, s2) (h9, s3) (h10, s3)
(h11, s3) (h12, s3) (root, s1) (root1, s2) (root2, s0) (s0, s1) (s0, s2) (s0, s3) (s1, s4)
(s2, s5) (s3, s6)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 root root1 root2
*** Starting controller
c0
*** Starting 7 switches
s0 s1 s2 s3 s4 s5 s6 ...
*** Configuring NAT and routes
*** Starting CLI:
mininet>

```

Figure 5.5: Pox Installation



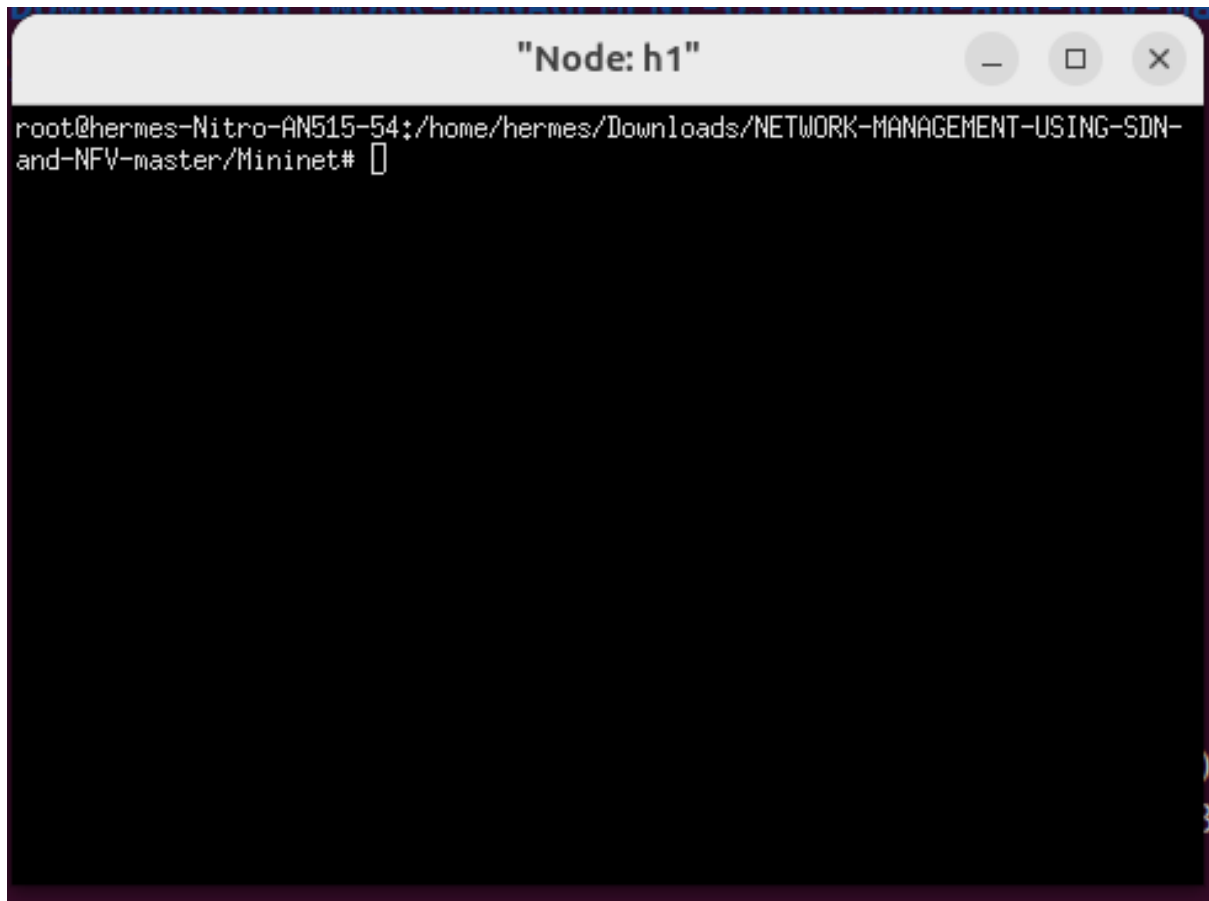


Figure 5.6: h1 Node interface

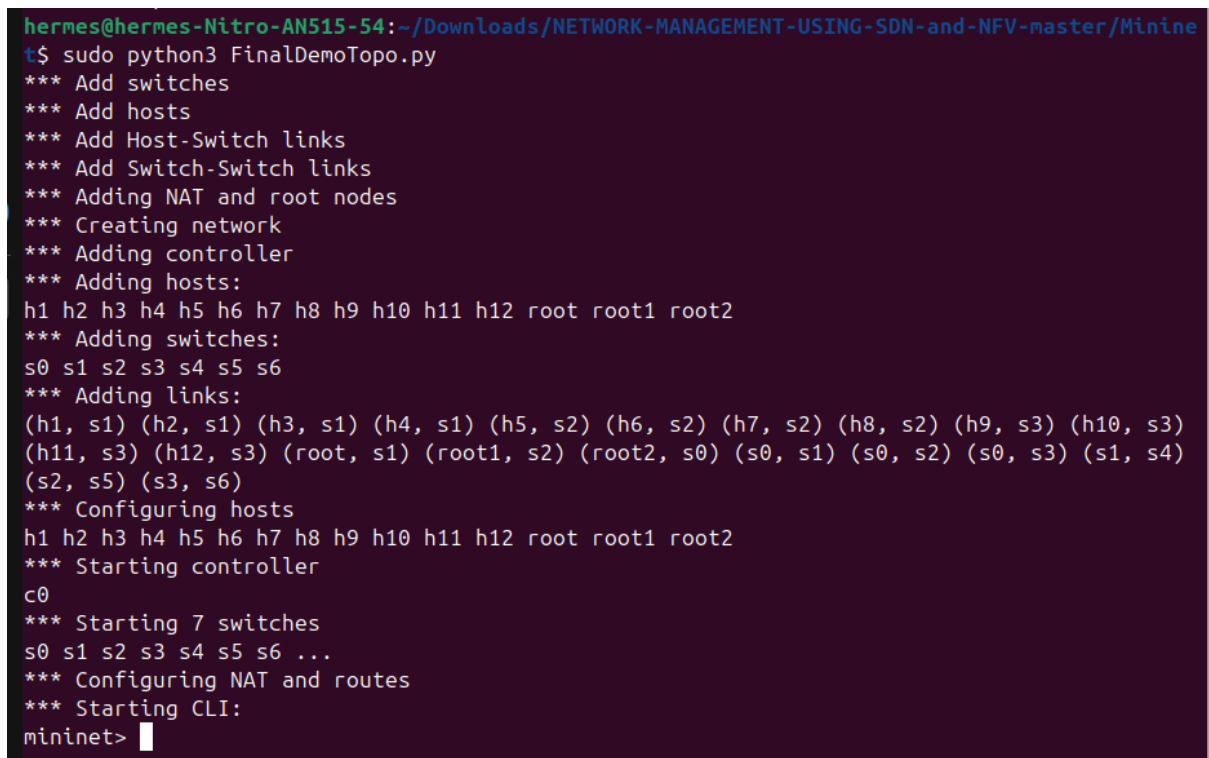


Figure 5.7: Running topology script

## 5.2 Overview

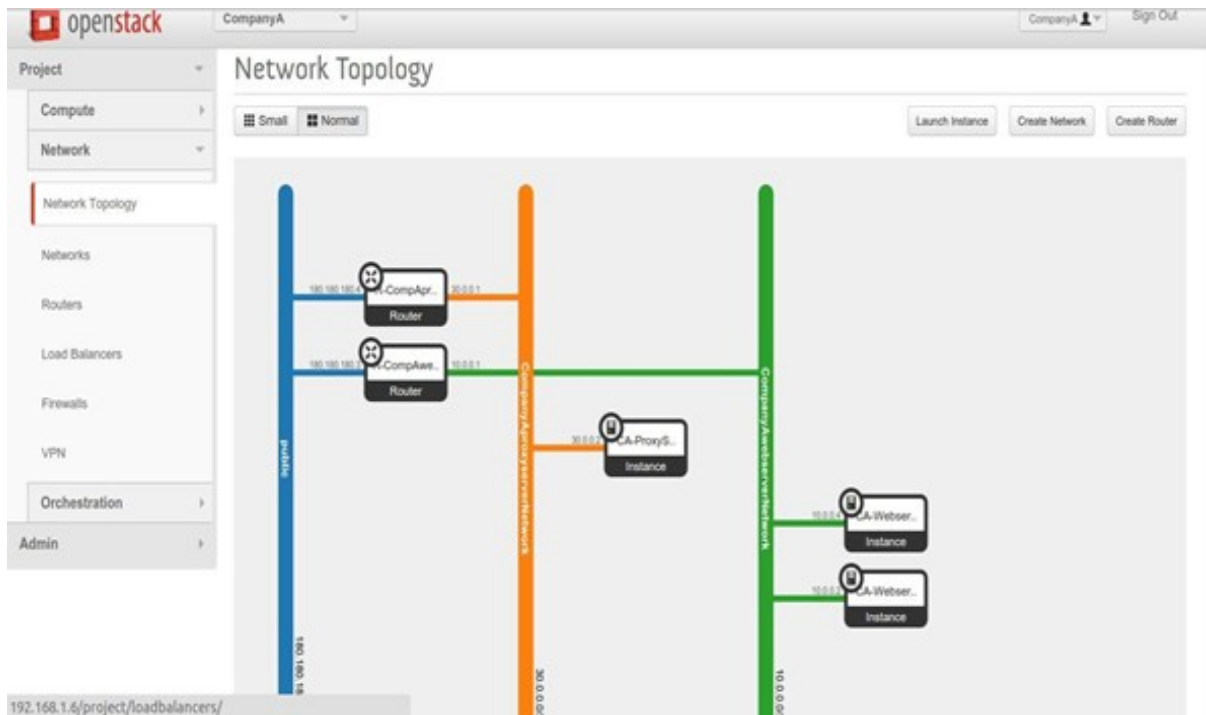


Figure 5.8: OpenStack dashboard view of CompanyA

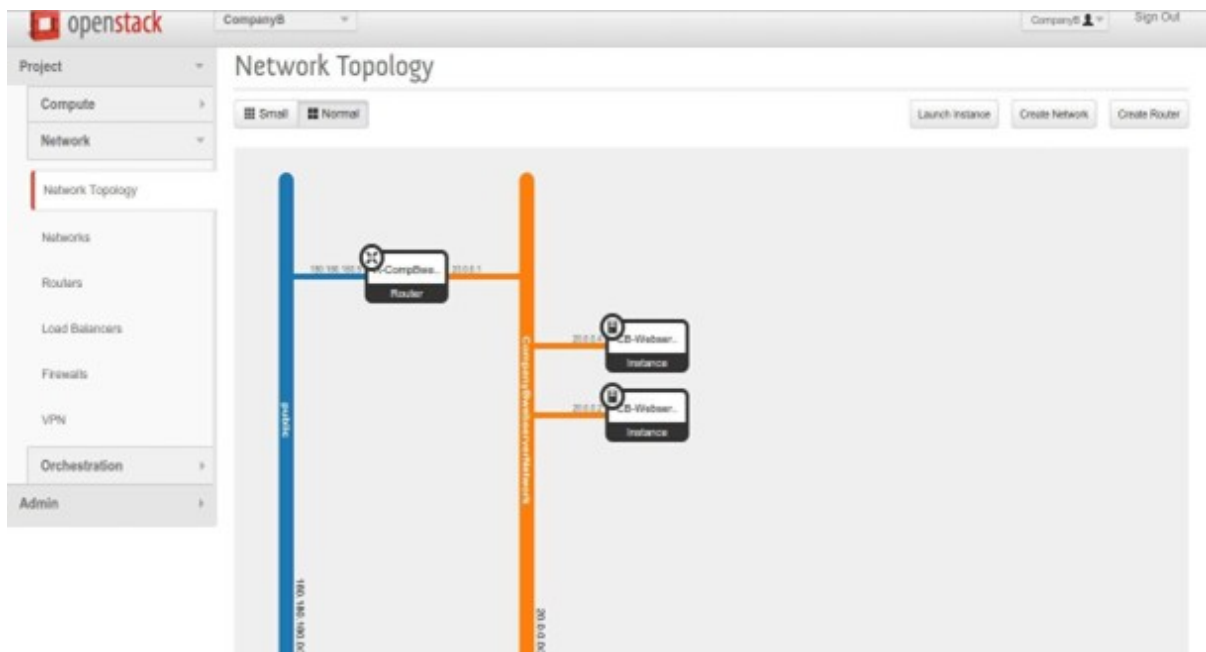


Figure 5.9: OpenStack dashboard view of CompanyB

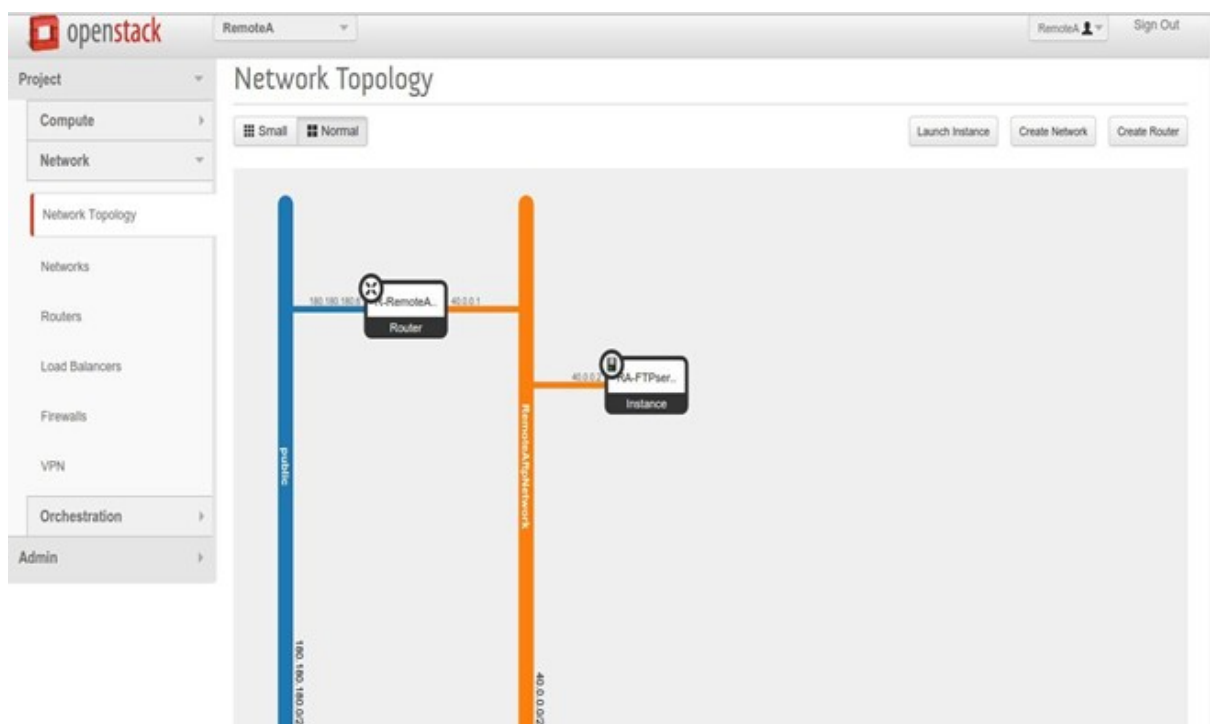


Figure 5.10: OpenStack dashboard view for RemoteA

## 5.3 Analysis

- Network Flexibility emerges from SDN implementation that enables administrators to make traffic rule adjustments automatically without needing manual intervention.
- Resource Optimization occurs when NFV and containerization (Docker + Kubernetes) manage hardware depending systems to create efficient resource utilization through scaling and orchestration processes.
- The platform scales VNFs automatically as part of Kubernetes operations to match traffic volume changes thus delivering enhanced performance under increased capacity requirements.
- Each Virtual Network Function within the system operates through containerization which promotes both isolation between units and independent module operation. Each service remains independent from other services because updating or failure will not impact them.
- Low-latency routing and dynamic control through real-time flow rule implementation occurs due to the use of Ryu controller and network state-based traffic handling.
- The Mininet platform provides an environment to emulate diverse network configurations through safe system tests ensuring correct system functionality before actual deployment.
- Python scripts generate performance and flow behavioral analytics in real time for the optimization of both VNF functions and traffic patterns.
- Open-source tools combined with free proprietary hardware make the system economical and available for both research activities and small deployment applications.

## Chapter 6

# Conclusion and Future Works

### 6.1 Conclusion

The designed implementation of the proposed network management system proves that SDN integration with NFV can solve traditional networking infrastructure challenges. The system implements open-source Mininet with Ryu and Docker while using Kubernetes to establish a productive virtualized network environment that offers automatic service deployment and efficient management. As an SDN component it substantially enhances network visibility and traffic policy control through centralized administration for application control throughout the network system. NFV allows service providers to install virtual network services like firewalls and intrusion detection systems inside containers which cuts down the need for expensive proprietary hardware systems.

A Kubernetes deployment serves as an additional automated solution for system scalability which links seamlessly to this operational design. The architecture provides Virtual Network Functions (VNFs) with durability together with comprehensive management solutions and instantaneous workload flexibility. The architecture accomplishes functional needs such as optimized resource management and fast service deployment and rapid traffic routing in addition to creating a solid base that allows future additions including AI-based intelligence and extended deployment.

The major results from this project become evident through both its simultaneously functional and economical implementation. The constructible system depends on only open-source components which allows it to deploy on standard hardware systems available to academic organizations alongside researchers who also operate in small enterprises. As an alternative to conventional networking configurations it functions effectively for developing testing methods and training techniques related to programmable network technologies. The project solution meets its objective for network management modernization by striking a balance between system complexity and operational performance and scalability.

### 6.2 Future Works

The proposed system achieves successful integration of SDN and NFV to build a programmable and scalable network management solution but future enhancements can be developed through several promising directions. Researchers should prioritize future development in Artificial Intelligence systems and Machine Learning methods within their

projects. SDN controller gains predictive abilities through these technologies which allow it to discover anomalies and predict traffic patterns and make automated decisions by processing historical and real-time data. Enhanced network systems will develop into automated platforms that require minimal human supervision through these improvements.

The system requires expansion for a multi-controller architecture to represent a major advancement. A distributed SDN controller deployment system enhances network reliability and load distribution and response performance in extensive or wide-ranging network environments. The system becomes ready for practical enterprise or telecom environments through its ability to function even when a single controller faces performance limitations.

The security domain represents an additional area for improving this system. The current system enables integration of basic VNFs including firewalls as well as intrusion detection systems. As this project evolves its developers should consider incorporating advanced security virtual network functions which combine deep packet analysis methods with behavior analysis techniques and blockchain technology for secure network communication and integrity authentication. The system's resistance to contemporary cybersecurity threats including DDoS attacks and malware infiltration and insider threats would increase through these improvements.

Future development of this project should focus on deploying the system in hybrid cloud platforms including AWS, Azure and OpenStack for performance testing with live operational workloads. Such deployment would provide essential knowledge about how resources are distributed along with service chaining and network reliability under unpredictable real-world conditions.

The system usability would benefit significantly from a user-friendly web-based interface that enables network performance visualization in addition to VNF management and SDN policy configuration.

## Chapter 7

## References

1. Cisco's, "*SDN, Transformation Through Innovation*". Available at: <http://www.cisco.com/web/solutions/trends/sdn/index.html>
2. F5 white paper, "*Network Functions Virtualization—Everything Old Is New Again*". Available at: <http://www.f5.com/pdf/white-papers/service-provider-nfv-white-paper.pdf>
3. EMC2, "*WAN Optimization Controller Technologies*". Available at: <http://www.emc.com/collateral/hardware/technical-documentation/h8076-wan-optimization-tb.pdf>
4. Citrix white paper, "*Application Delivery Controller*". Available at: [https://www.citrix.com/content/dam/citrix/en\\_us/documents/products-solutions/what-is-an-application-delivery-controller-adc.pdf](https://www.citrix.com/content/dam/citrix/en_us/documents/products-solutions/what-is-an-application-delivery-controller-adc.pdf)
5. Forbes, "*Two Vital Keys For The Future Of The Software-Defined Data Center*". Available at: <http://www.forbes.com/sites/symantec/2014/09/25/two-vital-keys-for-the-future-of-the-software-defined-data-center/>
6. ONF, "*Software Defined Networking*". Available at: <https://www.opennetworking.org/sdn-resources/sdn-definition>
7. The OpenDayLight Project. Available at: <http://www.opendaylight.org/project>
8. Project FloodLight, "*Open Source Software for Building Software-Defined Networks*". Available at: <http://www.projectfloodlight.org/floodlight/>
9. ONF, "*OpenFlow Protocol*". Available at: <https://www.opennetworking.org/ja/sdn-resources-ja/onf-specifications/openflow>
10. OpenStack. Available at: <http://www.openstack.org/>
11. OpenNF. Available at: <http://opennf.cs.wisc.edu>
12. OpenNF. Available at: <http://opennf.cs.wisc.edu>
13. Mininet, "*An Instant Virtual Network on your Laptop (or other PC)*". Available at: <http://www.mininet.org>

14. Bitbucket, Group2UnnA. Available at: <https://bitbucket.org/group2unna/ik2200/src/4e9828a5fe33861bebcd49b3fdde3adf0c4a524d/Mininet/?at=Final>
15. Mini-Stanford topology. Available at: <https://bitbucket.org/peymank/hassel-public/wiki/Mini-Stanford>
16. Fedora 20, All-in-One Image. Available at: [https://wiki.opendaylight.org/images/HostedFiles/Fedora20\\_ODL\\_OpenStack.zip](https://wiki.opendaylight.org/images/HostedFiles/Fedora20_ODL_OpenStack.zip)
17. OpenStack Icehouse install using DevStack. Available at: <http://sreeninet.wordpress.com/2014/05/11/openstack-install-using-devstack/>
18. Midokura GitHub. Available at: <https://github.com/midokura/odp-ovsdb/tree/master/resources/openstack>