

(This is the draft report and would like to know your suggestions about Table of Content, Objectives, Implementation and Use cases.)



Master Thesis

SDN based Network Management in Emulated environment

Submitted by:	Harshal Rajan Vaze
Matriculation no.:	1269879
First examiner:	Prof. Dr. Ulrich Trick
Second examiner:	Prof. Dr. Armin Lehmann
External supervisor:	Dr. Peter Gröschke

Content

1	Introduction	4
1.1	Aim and Motivation	4
1.2	Problem Statement	5
1.3	Thesis Structure	5
2	Theoretical Background	6
2.1	Software defined Networking	6
2.2	SDN Controllers	7
2.2.1	ONOS	7
2.2.2	ODL	7
2.2.3	Ryu	7
2.3	Virtual Emulated Environment	7
2.3.1	Mininet	7
2.3.2	GNS3	7
2.4	Previous Work	
3	Requirements Analysis	9
3.1	General Objectives	9
3.2	Work Plan	9
3.3	Previous Work	10
4	Realization	10
4.1	With Mininet	10
4.2	With GNS3	10
4.3	Problems identified	10
4.4	Use Cases	11
4.4.1	Use case-1: Basic network architecture	11
4.4.2	Use case-2: Testing network with multiple Controllers	11
4.4.3	Use case-3: Testing network with IPv6 addressing	11
4.4.4	Use case-4:	11

5	Summary and Perspectives	12
6	Abbreviations	12
7	References	12
8	Appendix	12

1 Introduction

In this era of network virtualization and automation, many functions are moving towards virtualized and more centralized control, allowing for more dynamic functions and easier optimization. For networking it would mean spinning up the virtualized versions of traditional network functions allowing for more complex decisions making, automating the networks, and changing network configuration more efficiently.

Software-Defined Networking (SDN) is an approach to networking that uses software-based controllers and application programming interfaces (APIs) to communicate with underlying hardware infrastructure and direct traffic on a network. This method is different from that of traditional networks, where the configuration of dedicated hardware devices like Routers and Switches needs to be done node by node to control network traffic. SDN can create and control a virtual network or also control a traditional hardware via software. Because the control plane is software-based, SDN is much more flexible than traditional networking. It allows administrators to control and manage the network from a centralized user interface, without adding more hardware.

A SDN controller is an application in a software-defined networking architecture that manages flow control for improved network management and overall network performance. The SDN controller platform typically runs on a server and uses protocols to tell switches how to forward the packets. SDN controllers direct traffic according to forwarding policies that a network operator puts in place, thereby minimizing manual configurations for individual network devices. By taking the control plane off of the network hardware and running it instead as software, the centralized controller facilitates automated network management and makes it easier to integrate and administer various applications. In effect, the SDN controller serves as a sort of operating system (OS) for the network.

The SDN controller is the core of a software-defined network. It resides between network devices at one end of the network and applications at the other end. Any communication between applications and network devices must go through the controller. The SDN controller communicates with applications via northbound interfaces. The Open Networking Foundation (ONF) created a working group in 2013 focused specifically on northbound APIs and their development. The industry never settled on a standardized set, however, largely because application requirements vary so widely. Typically northbound RESTful APIs secured with TLS are used to push the configuration changes from the application to the SDN controller. The SDN controller talks with individual network devices using a southbound interface with the help of well-known OpenFlow protocol and other protocols such as NetConf, BGP, SNMP, etc. These southbound protocols allow the controller to configure network devices and choose the optimal network path for application traffic. OpenFlow was created by ONF in 2011 and latest version of OpenFlow protocol is OpenFlow 1.5.1.

1.1 Aim and Motivation

There are various open-source SDN controllers developed by various groups of developers and are ready to be deployed and tested in the production environment. SDN controllers are primarily made to provide some policies and centralized managements for the networking layer, which will further allow inter-working configurations between the interfaces. Iptables, network namespaces, and Open vSwitch are components of the Linux kernel's L3 IP routing, Linux bridges, and SDN controller technologies. Being open-source controllers, not all of the available SDN controllers are developed on the same scale of mechanism in terms of supporting these various protocols, especially new versions of protocols and many of them are not developed yet to completely support all the services required for direct integration with the real world network. Therefore, the aim of this Master Thesis is to research and study different open-source SDN controllers in terms of their functionality with services and protocols.

After research, the aim would be to build and setup the virtual network with network elements in the emulation software for testing the functionality of SDN controller. Further, to develop some use cases for demonstrating the services and protocols used in the network.

1.2 Problem Statement

Due to advances in the Information and Communication Technology, the configuration and management of the network components becomes highly complex and time-consuming. A fundamental characteristic of SDN is the logically centralized, but physically distributed controller component. SDN offers to batch-configure automatically multiple components in one step, while the traditional way would mean to log into each device. Many operators struggle with the migration from IPv4 to IPv6, SDN with its centralized control and the possibility to reduce human error due to increased poses an opportunity to help make this migration easier. The controller maintains a global network view of the underlying forwarding infrastructure and programs the forwarding entries based on the policies defined by network services running on top of it. The traditional networking approach has very limited facilities to explore these aspects of networking and the goal would be to study these futuristic characteristics of networking.

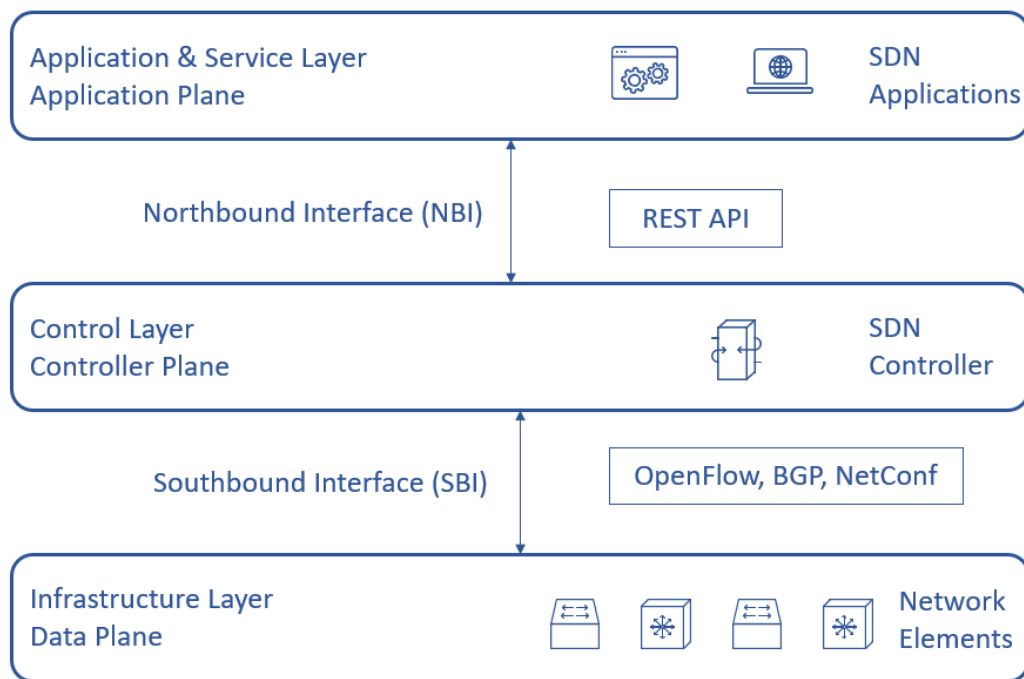
1.3 Thesis Structure

This thesis work is structured in five main chapters: chapter 1 gives the introduction to the topic and discusses the statement of the problem. Chapter 2 reviews the theoretical background knowledge of the topics associated with this thesis. Furthermore it also describes the in-depth information about the components used in this Thesis. In chapter 3, requirement analysis, the main objectives of the thesis are discussed focusing on question ‘which open-source SDN controllers are available to be tested in emulated environment?’. The realization of the implementation in different environments along with some use cases are presented and discussed in the chapter 4 named, realization. This chapter focuses on the question ‘how to implement suitable environment for testing the use cases?’. Finally, at end a summary, perspectives and future work are provided in chapter 5.

2 Theoretical Background

2.1 Software defined Networking

Dr. Martin Casado developed an architecture to separate control and forwarding functions of networking de-vices, migrating control to a centralized policy server. This architecture evolved to what is now known as Software Defined Networking (SDN) today. One of the first challenges was the need for a common South Bound Interface (SBI) protocol between the SDN Controller and the forwarding networking device. OpenFlow developed by the Open Networking Foundation (ONF) and is used over a secure channel (Transport Layer Security (TLS) over Transmission Control Protocol (TCP) port 6633 and 6653) to modify the group and flow tables in a OpenFlow networking device. OpenFlow has evolved to version 1.5.1.



2.2 SDN Controllers

2.2.1 ONOS

2.2.2 ODL

2.2.3 Ryu

2.3 Virtual Emulated Environment

2.3.1 Mininet

Mininet is a tool specially designed for software defined networks. It is an emulator of a network and it is used to visualize and test the OpenFlow switches and application of software-defined networks in a virtualized environment. Mininet provides the space to develop and test software-defined network applications without the need to set up a physical environment. It gives a network testbed thereby allowing to develop and test applications that are using OpenFlow protocols. It also gives the flexibility to integrate python API, thereby paving the way for creating and experimenting with networks. It also has a CLI which is aware of topology and OpenFlow thereby allowing to debug or run network tests for applications. Mininet is majorly used as a learning tool to test, experiment, and learn about software-defined networks and it is preferable because it is very fast and helps to create customizable topologies.

Advantages of Mininet :

- It is very fast and takes very little time for booting.
- It is easy to install and use.
- It saves money because the emulators are cost-effective instead of testing with hardware devices.
- It is also very easy to connect with other network devices.
- It has high availability.

2.3.2 GNS3

GNS3 (Graphical Network Simulation 3) is an open source, free network software emulator , which emulates and interconnects networking devices, including routers, switches, firewalls, and other devices which are interconnectable within OSI model. GNS3 is used to configure, test and troubleshoot virtual and real networks. Latest version (Version 2.2.33.1) of GNS3 support devices from multiple network vendors such as Cisco virtual switches, Cisco SD-WAN components, Open vSwitch, Brocade vRouters, Cumulus Linux switches, Docker instances, Juniper vRR, HPE VSRs, Nokia vSIM, multiple Linux appliances and many others. Furthermore, GNS3 offers no limitation on the number of devices supported or complexity of network topologies. The only possible limitations are in the CPU and memory of the hardware that runs it.

GNS3 emulates the hardware of a device and runs real images in the virtual device, so it can be used to design complex networks and do simulations about them. Since it runs real images, it is necessary to have the images of the devices to be simulated. GNS3 offers on his official website different appliances that have and already configured image and can be used to emulate a device.

GNS3 consists of two software components:

- Client part: The GNS3-all-in-one software (GUI)
- Server part: The GNS3 virtual machine (GNS3 VM)

The GNS3-all-in-one is the graphical user interface (GUI) where the topologies can be created. When the topologies are created, the devices are hosted and run by a server process. The options for the server part are the following:

- Local GNS3 server: run on the same PC where the GUI is installed.
- Local GNS3 VM: run on the same PC using virtualization software such as VirtualBox or VMware.
- Remote GNS3 VM: run remotely using VMware ESXi or in the cloud.

For this Thesis VirtualBox was preferred to host the devices on the GNS3 VM.

Advantages of GNS3:

- Free and Open Source software
- No monthly or yearly license fees
- No limitation on number of devices (only limitation is the user hardware: CPU and memory)
- Supports multiple switching options (Open vSwitch, Cumulus Linux Switches, IOU/IOL Layer 2 images, VIRT IOSvL2):
- Supports all VIRT images (IOSv, IOSvL2, IOS-XRv, CSR1000v, NX-OSv, ASA v)
- Supports multi-vendor environments
- Can be run with or without hypervisors
- Supports both free and commercial hypervisors (VirtualBox, HyperV, VMware workstation, ESXi, Fusion)
- Downloadable, free, pre-configured and optimized appliances available to simplify deployment
- Native support for Linux without the need for additional virtualization software
- Software from multiple vendors freely available
- Large and active community (800,000+ members)

Disadvantages:

- Cisco as well as other software images need to be supplied by user (download from Cisco.com, or purchase VIRT license, or copy from physical device).
- Not a self-contained package, but requires a local installation of software (GUI).
- GNS3 can be affected by PC's setup and limitations because of local installation (firewall and security settings, company laptop policies etc.).

(more detailed information is yet to be written about all sub-chapters in this chapter.)

3 Requirements Analysis

Due to advances in the Information and Communication Technology, the configuration and management of the network components becomes highly complex and time-consuming. A fundamental characteristic of SDN is the logically centralized, but physically distributed controller component. SDN offers to batch-configure automatically multiple components in one step, while the traditional way would mean to log into each device. Many operators struggle with the migration from IPv4 to IPv6, SDN with its centralized control and the possibility to reduce human error due to increased poses an opportunity to help make this migration easier. The controller maintains a global network view of the underlying forwarding infrastructure and programs the forwarding entries based on the policies defined by network services running on top of it. The traditional networking approach has very limited facilities to explore these aspects of networking and the goal would be to study these futuristic characteristics of networking.

3.1 General Objectives

The detailed requirements for this thesis and the involved tasks are stated below:

- Build a suitable network with different network devices in the emulation software.
- Manage different services and network configurations with SDN controller in an emulated environment.
- Create and distribute the network configurations for network devices.
- Develop a rationale and setup an IPv4 and IPv6 scheme for the network.
- Provide services and user groups that have different requirements.
- Evaluate advantages and disadvantages of network with SDN controller over traditional network.
- Proof and validation of functioning failover mechanisms to improve resilience.

Throughout this thesis, the following research questions are answered:

- Research possible open-source SDN controllers to implement.
- Research alternative configuration methods with the goal of finding the best possible method to configure and manage the network through Network Controller.
- How to provide different paths in the network with different QoS properties?
- Algorithms that are responsible for the optimization of the paths.
- When a service is accessible at multiple times, how to choose the best one.

3.2 Work Plan

Work flow for this Thesis is as follows;

- Research about the SDN controllers.
- Research about the emulation software.
- Installation and testing of different SDN controllers.
- Configuration of these SDN controllers.
- Installing and activating the features required for the services and applications running on the SDN controller.
- Installing different network devices preferably open-source devices to run inside the emulated environment.
- Setting up the connection between these networking devices and the SDN controller.
- Document the difficulties and errors found while installation and debugging the connectivity between the different network devices and SDN controller

- Creating and implementing different network topologies.
- Creating use cases suitable for SDN controller in the emulated environment.
- Implementing and testing these use cases in the emulated environment.

3.3 Previous Work

Through the study and research about implementation of SDN controllers in virtual environments, the following points were found;

- Currently various Open-source SDN controllers are available to be implemented and tested in the environment. To name few, OpenDayLight (ODL), Open Network Operating System (ONOS), Ryu and Faucet.
- ONOS and ODL are built to have centralized architectures. Hence, they tend to be easier to maintain and confer lower latency between the tightly coupled southbound APIs and northbound APIs.
- Faucet is built to have distributed architectures which generally are more complex to maintain and deploy but can allow the platform to scale more effectively. By decoupling the processing of PCE, Telemetry and Southbound interface traffic, each function can be scaled independently to avoid performance bottlenecks.
- Whereas, Ryu is different to the other options, although having a core set of programs that are run as a platform, it is better thought of as a toolbox, with which SDN controller functionality can be built.
- ONOS and ODL are written in Java, for which development resources are abundant in the market, with good supporting documentation and libraries available.
- Ryu and Faucet are written in Python, a well-supported language and has an active community developing the framework. The documentation is concise and technical, aimed at developers to maximize the utility of the system.
- Both ODL and ONOS benefit from large developer and user communities under the Linux Foundation Networking banner. Many large international players are involved in the development and governance of these projects, which could add to the longevity and security over time.
- Ryu and Faucet are well supported, targeted controllers. Due to the emerging nature of the field, both options look to have a bright future, with a simpler, streamlined approach to change submission and testing.
- Most of the software defined networking testbeds are implemented and analyzed with the Mininet environment.
- Mininet being network emulator is used to deploy quick network topologies with Open vSwitch supporting SDN protocols along with in-built SDN controller. These are ran as virtual instances inside the Mininet environment.
- In GNS3, real images of the network devices are used which are almost identical to the real world network devices. Quite few experimental research was found where SDN controller was tested with network topologies created using the network devices in GNS3 environment.
- New versions of OpenDayLight controller do not support the GUI (DLUX application features) and L2 switch applications. Study reveals the projects associated with these applications were separated from OpenDayLight.

4 Current Status of progress

After researching about various SDN controllers, few suitable SDN controllers were selected for building an virtual environment on the emulation software. These SDN controllers were Open Network Operating System (ONOS), OpenDayLight (ODL) and Ryu. Similarly, to create the network with network elements, two emulation software are selected, namely, Mininet and GNS3.

4.1 Installation

For setting up the testbed for this Thesis work, all the major components were installed on the Oracle VM VirtualBox application. Different virtual machines were created for each SDN controller.

Mininet-VM and GNS3 VM were also running on different instances of Virtual Machines. Mininet-VM is capable of creating topologies with Open vSwitches, Hosts and SDN Controller all in itself. When topologies are created in GNS3 using the all-in-one software GUI client, the devices created are need to be hosted and run by a server process, for this GNS3 VM instance is used. All SDN controllers have their complete installation in respective virtual machines.

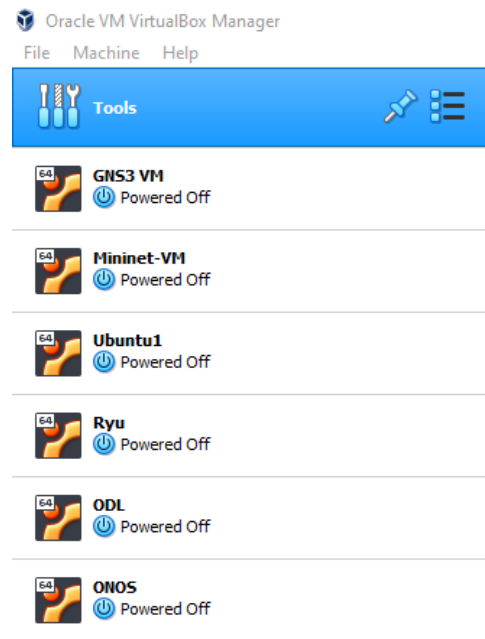


Figure 4.1: Different virtual machines instances in VirtualBox

Topologies created inside the GNS3 and Mininet are able to connect to the outside SDN controller using OpenFlow protocol. SDN controllers are running as a service, meaning after installation the controllers VMs were configured to directly run them as SDN controller without need of any separate command to start the controller.

4.2 Implementation

After complete installation and necessary configuration of the components required for this Thesis work, a simple topology with four Open vSwitches and routers was created in the GNS3 application. For testing purpose of the routers are deployed at the far ends of the topology.

In GNS3, to connect any network device to the SDN controller a NAT interface is used. A L2 switch (Switch1 in Figure 4.2) is used so that all devices can be directly connected to the NAT interface with just single interface. SDN controller used here is ONOS controller.

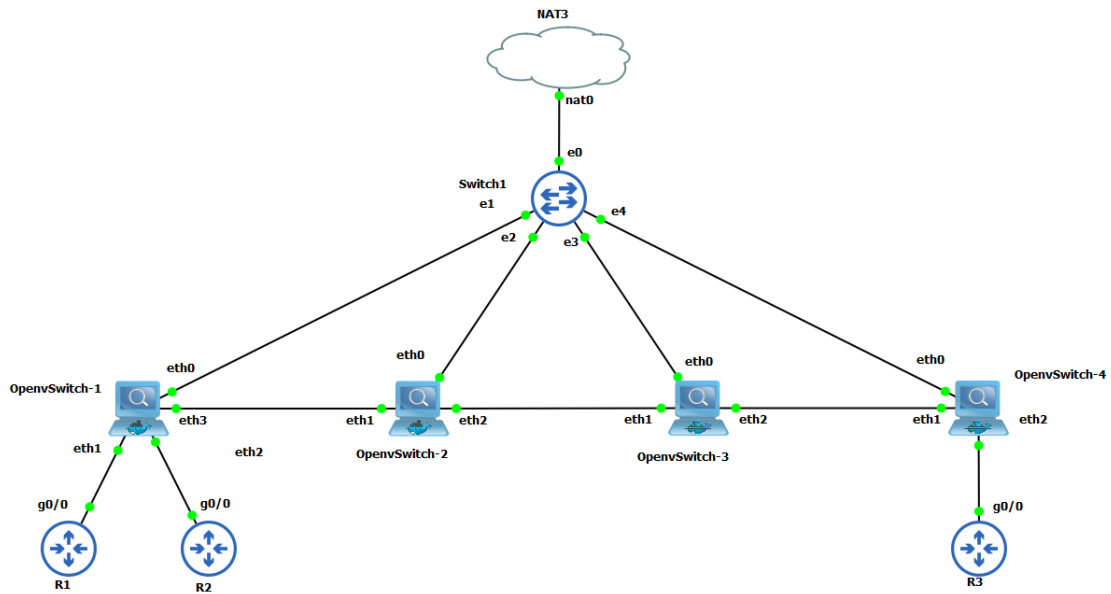


Figure 4.2: Topology created in the GNS3 with different Network devices

Before connecting the Open vSwitches to the SDN controller, OpenFlow protocol version (here OpenFlow version 1.3) needs to be specified.

Bridge br0 is the management interface on the Open vSwitch which accepts the configuration commands from the SDN controller. To connect this interface to the SDN controller, external IP address of the SDN controller (here, 192.168.0.113) and port number of OpenFlow protocol needs to be specified (here, 6653).

```
OpenSwitch-1
#
#
#
# ovs-vsctl set bridge br0 protocols=OpenFlow13
#
# ovs-vsctl set-controller br0 tcp:192.168.0.113:6653
#
# ovs-vsctl show
a5e2f07c-602a-4749-903c-2d1345ba13a1
  Bridge "br1"
    datapath_type: netdev
    Port "br1"
      Interface "br1"
        type: internal
  Bridge "br0"
    Controller "tcp:192.168.0.113:6653"
    is_connected: true
    fail_mode: secure
    datapath_type: netdev
    Port "eth13"
      Interface "eth13"
    Port "eth5"
      Interface "eth5"
    Port "eth11"
      Interface "eth11"
    Port "eth2"
      Interface "eth2"
    Port "eth7"
      Interface "eth7"
    Port "eth6"
      Interface "eth6"
    Port "eth10"
      Interface "eth10"
    Port "eth4"
      Interface "eth4"
    Port "eth8"
      Interface "eth8"
    Port "eth14"
```

Figure 4.3: Commands and confirmation of Open vSwitch connection to the ONOS controller

As seen in Figure 4.3, interface Br0 was connected to the controller at the given IP address and the Boolean value was set to **True**. To confirm the connection from the controller side, ONOS GUI can be ran on the browser. The *onos-gui* feature must be installed in ONOS. The GUI listens on port 8181. The base URL is `/onos/ui`; for example, to access the GUI on localhost, use: <http://localhost:8181/onos/ui>

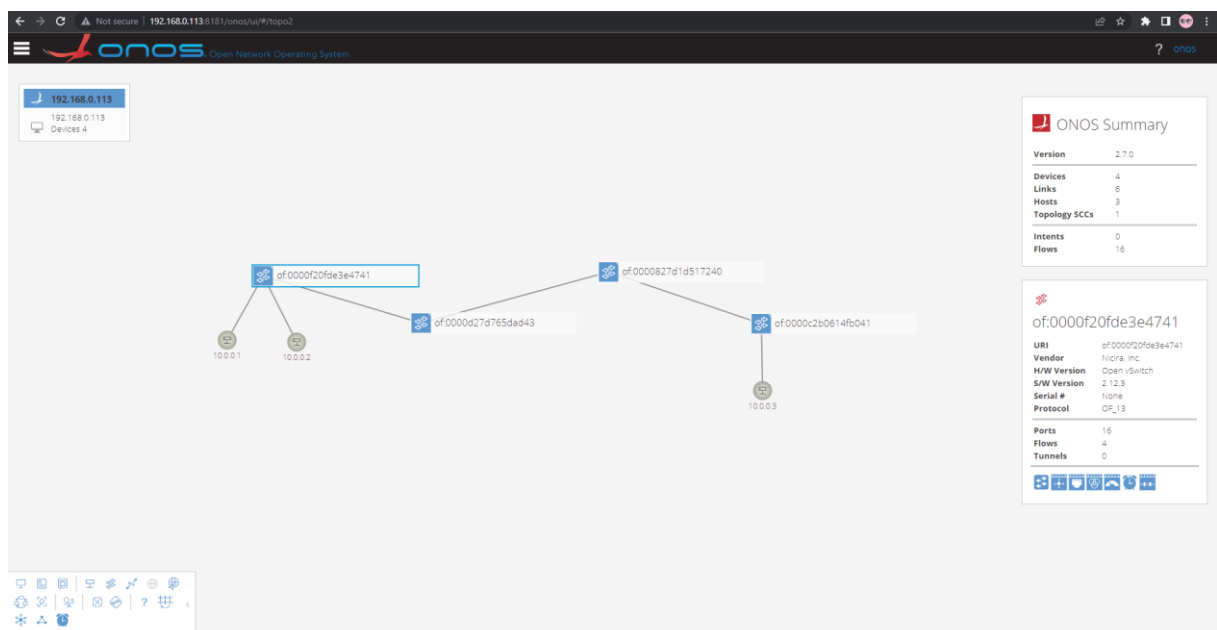
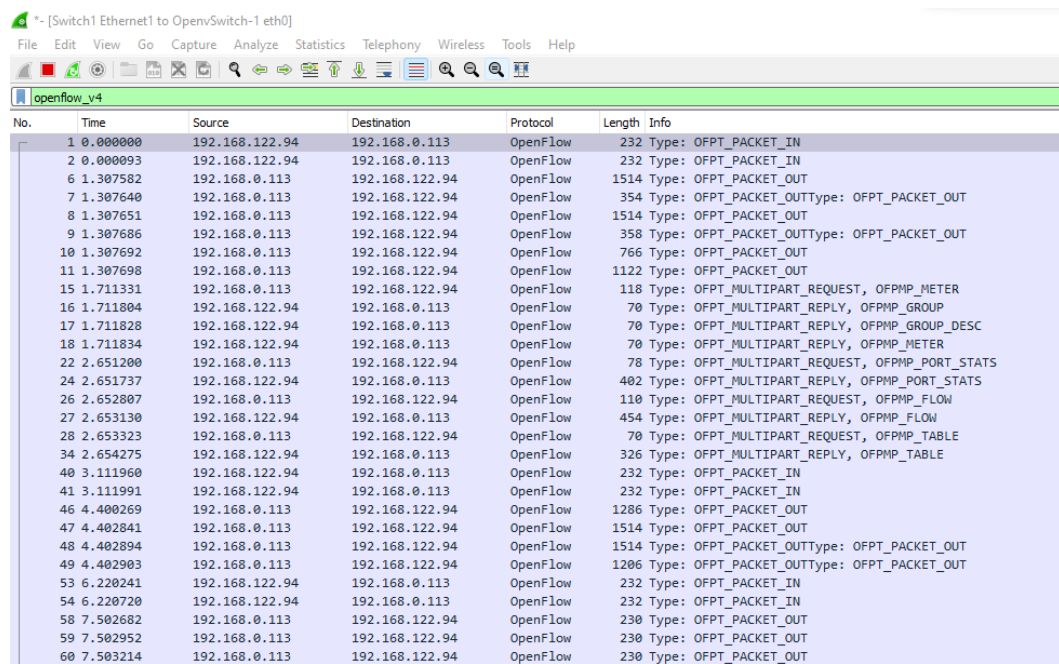


Figure 4.4: Topology view on the ONOS controller GUI

The ONOS GUI is a single-page web-application, providing a visual interface to the ONOS controller. ONOS GUI provides a great and easy to understand information about the ONOS controller and the topology connected to with it. Information such as applications installed on controller, number of devices connected, number of hosts, port numbers used in the topology, number of packets transferred between the links, and many more information is easily accessible through ONOS GUI. The ONOS Cluster Node Panel indicates the cluster members (controller instances) in the cluster. The Summary Panel gives a brief summary of properties of the network topology. The Topology Toolbar provides push-button / toggle-button actions that interact with the topology view.

A Wireshark was ran between the ONOS controller and one of the Open vSwitch to release the packets transferred between them. Southbound protocol OpenFlow was also captured in this link and figure 4.5 shows the Wireshark capture of the same.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.122.94	192.168.0.113	OpenFlow	232	Type: OFPT_PACKET_IN
2	0.000093	192.168.122.94	192.168.0.113	OpenFlow	232	Type: OFPT_PACKET_IN
6	1.307582	192.168.0.113	192.168.122.94	OpenFlow	1514	Type: OFPT_PACKET_OUT
7	1.307640	192.168.0.113	192.168.122.94	OpenFlow	354	Type: OFPT_PACKET_OUTType: OFPT_PACKET_OUT
8	1.307651	192.168.0.113	192.168.122.94	OpenFlow	1514	Type: OFPT_PACKET_OUT
9	1.307686	192.168.0.113	192.168.122.94	OpenFlow	358	Type: OFPT_PACKET_OUTType: OFPT_PACKET_OUT
10	1.307692	192.168.0.113	192.168.122.94	OpenFlow	766	Type: OFPT_PACKET_OUT
11	1.307698	192.168.0.113	192.168.122.94	OpenFlow	1122	Type: OFPT_PACKET_OUT
15	1.711331	192.168.0.113	192.168.122.94	OpenFlow	118	Type: OFPT_MULTIPART_REQUEST, OFPMP_METER
16	1.711804	192.168.122.94	192.168.0.113	OpenFlow	70	Type: OFPT_MULTIPART_REPLY, OFPMP_GROUP
17	1.711828	192.168.122.94	192.168.0.113	OpenFlow	70	Type: OFPT_MULTIPART_REPLY, OFPMP_GROUP_DESC
18	1.711834	192.168.122.94	192.168.0.113	OpenFlow	70	Type: OFPT_MULTIPART_REPLY, OFPMP_METER
22	2.651200	192.168.0.113	192.168.122.94	OpenFlow	78	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_STATS
24	2.651737	192.168.122.94	192.168.0.113	OpenFlow	402	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_STATS
26	2.652807	192.168.0.113	192.168.122.94	OpenFlow	110	Type: OFPT_MULTIPART_REQUEST, OFPMP_FLOW
27	2.653130	192.168.122.94	192.168.0.113	OpenFlow	454	Type: OFPT_MULTIPART_REPLY, OFPMP_FLOW
28	2.653323	192.168.0.113	192.168.122.94	OpenFlow	70	Type: OFPT_MULTIPART_REQUEST, OFPMP_TABLE
34	2.654275	192.168.122.94	192.168.0.113	OpenFlow	326	Type: OFPT_MULTIPART_REPLY, OFPMP_TABLE
40	3.111960	192.168.122.94	192.168.0.113	OpenFlow	232	Type: OFPT_PACKET_IN
41	3.111991	192.168.122.94	192.168.0.113	OpenFlow	232	Type: OFPT_PACKET_IN
46	4.400269	192.168.0.113	192.168.122.94	OpenFlow	1286	Type: OFPT_PACKET_OUT
47	4.402841	192.168.0.113	192.168.122.94	OpenFlow	1514	Type: OFPT_PACKET_OUT
48	4.402894	192.168.0.113	192.168.122.94	OpenFlow	1514	Type: OFPT_PACKET_OUTType: OFPT_PACKET_OUT
49	4.402903	192.168.0.113	192.168.122.94	OpenFlow	1206	Type: OFPT_PACKET_OUTType: OFPT_PACKET_OUT
53	6.220241	192.168.122.94	192.168.0.113	OpenFlow	232	Type: OFPT_PACKET_IN
54	6.220720	192.168.122.94	192.168.0.113	OpenFlow	232	Type: OFPT_PACKET_IN
58	7.502682	192.168.0.113	192.168.122.94	OpenFlow	230	Type: OFPT_PACKET_OUT
59	7.502952	192.168.0.113	192.168.122.94	OpenFlow	230	Type: OFPT_PACKET_OUT
60	7.503214	192.168.0.113	192.168.122.94	OpenFlow	230	Type: OFPT_PACKET_OUT

Figure 4.5: Topology view on the ONOS controller GUI

In figure 4.6 we can observe the OpenFlow protocol version 1.3 being used for communication between the ONOS controller and Open vSwitches.

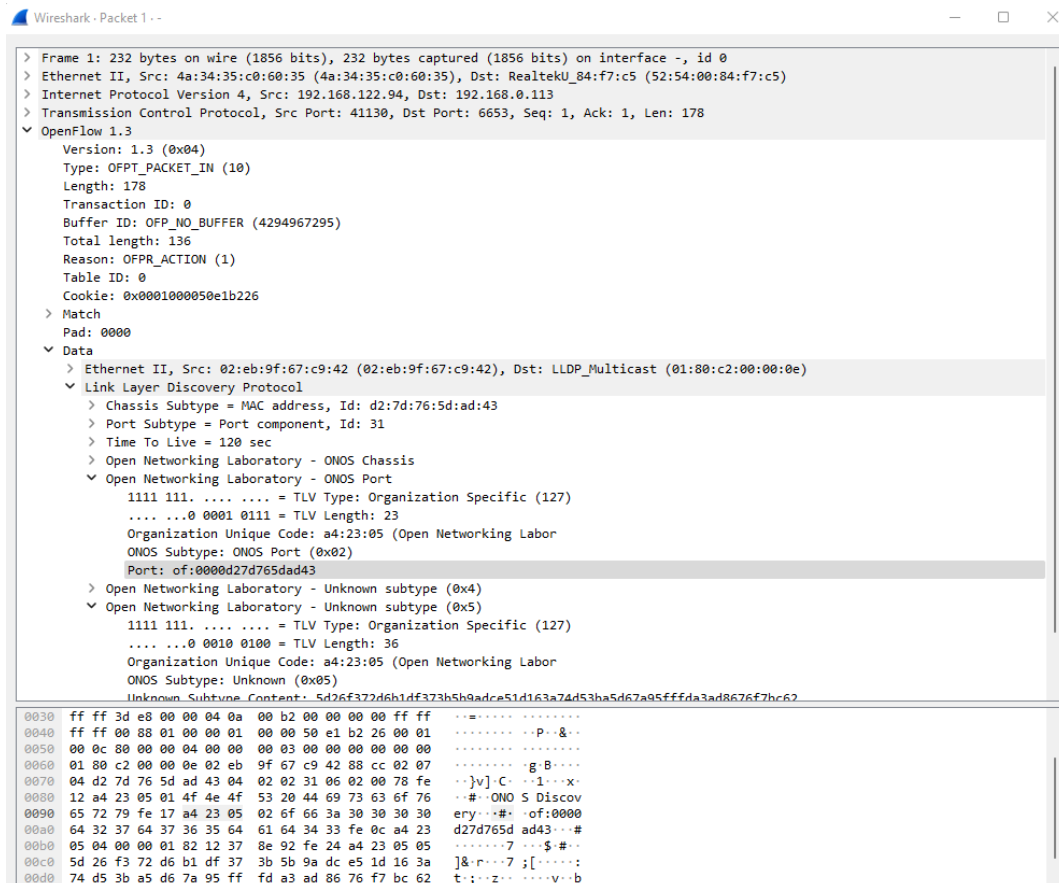


Figure 4.6: Topology view on the ONOS controller GUI

4.3 Problems identified

During the installation of the components used in this Thesis few challenges were faced. Similarly while implementing and testing the connectivity between the SDN controller and the network devices few problems were identified. Following are the problems encountered and possible fixes for the same;

- Installation of SDN controllers is easier when they are first downloaded as a packet rather than directly installing from source code.
- Open vSwitch application available for GNS3 on GNS3 Marketplace (named, Open vSwitch with management interface) has some errors and needs to be fixed before using.
- New versions of OpenDayLight controller do not support the GUI and L2 switch applications since these projects were separated from OpenDayLight.
- Unlike previous versions of OpenFlow protocol, latest version of this protocol needs to be specified on all the networking devices.
- Before connecting the Open vSwitch to the controller, Open vSwitch needs to be specified the version of OpenFlow protocol to be used.
- Also, need to specify the latest version of OpenFlow protocol while executing the topology commands on Mininet.

4.4 Next tasks

With the basic environment being set up the next tasks would be to test this environment by keeping following points into consideration;

- Build a suitable network topology with different network devices in the emulation software.
- Manage different services and network configurations with SDN controller in an emulated environment.
- Create and distribute the network configurations for network devices.
- Develop a rationale and setup an IPv4 and IPv6 scheme for the network.
- Develop some use cases and test them in the built environment.