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
Date of start:	01.05.2022
Date of submission:	04.10.2022

Statement

I confirm that I have written this thesis on my own. No other sources were used except those referenced. Content which is taken literally or analogously from published or unpublished sources is identified as such. The drawings or figures of this work have been created by myself or are provided with an appropriate reference. This work has not been submitted in the same or similar form or to any other examination board.

_____

Date, signature of the student

Content

List of Figures

List of Tables

1 Introduction

In this era of network virtualization and automation, network functions are being moved to a more centralised setup and are migrated from boxes to virtual network functions (VNF), allowing for more dynamic functions and easier optimization. For networking it would mean spinning up the virtualised versions of traditional network functions allowing for automation of the networks and changing the network configuration more efficiently.

Software-defined Networking (SDN) [1] is an approach to networking that uses software-based controllers and application programming interfaces (APIs) to communicate with underlying hardware infrastructure and manage the traffic on the 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 network built from physical and virtual network components via software. Because the control plane is centralised, SDN can perform more immediate mechanisms like re-configuration of a network faster and more efficient than a traditional network. It allows administrators to control and manage the network from a centralised user interface, without adding more hardware.

The SDN controller is the vital entity in a software-defined networking architecture that manages transmission of traffic of underlying network nodes for improved network management and overall network performance. The SDN controller platform typically runs on a server and uses protocols to instruct network devices 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 decoupling the control plane from the network hardware and running it instead as software, the centralised 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.

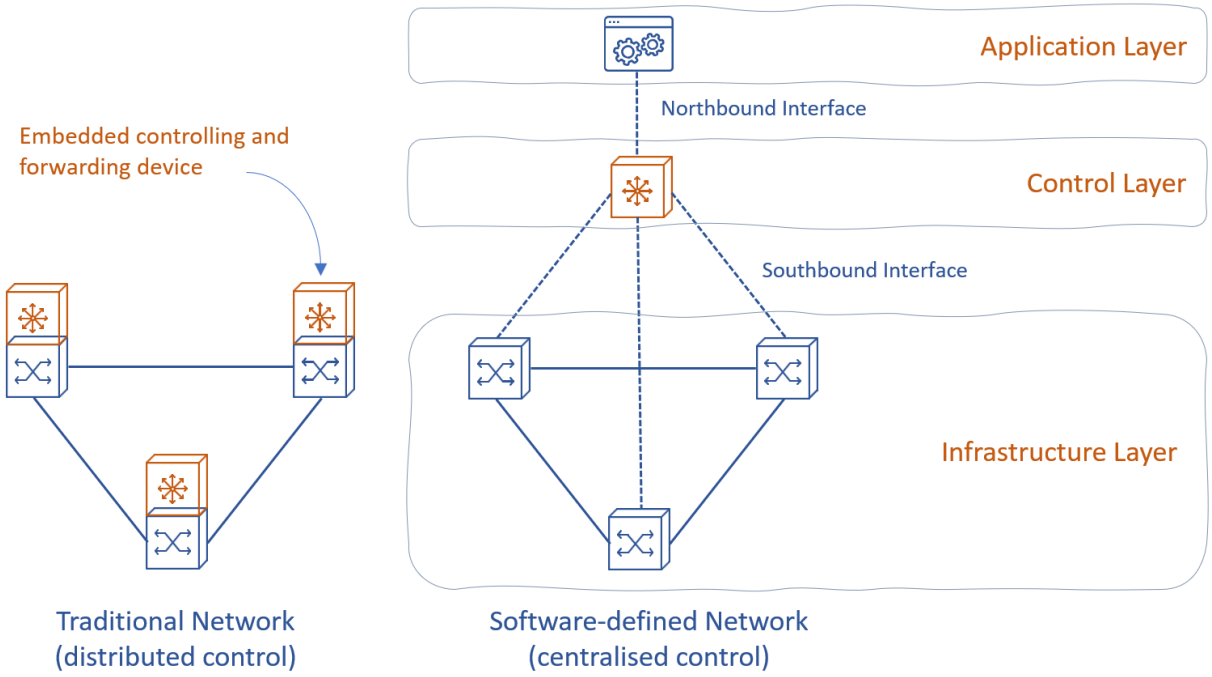


Figure 1.1: Traditional Network compared to Software-defined Network

The SDN controller is the core of a software-defined network. It resides between network devices at one end of the network and SDN 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) [2] created a working group in 2013 focused specifically on Northbound APIs and their development. The industry never settled on a standardised set, however, largely because application requirements vary so widely. Mainly Northbound RESTful APIs [3] secured with TLS are used to push the configuration changes from the application to the SDN controller. The SDN controller communicates 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 the network devices and choose the optimal network path for application traffic.

1.1 Aim and Motivation

As the software-defined networks developed along the years, different network components were developed by several communities. The primary component, SDN controllers were developed to provide control rules and centralised management commands for the network devices in the infrastructure layer, which further allows inter-working configurations between the interfaces. By means of this objective, several SDN controllers were developed. They might show similarities as well as differences in terms of functionality and development of their framework. Numerous challenges were faced by the early released versions of these SDN controllers. The goal was to study these different SDN controllers in terms of their development, functionality and managing capabilities of the underlying network devices.

Several applications were developed to assist the SDN controller for implementation network services. Along with this, different network components such as software switches, protocols utilised in the SDN infrastructure are available today. Therefore, the aim of this Master Thesis was to research and study these different components related to the software-defined networks in terms of their functionality with services.

After detailed research, the testbed of virtual network was setup with the selected network components in the network emulation software for testing the performance and functionality of these SDN components. Further, some use cases were implemented for demonstrating the different network services and addressing the difficulties faced nu internet service providers.

1.2 Problem Statement

The network utilisation and requirements are increasing rapidly because of advances in the Information and Communication Technology (ICT), which demands for network automation infrastructure. Traditional networks are complicated to automate due to their distributed decision-making processes for switching and routing. Moreover, re-configuration and management of traditional networks becomes highly complex, expensive, and time-consuming. The fundamental characteristic of software-defined networking is to have a centralised infrastructure to control and manage the network services. 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 centralised control and the possibility to reduce human error, 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 various topics associated with this thesis. Furthermore, it also describes the in-depth information like basic operations and functionality 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 implemented and tested in the network emulated environment?’. The evaluation of the implementation in different environments along with some use cases designed to test the performance of SDN controller are presented and discussed in the chapter 4 named, realization. This chapter focuses on the question ‘how the software-defined networks try to answer the challenges faced by internet service providers in various aspects of networking?’ Finally, at the end a summary of work, perspectives and future work are listed in chapter 5.

2 Theoretical Background

2.1 Software-defined Networks

Software-defined Networks (SDN) is a modern network architecture in which the control plane and the data plane are separated from each other. The architecture of SDN involves three primary layers which are the Application and Service layer, the Control layer, and the Infrastructure layer. These layers are defined on the application plane, the control plane and the data plane of the network device as seen in the following figure. The SDN initiative led by the Open Networking Foundation (ONF) [2], proposed this open architecture to answer the networking difficulties with the ability to simplify the network configuration automation and provide platform to deploy programmable networks. The SDN architecture enhances the degree of abstraction by separating the network data and control planes, in contrast to the conventional distributed network architecture, where network devices are closed and vertically coupled, binding the software with hardware.

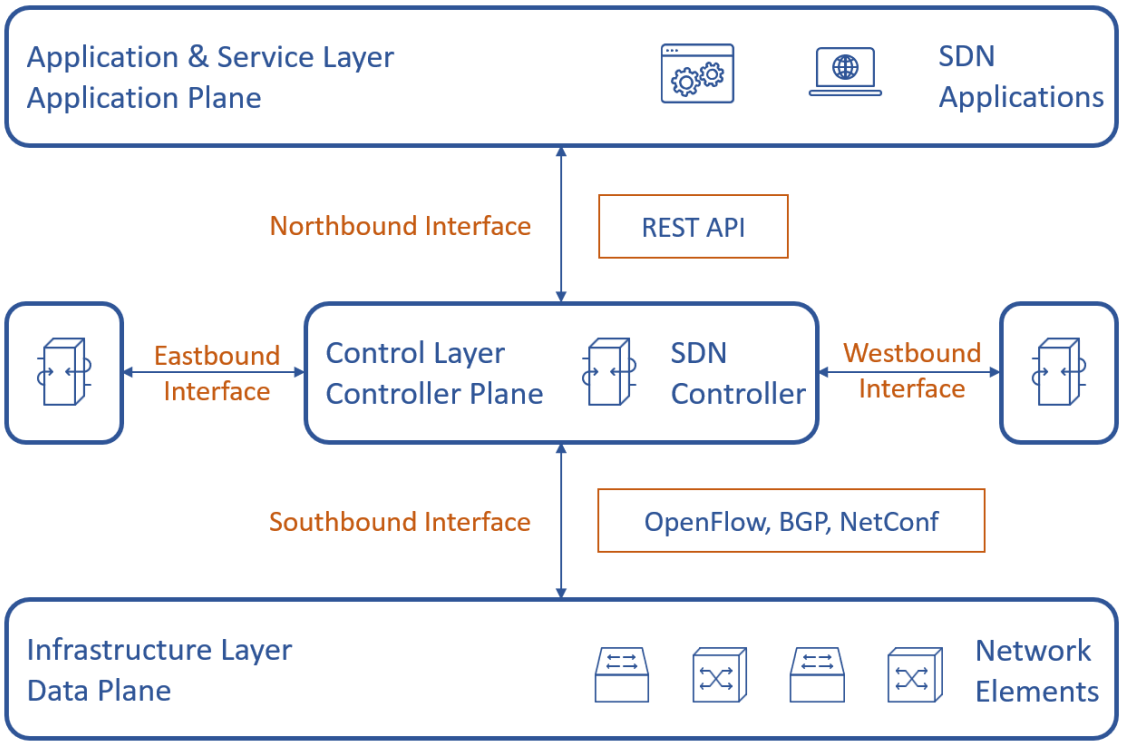


Figure 2. 1: Software-defined Network architecture

SDN relies on the concept of a simple forwarding switch, which forwards the packets according to the set of rules. By decoupling the control and the data plane, the network device becomes the switch, and all the control and forwarding logic is handled by the software controllers. Because of its open concept, SDN makes it considerably simpler and more flexible for researchers to create and execute new, creative network functions and services. These features of SDN are considered to make network management easier and more effective by enabling the network to be managed according to high-level policies that are presented as applications.

The Northbound interface (NBI) between the application plane and control plane is utilised for human interaction or applications with the SDN controller and the Southbound interface (SBI) between the control plane and the data plane is utilised to transfer the set of rules to the network devices. Through SBI, the SDN controller is capable to effectively handle the heterogeneity of the underlying infrastructure. During the initial days of SDN, there was a requirement to develop the common SBI protocol which is not bound to any vendor and satisfies the open concept of the SDN. OpenFlow [4] protocol developed by the ONF, which works over a secure channel of Transport Layer Security (TLS) with Transmission Control Protocol (TCP) port 6633 and 6653 to update the forwarding flow rules in a OpenFlow networking device. The OpenFlow switches [5] were developed to support the OpenFlow features like flow tables, group table, meter table, etc. Seeing the advancement of SDN, network devices vendors also developed their devices to be compatible with the SDN standards.

The application plane of SDN consists of various applications which are designed to implement the network services and features on the underlying network. This plane communicates with the control plane through Northbound interface (NBI). The SDN applications learns the deployed network topology and requirements of the configured network services from the SDN controller. Accordingly, SDN applications provides the solution to the SDN controller through NBI, and the controller translates them into controlling commands and forwarding rules for each of the network devices in the network. Different SDN applications were developed according to the base platform of different SDN controllers to support the network services [6].

The control plane of SDN consists of a centralised software controller, which is the core of SDN networks. The SDN controller acts as a mediator between the SDN applications and the infrastructure layer. The fundamental function of the SDN controller is to convert the requirements of application plane and data plane to each other in a way that they understand their role to achieve the basic goal of implementing the desired network service. Well-known as Network Operating System (NOS), this plane has the connectivity through NBI, SBI and also with the other SDN controllers through Eastbound and Westbound interface. To increase the scalability and high availability of the network, the best practice is to have multiple SDN controllers in the network. In distributed SDN controllers environment, each controller consists of its domain with underlying forwarding devices and to exchange the information of this domain with other controllers eastbound interface is used [3]. Different controllers use different eastbound interface protocols for example, SDNi [7] by OpenDaylight [8] controller, Raft [9] by ONOS [10]. The westbound interface is used to establish the communication between SDN controller and the traditional network devices such as routers mostly, by using Border Gateway Protocol (BGP).

The data plane of SDN consists of the network elements, the only hardware component required in the SDN network. Also known as Infrastructure layer, this plane comprises of network devices, mostly OpenFlow switches [11], which are responsible for forwarding the data packets in the network. This plane interacts with SDN controller through the Southbound interface (SBI), providing the controller information about the link status between the devices, connected host status, switchport information, etc. All the queries of the network devices about the received data packet at any switchport, are answered by the SDN controller. The well-known SBI protocol, OpenFlow [4] provides the control plane with the network control requirements and allows it to configure the forwarding flow rules on the OpenFlow switches.

The challenges faced by the static and inflexible architecture of the legacy networks in this increasing dynamic networking era, SDN provides the opportunity to answer these difficulties and provide better platform for next generation networking. As a result of the separation of the control plane and data plane, which simplifies network management, the main advantage of SDN is the logical construction of a centralised controller. By enabling centralised administration of the network, SDN aims to execute network operations on large scale, more efficiently, with greater speed and provide automatism in the network.

2.2 SDN Controllers

The Controller being the key component of the software-defined networks, it resides on the control plane of the SDN architecture and has connectivity with the SDN applications and underlying network. The SDN controller provides flexible and efficient management of the network with its basic operation of converting the network services learnt from the SDN applications into appropriate forwarding flow rules for the underlying network devices. Also, the network devices provide the information and updates about the status of links, devices, etc. to the SDN controller, which further pushes this information to appropriate applications. Keeping this functionality in mind, over the years various open-source SDN controllers were developed by different open-source communities and network device vendors.

The first SDN controller developed was the NOX controller [12] in 2009 by the Nicira Networks which was later acquired by VMware [13]. The NOX controller was the basic platform for other SDN controllers which were developed later such as NOX-MT, POX [14], ONIX [15], Ryu [16] and Beacon [17]. Along the years new features were developed for better functionality of the SDN controller, and to overcome the challenges faced by other SDN controllers, new SDN controllers kept emerging. To name few, Floodlight [18], OpenDaylight [8], ONOS [10], Orion [19] Disco [20] and Kandoo [21]. Later, big network device vendors like HPE, Cisco, Juniper VMware, IBM also participated and designed their own approach towards building the SD-WAN architectures. The SD-WAN architecture was designed to fully support applications hosted in on-premises data centres, cloud services, and Software-as-a-Service (SaaS) services. Today, a variety of SDN controllers are available for re-

search and evaluation from which the potential ones joined projects with other communities working in same field of interest whereas, some disappeared because of scarcity of contributors and developers. The open-source SDN controllers listed in table 2.1 were the most cited ones in research and academic papers.

Table 2. 1: Popular Open-Source SDN controllers

Controller	Implementation	Developers	Architecture
NOX	C++	Nicira	Physically-centralised
POX	Python	Nicira	Physically-centralised
Ryu	Python	NTT laboratories	Physically-centralised
Floodlight	Java	Big Switch Networks	Physically-centralised
ONOS	Java	Open Networking Lab	Physically-distributed logically-centralised
OpenDaylight	Java	The Linux Foundation	Physically-distributed logically-centralised

To extend the SDN to large networks, the different architectures were developed. These types include physically-centralised and physically-distributed which further categorises into logically-centralised, logically-distributed, flat and hierarchical architecture. In the initial days of SDN, with the aim to have a centralised controlling point of network, the SDN controllers were developed with the physically-centralised architecture. However, because of problems like reliability of SDN controller, scalability and high availability of network functions physically-distributed architectures were developed. Various research and surveys have been carried out, for providing the overview of these architectures [22], challenges faced by these architectures [23] and controller selection criteria [24].

A physically-centralised architecture of SDN consists of physically-centralised control plane with single SDN controller for complete underlying network. The SDN was designed with the aim to have a single point of contact to control and manage all the network devices which only contains data planes. However, a single SDN controller for the entire network is an unsafe practice because if the controller breaks down the entire network collapses along with it. Correspondingly, the managing of complete network devices with all the control packets requests and replies especially, with the OpenFlow protocol which exchanges a lot of control packets between SDN controller and OpenFlow switches, may create the bottleneck situation for the SDN controller. Also, to tackle the different requirements of the large networks such as reliability, scalability, high availability, performance guarantee, etc. this centralised architecture of SDN doesn't have a feasible solution. All these demands requested for deploying the multiple controllers for smooth functioning of the network. The NOX [12], first SDN controller was developed on this physically-centralised architecture, but it faced the throughput issues. Later, new versions, NOX-MT and POX [14] controller were developed to tackle this problem and provide comfortable platform for developers. Also, Java based Floodlight [18] controller by Big Switch Networks [25] and Python based Ryu [16] controller by NTT laboratories [26] were developed on this physically-centralised architecture. Limitations of the innovative first SDN controllers were answered by developing new SDN controllers with improved feature sets, better scalability and performance, and resiliency for real-world application though they were developed on the physically-centralised architecture of the SDN.

A physically-distributed architecture of SDN was developed to address all the limitations faced by the physically-centralised architected SDN controllers. As the name suggests, this architecture consists of physically-distributed control points of the network. The scale-out approach of the control plane was proved to be potential solution to achieve the requirements of the large-scale networks [27]. The SDN controllers like ONIX [15], OpenDaylight [8], ONOS [10] and Orion [19] were developed supporting this architecture. The physically-distributed architecture of SDN can be further classified into logically-centralised and logically-distributed architecture [22] along with flat and hierarchical architecture of SDN [23].

In other words, the physically-distributed logically-centralised architecture means multiple controllers are deployed in the network but they act like a single controller in terms of operations and functionality. In this type of architecture, multiple controllers are implemented in the control plane of the SDN. These controllers together form a cluster and are able to communicate with each other via East-Westbound interface and with network devices via SBI. A cluster formed of these controllers, distribute network control operations among themselves.

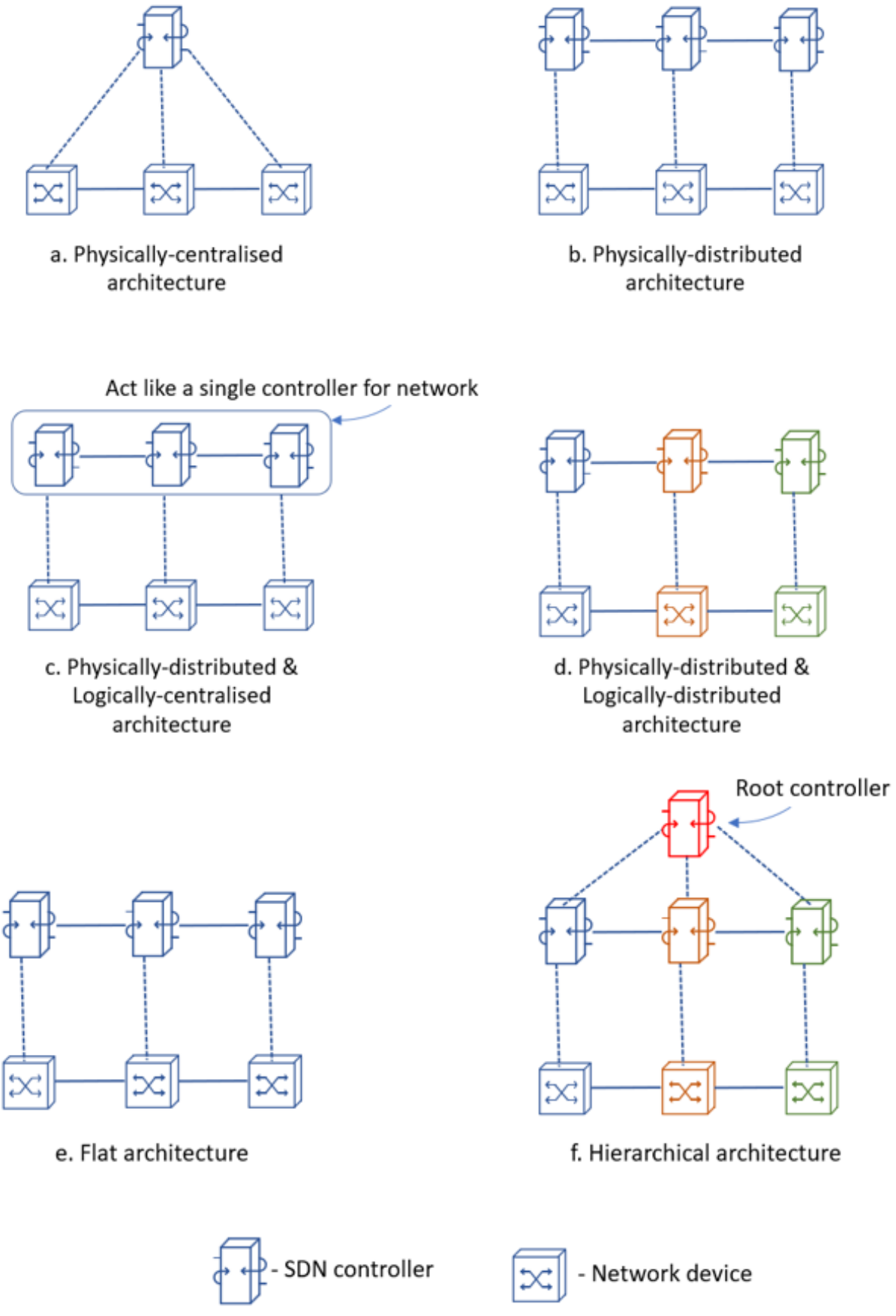


Figure 2. 2: Different architectures of SDN

All the controllers have the complete global view of network topology but may or may not have the controlling authority of all the network devices in physically-distributed logically-centralised architecture. The network devices are generally distributed among all the controllers to achieve highest performance guarantee. SDN controllers such as ONIX, OpenDaylight and ONOS were developed based on this type of architecture.

Whereas, in physically-distributed logically-distributed architecture, the control plane is physically as well logically separated in the network. In this type of architecture, multiple controllers form a cluster together, but all the controllers may or may not have the complete view of the network. Generally, every controller has a view of some section of the network and only has the controlling authority of the network devices associated in that section of the network. All the controllers communicate with each other and share the necessary network service information with each other. SDN controllers such as DISCO [20] and AtlanticWave [28] were developed based on this type of architecture. These were the two SDN architectures defining the logical centralisation and logical distribution of the controllers present in the control plane of the SDN. To classify the working of control plane itself, two more architecture types were proposed, and they were flat architecture and hierarchical architecture [23].

The flat architecture also known as horizontal architecture consists of controllers arranged in the horizontal approach in the control plane of the SDN. The control plane has a single layer of operation, and every controller is part of that one layer. The controllers can have logically-centralised or logically-distributed architecture for controlling the operations of the networks. Arranging controllers in the flat approach has the benefits like increase the scalability of the network, decrease in control latency and improved resiliency. SDN controllers such as ONIX, OpenDaylight and ONOS use this flat architecture of SDN.

On the other hand, the hierarchical architecture of SDN consists of several layers of the control plane. This type of architecture is also known as vertical architecture because the controllers are arranged in the vertical approach. The controllers are partitioned on the multiple levels, generally two or three, as per the requirements of the network. As per the deployment, all the controllers may or may not have the complete view of the network. Arranging controllers in the hierarchical approach further increases the scalability and performance of the network. SDN controllers such as Kandoo [21] and Orion [19] are part of this hierarchical architecture of SDN.

2.2.1 ONOS Controller

The Open Network Operating System (ONOS) [10] controller was developed by Open Networking Lab (ON.Lab) [2] in 2014. Later in 2015, ONOS joined the Linux Foundation [29] organisation for its project collaboration. The ONOS source code was written in Java and was released to start the ONOS open-source community. ONOS is distributed under the Apache 2.0 license. ONOS was developed on physically-distributed and logically-centralised architecture of SDN and supports the flat architecture of the control plane. ONOS controller, focuses on network scalability, reliability, and high performance through its distributed architecture.

ONOS is designed on four modules named, Code modularity, Configurability, Separation of Concern and Protocol agnosticism [30]. These modules provide the developers of the controller an organised platform to install the new services or protocols on the ONOS. ONOS operates on several subsystems that function together to provide the network services. These subsystems are:

- Application Subsystem - to deploy applications and provide accessibility to the controller.
- Device Subsystem - to manage the inventory of network devices.
- Host Subsystem - to manage the inventory of endpoints or hosts.
- Link Subsystem - to manage the inventory of links between the network devices.
- Topology Subsystem - to manage time-ordered snapshots of network graph views.
- PathService - to find paths between network devices or between endpoints.
- FlowRule Subsystem - to manage inventory of the flow rules.
- Packet Subsystem - to translate the packets information into applications services.
- Intent Subsystem - to translate applications outputs into control commands for network devices.
- Tunnel Subsystem - to support tunnel setup for applications.

For the seamless interworking of the subsystems, communication components are available to facilitate the efficient exchange of information. Along with this, the ONOS functionality is further divided into seven tiers out of which major three are Application tier, Core (control plane) tier and Network tier. All of these subsystem's function on at least one of these major tiers and [31] provides the detailed functionality of these subsystems along these tiers.

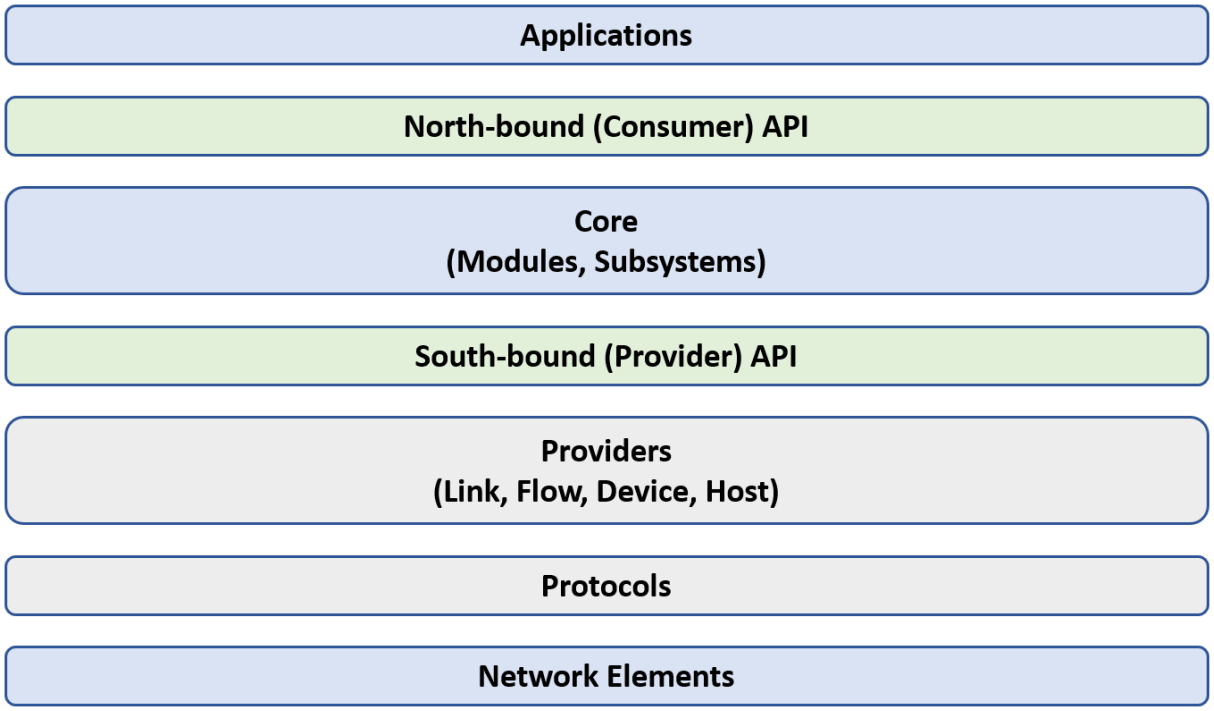


Figure 2. 3: ONOS controller architecture [31]

Over the years, the ONOS has been developed to support the majority of the SBI protocols and network services. Till the date of this thesis, there were 22 major releases with the latest version 2.7.0 named X-wing released on 17th January 2022. ONOS community declared this version to be the last release under 2.x series and the next major release will be part of 3.x series which will focus on ONOS cluster deployments on Kubernetes/Helm environments [32]. For the research and development purpose, ONOS provides two methods to install the controller in the environment. First, ONOS can be downloaded as a package and can be installed to use as an administrator. Second, ONOS can be installed from the source code and can be used to test the developed applications. Proper guides have been documented to install the ONOS in each environment along with necessary prerequisites and configuration [33]. For being the popular open-source SDN controller and well availability of the documentation, this controller was selected for studying and implementing the use cases. The ONF has developed and provided the great platform to strengthen the open-source community building together the next generation networks.

2.2.2 OpenDaylight Controller

The OpenDaylight (ODL) [8] controller was developed as a project by Linux Foundation [29] in 2014. The objective of the project was to create an open-source platform to advance the implementation and innovation of Software-defined Networking (SDN) and Network Functions Virtualization (NFV). The OpenDaylight source code was written in Java. OpenDaylight was crafted using physically-distributed and logically-centralised architecture of SDN and supports the flat architecture of the control plane.

OpenDaylight was developed as model-driven controller using YANG [34] as its modelling language. The controller operates on the three main technologies which are OSGI, Karaf and YANG. The Open Service Gateway Initiative (OSGI) [35] is a Java framework operating at the back end of OpenDaylight and allowing it to manage bundles and packages for exchanging information. Karaf [36] is an OSGI's application container which helps OpenDaylight's operations of packaging and installing applications. Yet Another Next Generation (YANG) is a data modelling language used by the controller's applications to configure the models. Similar to ONOS, OpenDaylight is developed to function on modular subsystems. Major subsystems of OpenDayLight are:

- Config Subsystem - a framework to configure, activate and inject dependencies.
- MD-SAL - (Model-Driven Service Abstraction Layer) a functionality to notify applications and data storage.
- MD-SAL Clustering - a functionality to from cluster and provide access to modelled data.

The functionality of OpenDaylight can be categorised into three components consisting of MD-SAL, Modular and Multiprotocol tier and S3P. MD-SAL is the core of the OpenDaylight controller whose basic function is to communicate between the network devices and applications. Various components operate with MD-SAL to exchange YANG modelled data for the network services. Modular and Multiprotocol tier provides the developers a flexible platform to design own applications and modules as per the requirement. OpenDaylight supports various Southbound interface protocols like OpenFlow, OVSDB, NETCONF, BGP, etc. These protocols and the network services can be individually selected and bundled together as per the requirements of the use case. The security, scalability, stability, and performance or S3P are the major fields where OpenDaylight community focuses on. The community groups perform tests on the controller to satisfy the requirements of the use cases [37].

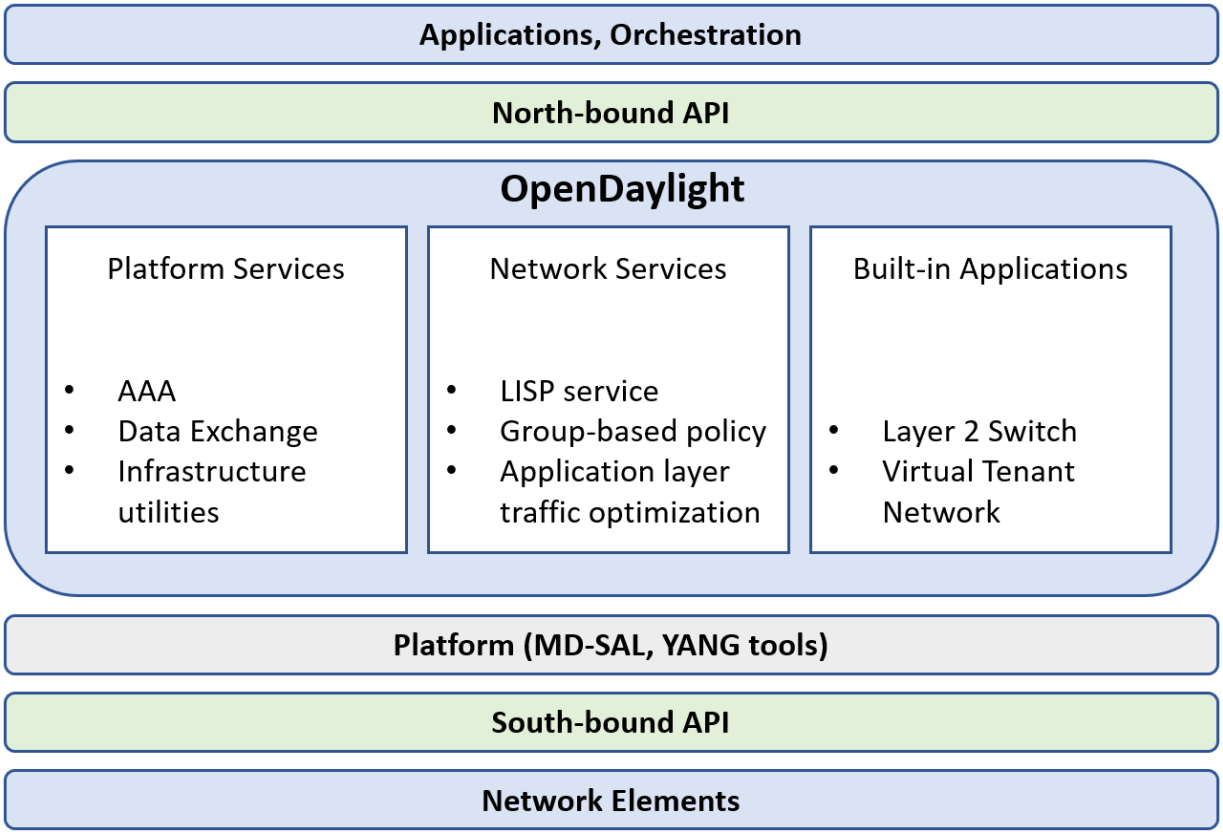


Figure 2. 4: OpenDaylight controller architecture [8]

Till the date of this thesis, there were 16 major releases with the latest version Sulfur-SR1 released in March 2022. Similar to the ONOS, for the research and development use, OpenDaylight provides two methods to install the controller in the environment. First, OpenDaylight can be downloaded as a package and can be installed to use as an administrator. Second, OpenDaylight can be installed from the source code and can be used to test the developed applications. The installation, configuration and how-to guide is well documented by the OpenDaylight community [38]. For being the popular open-source SDN controller and well availability of the documentation, this controller was selected for research purpose of this thesis.

2.2.3 Ryu Controller

The Ryu controller [16] was developed by Nippon Telegraph and Telephone Corporation (NTT) Laboratories [26] of Japan in 2013. The Ryu controller's source code was written in Python and is distributed under the Apache 2.0 license. The Ryu controller was designed using physically-centralised architecture of SDN.

Ryu is a component-based software-defined networking framework. The Ryu framework was designed to provide the platform for building the SDN applications, useful libraries, and well-defined APIs. Ryu provides the bunch of components which are useful for developing the SDN applications.

The major components and libraries of Ryu framework are:

- OF REST- to configure network devices through REST APIs.
- VRRP - to add virtual router redundancy protocol to support OpenFlow switches.
- OpenStack Quantum - to provide network-as-as-service with OpenStack.
- Stats component - to visualise and analyse the statistics of switches and stores the data.
- Topology component - to determine links in the network and create topology along with path calculation features.
- Topology viewer - to show topology and flows dynamically.

Other components such as Firewall, layer 2 switch, Endpoint, etc. are also part of the Ryu framework [39] and libraries such as NetFlow, OVSDB JSON, etc. consists of functions that are directly accessed by the components. Among the initial applications of Ryu controller were GRE tunnelling, VLAN support, Topology discovery and MAC based segregation.

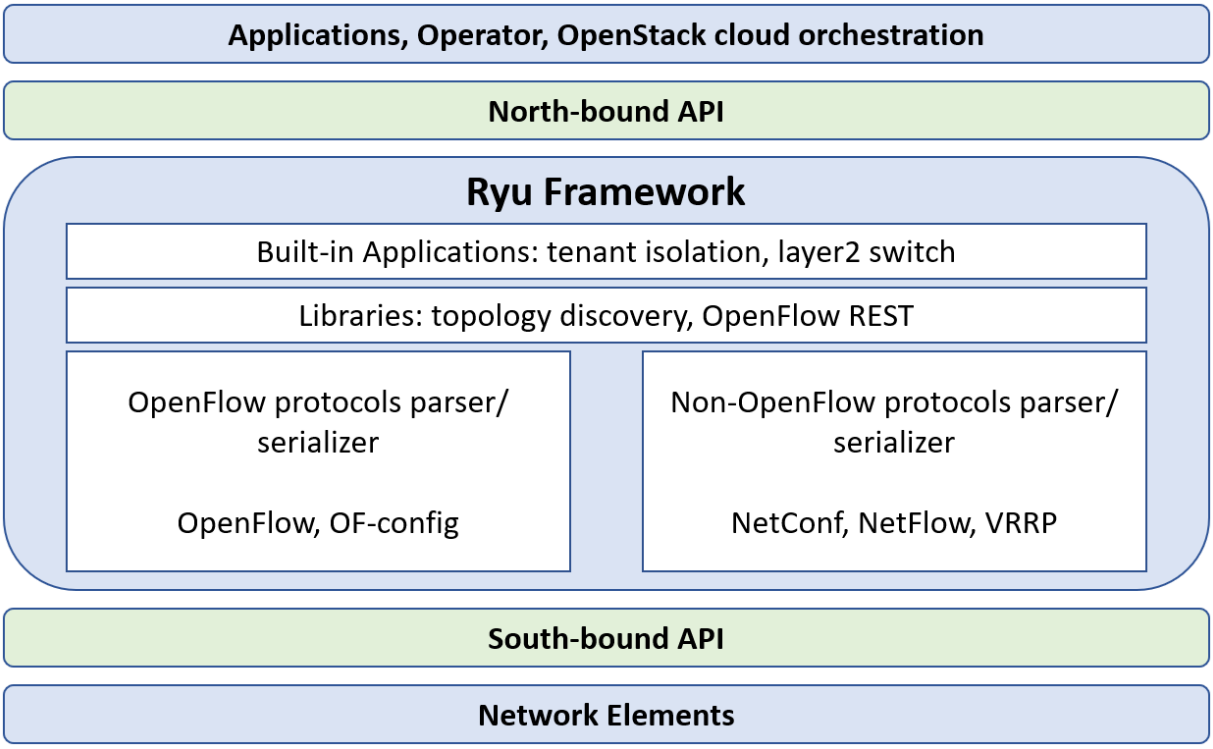


Figure 2. 5: Ryu controller architecture [26]

The main aim of Ryu was to become the standard network controller of cloud software like OpenStack. Ryu plugin was merged into the OpenStack Essex [40]. Ryu provides OpenStack tunnelling-based isolations and flat layer 2 networks regardless of the underlying network. Furthermore, the Ryu framework was developed to perform traffic monitoring and provide QoS in the network [41]. For user friendly management of flow tables on the OpenFlow switches from the Ryu controller, an application called, FlowManager was developed. By using this application, an operator can create, modify, and remove the flows as well as an operator can monitor the OpenFlow switches with the help of statistics.

The Ryu can be downloaded as a Python package as well as a source code repository [42]. The detailed information guides for installation, configuration, and templates for creating Ryu applications are well documented [43].

Other SDN controller such iSDX [44] and Ravana [45] were developed based on the Ryu controller. The iSDX controller was designed to solve the scalability of the Ryu controller by reducing the compilation time of policies and size of forwarding table. Whereas Ravana controller was developed as a fault tolerant SDN controller platform.

2.3 Software Switches

The traditional network switches build with network operating system consisting of the packet forwarding knowledge and routing intelligence were not suited for the functionality of Software-defined networks. The requirement for decoupling of control plane and data plane lead to the development of several new switches which were built to support the Southbound interface protocols. The basic functionality of these switches is to forward the data packets in the network as instructed by the SDN controller. When a new type of data packet arrives at the switchport, the switch not knowing how to process the received packet, asks this query to the SDN controller. The SDN controller processes the received packet and instructs the switch with controlling commands to forward or discard the received packet. Along with this, the SDN controller registers a set of forwarding flow rules on the switch stating the match-fields and action set for the requested query. Switches consists of some memory to store several sets of basic forwarding flow rules information, so that the SDN controller is not over-populated with the control packets for each packet.

Considering this functionality, various switches were developed to support Southbound interface protocols, mainly OpenFlow. Some of the developed OpenFlow switches are Open vSwitch [46], Pica8 [47], OpenFlow SoftSwitch 1.3 [48] , LINC [49] and Indigo Virtual Switch (IVS) [50]. First OpenFlow switch was developed in 2009 to support the OpenFlow protocol version 1.0. With the advancement of OpenFlow protocol, OpenFlow SoftSwitch 1.3 was designed to support the most stable version of OpenFlow protocol 1.3. It supported the components like OFdatapath, OFlib, Dpctl and OFprotocol. Pica8, an open switch software platform for hardware switching chips that includes the layer 2 and layer 3 stack and support for OpenFlow protocol. It provides an OpenFlow-compatible NOS that runs on a range of white box switch hardware. LINC is an open-source project to develop OpenFlow software switches. It was written in Erlang [51] and has modular architecture. LINC supports OpenFlow protocol versions 1.2, 1.3 and 1.4. Indigo Virtual Switch is an open-source OpenFlow virtual switch developed for high performance and minimal administration.

Among these software switches, Open vSwitch gained more popularity for its multi-server virtualisation deployments in large scale virtualised environments. It is designed to support multi-layer extensions and is distributed under the Apache 2.0 license. It supports vast variety of operating systems such as Windows and Linux distribution like Ubuntu, Debian, Fedora and openSUSE [52]. Open vSwitch also supports the FreeBSD and NetBSD operating systems. Open vSwitch utilizes OpenFlow protocol to support the efficient management, virtual switch configuration and deploy network services across a large-scale network.

Along the new releases of OpenFlow protocol versions, the Open vSwitch versions were developed to support the latest versions of OpenFlow. Currently, it supports all the versions of OpenFlow protocol. As well, it supports other standard management protocols such as SNMP and NETCONF. Additionally, Open vSwitch provides interfaces to monitoring protocols such as NetFlow. Open vSwitch is primary switch for virtual SDN network emulator environment like Mininet. In this thesis, Open vSwitch was utilised in the infrastructure layer due to its popularity and easy availability. The following table 2.2 lists the overview of OpenFlow protocol versions supported by each version of Open vSwitch.

Table 2. 2: Open vSwitch versions supporting OpenFlow protocol versions [53]

Open vSwitch version	OF 1.0	OF 1.1	OF 1.2	OF 1.3	OF 1.4	OF 1.5
1.9 and earlier	Yes	No	No	No	No	No
1.10 and 1.11	Yes	No	*	*	No	No
2.0 and 2.1	Yes	*	*	*	No	No
2.2	Yes	*	*	*	*	*
2.3 and 2.4	Yes	Yes	Yes	Yes	*	*
2.5, 2.6 and 2.7	Yes	Yes	Yes	Yes	*	*
2.8, 2.9, 2.10 and 2.11	Yes	Yes	Yes	Yes	Yes	*
2.12 and later	Yes	Yes	Yes	Yes	Yes	Yes

* - Supported but some features are missing

2.4 Virtual Emulated Environment

2.4.1 Mininet

Mininet [54] is a software developed specially to implement the software-defined networks. It is a network emulator designed to create the desired SDN network topology and analyse the performance of desired component. Mininet provides the ready to deploy network testbed for the SDN applications developers without the need to set up a physical environment.

Mininet created the realistic virtual network consisting of SDN controller, OpenFlow switches and host devices. All these network elements operate on kernel along with application code and everything functions on a single machine. It also includes a CLI that is topology aware and OpenFlow aware, which makes it easier for developers for debugging and analysing the complete network. Mininet supports various SDN controllers such as OpenFlow controller, OpenFlow reference controller, Open vSwitch's ovs-controller, NOX, and Ryu. A desired SDN controller can be selected for implementing the SDN network. Furthermore, Mininet provides the option to connect the network with the external SDN controller which might also be running on different machine. By default, Mininet runs Open vSwitch in OpenFlow mode, which requires an OpenFlow controller.

Mininet supports creation of many different topologies and running different tests on the network. Mininet is developed with in-built network topologies like linear and tree topology that can be selected while initiating the network. Also, the desired custom topologies can be created specifying the details of the network devices and links between them. With just simple command of ***sudo mn,*** the SDN network can be deployed consisting of two hosts, one Open vSwitch and one OpenFlow controller.

At initialisation of Mininet network, by default the endpoints (hosts) are created in their own individual namespaces whereas the switches and controller are created in the root namespace of the host machine sharing the attributes like process and directories of host machine. However, the switches can be initiated in their own namespace, by passing the ***--innamespace*** argument while starting the Mininet. This option does provide an example of how to isolate different switches. In terms of functionality this results in, the switches will communicate with the controller through a separately bridged control connection instead of using a loopback interface. Mininet also provides the access to the CLI of these network elements created in different namespaces to get the network view from respective network element's point of view.

Mininet also gives the flexibility to integrate Python API, thereby paving the way for creating and experimenting with networks. Mininet is majorly used as a learning tool to test, experiment, and study the software-defined networks and it is preferable because of its quick initialisation, simplicity, and ability to create customisable topologies. Mininet has been shown to be a valuable tool in the research communities of software-defined networking. Due to the prominent use of Mininet in SDN, it was selected for the implementation of this thesis.

Mininet is available to be installed as a packet or from the source code. The Mininet community has created a VM image pre-installed with Mininet over the Ubuntu operating system [55]. The Mininet VM image supports all varieties of hypervisors like VirtualBox, KVM, Qemu, Microsoft Hyper-V, VMware Fusion, and VMware Workstation Player. Because of the single server implementation of Mininet, it faced performance challenges on large-scale networks. To resolve this problem, Maxinet [56] an extension of Mininet and Distributed OpenFlow Testbed (DOT) [57] was developed.

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.

Disadvantages of Mininet:

- Deployed network elements cannot exceed the CPU and bandwidth available on host machine.
- Single server implementation gave rise to the performance problems.
- All Mininet hosts share the host file system and PID space, a careful assessment of running daemons is required for process of debugging.

2.4.2 GNS3

GNS3 (Graphical Network Simulation 3) [58] 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 2.2.34) 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 perform simulations about them. Since it runs real images, it is necessary to have the images of the devices to be simulated. GNS3 offers a marketplace [59] on its official website which is basically a platform for different appliances that are pre-configured images and can be used to emulate a network device. The Open vSwitch appliance, Cisco routers appliances and Ubuntu Desktop appliance were installed from this marketplace for the implementation of use cases in this thesis.

GNS3 consists of two software components:

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

The GNS3-all-in-one is the graphical user interface (GUI) where the topologies are created. When the topologies are created, the network 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.

Both of these GNS3 software components are easy to install and configure [60]. For this Thesis, VirtualBox was preferred to host the devices on the GNS3 VM. GNS3 also provides the option to utilise other virtual machines installed within the VirtualBox or other hypervisors. A virtual machine that is separately installed in hypervisor can be imported inside the GNS3 software and utilised as a network device, server or just as an Ubuntu desktop client. For this Thesis, SDN controllers were installed on different virtual machines in VirtualBox and imported inside the GNS3 software.

Advantages of GNS3:

- Implements easy to access network topology.
- No limitation on number of devices or complexity of topology.
- Supports multiple vendors network devices.
- Supports multiple deployment options: local machine, hypervisor, or remote server.
- Pre-configured and optimised appliances available to simplify deployment.
- Availability of in-built software packages such as Wireshark and Putty.
- Supports sharing of implemented network including the device configuration.
- Large and active community

Disadvantages of GNS3:

- Limitation of CPU and memory resources provided by the host machine.
- The software images of the network devices such as Cisco routers are required to be purchased and downloaded by the user.
- GNS3 can be affected by PC's setup and limitations such as firewall and security settings, company laptop policies etc.

2.5 OpenFlow Protocol

The most used Southbound interface protocol in SDN architecture is OpenFlow protocol. OpenFlow protocol was designed by Nick McKeown's team from Stanford University in 2007 [4] with the basic idea to centrally manage the global policies, a *flow-based network* was proposed which eventually gained the popularity in the software-defined networks. After the establishment of Open Network Foundation (ONF) for standardising the emerging network technologies, OpenFlow protocol went through rapid growth and development.

Over the years new features were developed to support the different network services. Six different versions of protocol from version 1.0 till version 1.5 and later updated versions after few modifications were released. The first version [11] of OpenFlow consists of a single flow table and an OpenFlow secure channel to communicate with the SDN controller. The flow table keeps the flow entries that are suggested by the SDN controller. To increase the scalability and performance of protocol, some additional features like multiple flow tables, group tables, pipeline processing along with support for VLAN and MPLS service were added to the second major release [61] of the protocol. The next 1.2 version [62] released extended the support for IPv6 addressing scheme and additional matching capabilities were added. The OpenFlow version 1.3 [63] introduced new features like meter table to support QoS, extended support for multiple controllers for better load balancing and support for IPv6 header extension. The next 1.4 version [64] released added bundle mechanism and support for table synchronization. Also, the default TCP port of protocol was changed from 6633 to 6653. The last major release of protocol, version 1.5 [65], added egress table and modified pipeline to be aware of packet type. The modified version of OpenFlow protocol 1.5.1 added the new error in flow-mod type for unmatched meter ID [5]. The latest version 1.6 was released in 2016, but it is only accessible to ONF's members. The OpenFlow versions 1.3 and 1.4 are used widely as they are stable implementation.

The following table displays the major features released in each version of protocol.

Table 2. 3: Major modifications in OpenFlow protocol versions

Version 1.0	Version 1.1	Version 1.2	Version 1.3	Version 1.4	Version 1.5
Single flow table entry	Multiple flow table	Support for IPv6	Support for IPv6 extension header	Add in bundle mechanism	Egress table
Single Controller	Group table	Extensible match support	Flow meter	Support for Table synchronization	Packet type aware Pipeline
Support for IPv4	Virtual ports	Simplified behaviour of flow-mod request	Support for Table miss	Change default TCP port to 6653	Extensible flow entry statistics
	Pipeline Processing		On demand flow counters		
	Support for VLAN and MPLS				

The OpenFlow protocol allows the SDN controller to control the OpenFlow switch networks without the requirement to disclose the source code of switches. OpenFlow functions on the different components. The main components of the OpenFlow protocol are Flow table, Group table, Meter table and OpenFlow channel. OpenFlow channels acts as interfaces to connect the OpenFlow switches with the SDN controllers. Different channels are used to set up the connection with different SDN controllers connected in the control plane of SDN. The flow table and group table perform packet lookups and forwarding. The flow table consists of flow entries and group table consists of group entries. To define the per-flow meters, the meter table consists of meter entries which help set up the required QoS policies.

The following figure displays the main components utilised by OpenFlow protocol.

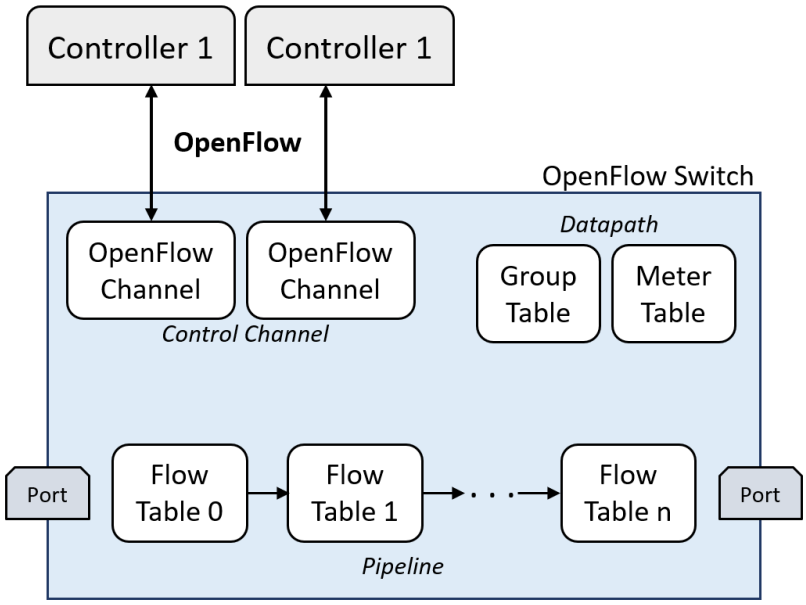


Figure 2. 6: Components of OpenFlow protocol [5]

When the data packet arrives at the switchport of OpenFlow switch, it reviews the packet header and checks the flow table for matching flow entries. An OpenFlow switch should at least have one flow table but can support of multiple flow tables. The flow table consists of multiple flow entries and can have maximum of 65535 flow entries in a single flow table. All the flow tables are numbered, starting from flow table 0 and are arranged in the pipeline. When the packet is checked for flow entry, the match-fields such as packet header, ingress port, source and destination addresses are checked. If the flow entry of the match-fields is found, as per the action set instructions, action is performed on the packet. The action set can consist of actions such as, to forward a packet from a specific port and also packets can be redirected to other flow tables if required. The priority value of each flow rule is set to provide the preference to specific service. In case of similar match-fields detected for multiple flow rules, the one with the highest priority flow rule will be executed. The pipeline handles the packet flow within the OpenFlow switches. If no entry is found in any of the configured flow tables, the packet is forwarded to the SDN controller through control channel. SDN controller informs the OpenFlow switch to take appropriate action on the packet and install the corresponding flow entry in the flow table [63].

The group table consists of set of actions for flooding and more complex forwarding semantics such as multi-path, fast reroute and link aggregation. An OpenFlow switch can have maximum of 1024 group tables. The group table consists of group entries and each group entry consists of action buckets. Four types of group tables exist: *All, Select, Indirect* and *Fast failover*. *All* type of group tables executes all the action buckets in the group. *Select* type of group tables execute the any one action bucket from the group depending on the round-robin selection method. The *Indirect* type of group execute the one defined action bucket from the group. And *Fast failover* type of group executes the first live action bucket in the group [5]. The meter table consists of meter entries which enables OpenFlow to implement QoS functionalities like rate limiting. A meter table consists of meter entry components such as Meter ID, Meter bands and counters. A meter table can have multiple meter bands in itself, which are used to specify the rate at which the packet should be processed. Counters are incremented once the packet is processed by the meter table. Queues are utilised to schedule the forwarding of packets as per the priority set in QoS policies. Multiple flow entries can leverage on the same meter table and a packet can be processed by the multiple meter tables [65].

OpenFlow protocol supports three types of messages: ***Controller-to-switch, Asynchronous and Symmetric***. The *Controller-to-switch* message type is sent by the SDN controller to OpenFlow switch consisting of controlling commands. This message type includes Handshake, Switch configuration, flow table configuration, modify state messages, multipart messages, packet-out, barrier messages, role request, bundle messages and set asynchronous configuration messages. All these message types are used by the SDN controller to request information from the switch or to implement a set of rules on the switch.

The *Asynchronous* message type is utilised by the OpenFlow switch to communicate with the SDN controller. This message type includes Packet-In, Flow removed, Port status, Controller role status, Table status, Request forward messages and Controller status messages. The *Symmetric* message type can be originated either by the SDN controller or even OpenFlow switch. This message type includes Hello, Echo request, Echo reply, Error message and Experimenter message. This type of message is sent without solicitation. The following figure displays the initial message types exchanged between SDN controller and OpenFlow switch while and after setting up the connection [5].

Source	Destination	Protocol	Info
192.168.122.209	192.168.0.114	OpenFlow	Type: OFPT_HELLO
192.168.0.114	192.168.122.209	OpenFlow	Type: OFPT_FEATURES_REQUEST
192.168.122.209	192.168.0.114	OpenFlow	Type: OFPT_FEATURES_REPLY
192.168.0.114	192.168.122.209	OpenFlow	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
192.168.122.209	192.168.0.114	OpenFlow	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
192.168.0.114	192.168.122.209	OpenFlow	Type: OFPT_GET_CONFIG_REQUEST
192.168.122.209	192.168.0.114	OpenFlow	Type: OFPT_BARRIER_REPLY
192.168.122.209	192.168.0.114	OpenFlow	Type: OFPT_GET_CONFIG_REPLY
192.168.0.114	192.168.122.209	OpenFlow	Type: OFPT_MULTIPART_REQUEST, OFPMP_METER_FEATURES
192.168.122.209	192.168.0.114	OpenFlow	Type: OFPT_MULTIPART_REPLY, OFPMP_METER_FEATURES
192.168.0.114	192.168.122.209	OpenFlow	Type: OFPT_MULTIPART_REQUEST, OFPMP_DESC
192.168.122.209	192.168.0.114	OpenFlow	Type: OFPT_MULTIPART_REPLY, OFPMP_DESC
192.168.0.114	192.168.122.209	OpenFlow	Type: OFPT_ROLE_REQUEST
192.168.122.209	192.168.0.114	OpenFlow	Type: OFPT_ROLE_REPLY

Figure 2. 7: OpenFlow Message types

The first packet is initiated by the OpenFlow switch (192.168.122.209) to the SDN controller (192.168.0.114) with the Symmetric message type *OFPT_HELLO*. This is the first handshake message containing OpenFlow header and is initiated by OpenFlow switch. SDN controller replies with the *OFPT_HELLO* and *OFPT_FEATURES_REQUEST* message, which consists of some attributed request like DPID, number of tables and Maximum packets buffering capacity. Datapath unique ID (DPID) identifies a data path in the OpenFlow topology and consists of lower 48 bits of MAC address and upper 16 bits defined by implementer. The Open-Flow switch replies with *OFPT_FEATURES_REPLY* message consisting of the number of tables supported by the switch, each of which can have a different set of supported match fields, action sets, and number of entries. Then controller makes various other requests to get complete information about the connected OpenFlow switch via *OFPT_MULTIPART_REQUEST* message types and the OpenFlow switch replies with *OFPT_MULTIPART_REPLY* message types. As seen in the above figure 2.7, the controller requests for port description of switch via *OFPMP_PORT_DESC* message. The OpenFlow switch replies with *OFPMP_PORT_DESC* message consisting of port information such as port number, name, state, MAC address, etc. of all switchports. Similarly, other information is communicated between SDN controller and switch through OpenFlow protocol.

The major vendors such as Cisco, Juniper, Big Switch Networks, VMware, IBM, Versa Networks, HPE, Google have built their devices to support OpenFlow protocol. Other Southbound interface protocols like OpFlex [66], which distributes the complexity of controlling the network devices through policy-based model. However, its functionality to provide service was different in comparison with OpenFlow, it answers the scalability problem by distributing few control parameters to the forwarding devices. Forwarding and control element separation (ForCES) [67], which was developed before the popularity of SDN provided the framework for separation of control and forwarding elements that are in the same physical device, to manage the network without the re-quirement of centralised controller. Programming Protocol-independent Packet Processors (P4) [68], which was a programming language designed to control the network devices. P4 was considered as successor of OpenFlow protocol, by stating the differences between them, it was clear that how both can be combined to work for the enhancement of networks [69].

3 Requirements Analysis

With the advancement of software-defined networks, several network components were developed by different communities. The main component of the SDN architecture, SDN controller gained several production competitors for its development. Even though these controllers support the similar network services and try to overcome the same difficulties, many differences are observed in the development and functionality of the component. Few of the open-source SDN controllers well-known amongst researchers, developers, and commercial users gained more popularity because of being readily available, and flexible when it comes to self-developed changes and upgrades. These controllers gained more popularity also from the other communities for the software upgradation and modification from their base versions. Various SDN controllers were developed based on the framework designed by different controllers. These evolved versions of controllers were found to be rich in features and functionality along with solving the problems faced by the base controller. Other components of SDN such as software switches and their supported protocols also received several opponents for their development. Before implementing all these components of SDN in real-world networks, the detailed performance analysis of each component is crucial. To evaluate the functionality of SDN, various testbeds are available with promising real-world network resemblance in emulation environment. All these different aspects of SDN were researched over the course of this thesis.

3.1 General Objectives

The aim of this thesis was to study the functionalities of software-defined networks and practically develop the real-world resembling environment in the network emulator environment to evaluate the performance of different SDN components.

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

- Select the appropriate network emulator of implementation of the desired software-defined network.
- Build a suitable network with different network devices in the network emulator software.
- Manage different services and network configurations of network devices from the SDN controller in an emulated environment.
- Create and distribute the network configurations for network services through possible different methods.
- Develop a rationale and setup an IPv4 and IPv6 scheme for the network.
- Create isolated VPN services that have different requirements.
- Proof and validation of functioning failover mechanisms of SDN controller and other aspects of network to improve resilience.
- Integrate SDN network with legacy networks in a way to demonstrate the migration of legacy networks to SDN.
- Evaluate advantages and disadvantages of network with SDN controller over traditional network.

Throughout this thesis, the following research questions are answered:

- Which open-source SDN controllers provide the flexibility for network management and the difficulties they promise to overcome?
- What are the alternative configuration methods available with the goal of finding the best possible method to configure and manage the network through the SDN Controller?
- Which are the different architectures designed for SDN implementation and what objective they provide?
- What are the other components required in the software-defined networks such as network devices and supported protocols?

The research was carried out to study the open-source SDN controllers with respect to their base architecture, functionality, features, and network services they support and their respective applications availability.

3.2 Workflow

Workflow for this Thesis is as follows:

- Research about different open-source SDN controllers, their architectures and challenges faced by them, as discussed in chapter 2.
- Research about the network emulation software.
- Installing and testing of different SDN controllers in different emulated environments, as discussed in chapter 4.
- Configuration of these SDN controllers.
- Studying the different SDN applications implemented for network services.
- Installing and activating the features required for the services and applications running on the SDN controller.
- Installing different open-source network devices to run inside the emulated environment.
- Setting up the testbed in the emulation software with these networking devices and the SDN controller.
- Creating and implementing different network topologies.
- Creating use cases to test the performance of SDN controller in the emulated environment.
- Implementing and testing these use cases in different emulated environments.
- Evaluating the outcomes of these use cases.
- Document the difficulties and errors found while installation and debugging the connectivity between the different network devices and SDN controller.

3.3 Previous Work

Through the research and study about different aspects of SDN such as popular SDN controllers and software switches and their implementation in virtual environments, the following points were found:

- Various research surveys [23] have been carried out to study the different aspects of SDN such as the design architecture of SDN controller, different framework they are developed on, challenges they face and their countermeasures.
- Currently various open-source SDN controllers are available to be implemented and tested in the emulated environment. To name few which are popular amongst researchers and developers are, OpenDaylight (ODL), Open Network Operating System (ONOS) and Ryu.
- Outcome of the research carried out during this thesis, presented that the mentioned three SDN controllers have some similarities and more differences in terms of functionality, design architectures, development platform and support for different network services.
- Some projects were developed to study the SDN application of a certain controllers for testing the particular network service, and for these research questions, specific testbeds were created. These testbeds were implemented on local servers or different virtual machines each acting as a single network component of the SDN network. This approach of experimenting consumed a large amount of machine resources and could not practically test the performance of SDN network in terms of scalability and high availability.
- Contrastingly, a complete virtual testbed, specially designed to implement SDN networks and analyse the different features of SDN, was heavily utilised to experiment with the SDN networks. Mininet, the virtual network emulator used to deploy quick SDN network consisting of SDN controller, OpenFlow switches and endpoints provides supports to varieties of SDN components. Some of which are in-built, if not, an option to program that component or attach it externally is available. Every component deployed in the Mininet, is the virtual replica of the network components.
- The network emulation environment which deploys the real images of the network components would provide the ideal platform for testing the performance of SDN networks. The GNS3 application, which helps create the real-world resembling networks implemented on the single machine could bring the evaluation results closer to the real-world scenarios. Not enough research and quite few experimental set-ups with this network emulator were discovered.

4 Realization

After researching about various SDN controllers, few suitable SDN controllers were selected for building a 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 packages were selected, namely, Mininet and GNS3.

4.1 Installation of Testbed

For setting up the testbed for this thesis work, all the major components were installed on the Oracle VM VirtualBox virtualization 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 can create 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 deployed devices need to be hosted and run by a server process, for this GNS3 VM instance was used. All SDN controllers have their complete installation in respective virtual machines.

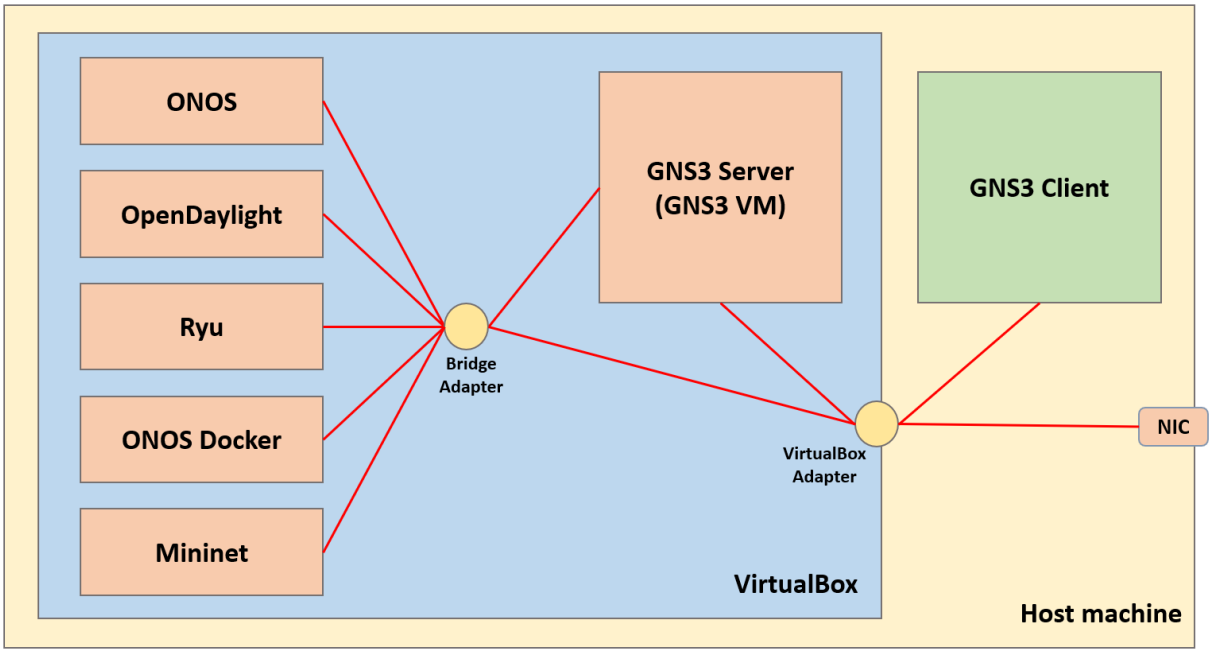


Figure 4. 1: Different virtual machine instances in VirtualBox

Topologies created inside the GNS3 and Mininet were able to connect to the outside SDN controller using OpenFlow protocol. SDN controllers were 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. All these SDN controllers and emulation software were set up using standard scripts and no special optimization was performed.

For experimenting the use cases with these SDN controllers few different virtual machines were created with the complete experiment setup. For example, in the use case for testing the network with multiple controllers, deploying the multiple controllers in the separate virtual machines would require a large number of resources. And for this reason, multiple controllers were installed as Docker containers and hence, a separate virtual machine named, *ONOS Docker* was created for this use case.

For the tasks associated with this master thesis, following is the list of software and machine configurations used.

Table 4. 1: Software and their versions utilised for implementation

Category	Software	Version
Host Machine Operating system	Windows 10 Home	21H2 OS build 22000.856
Host Machine Processor	Intel(R) Core(TM) i7-1065G7 8 CPUs @ 1.30GHz 1.50 GHz	
Host Machine RAM	16.0 GB	
Host Machine Graphics	Intel(R) Iris(R) Plus	
Software Switch	Open vSwitch	2.12.3
GNS 3 - Client part	GNS3-all-in-one software	2.2.33.1
Traffic Analyzer	Wireshark	3.6.6
Virtual Machines Hypervisor Manager	Oracle Virtual Box	6.1.36 r152435 (Qt5.6.2)
1. GNS3 VM		
a. GNS3 VM	Server	2.2.33.1
b. GNS3 VM Operating System	Ubuntu	20.04 Desktop
c. GNS3 VM RAM, Memory, CPUs	8.0 GB, 40 GB, 3	
2. Mininet-VM		
a. Mininet		2.3.0
b. Mininet-VM Operating System	Ubuntu	20.04 Desktop
c. Mininet-VM RAM, Memory, CPUs	2.0 GB, 10 GB, 2	
3. ONOS-VM		
a. ONOS	X-Wing (LTS)	2.7.0
b. ONOS-VM Operating System	Ubuntu	20.04 Desktop
c. ONOS-VM RAM, Memory, CPUs	4.0 GB, 15 GB, 3	
4. ODL-VM		
a. ODL	Sulfur	SR1
b. ODL-VM Operating System	Ubuntu	20.04 Desktop
c. ODL-VM RAM, Memory, CPUs	2.0 GB, 10GB, 3	
5. Ryu-VM		
a. Ryu		4.34
b. Ryu-VM Operating System	Ubuntu	20.04 Desktop
c. Ryu-VM RAM, Memory, CPUs	2.0 GB, 10 GB, 3	
6. ONOS Docker-VM		
a. ONOS	Peacock (LTS)	1.15.0
b. ONOS Docker-VM Operating System	Ubuntu	20.04 Desktop
c. ONOS-VM RAM, Memory, CPUs	3.0 GB, 20 GB, 3	

4.2 Implementation with GNS3

After complete installation and necessary configuration of the components required for this thesis work, a topology with four Open vSwitches and three routers was created in the GNS3 application. For testing the reliability of the network, the routers are deployed at same node as well as far end nodes of the topology.

In GNS 3, to connect any network device to the Internet or SDN controller, which was installed and running outside the GNS 3 software, a NAT interface was used. A layer 2 switch (Switch1 in Figure 4.2) was used so that all devices can be directly connected to the NAT interface with just single interface. SDN controller used here was ONOS controller.

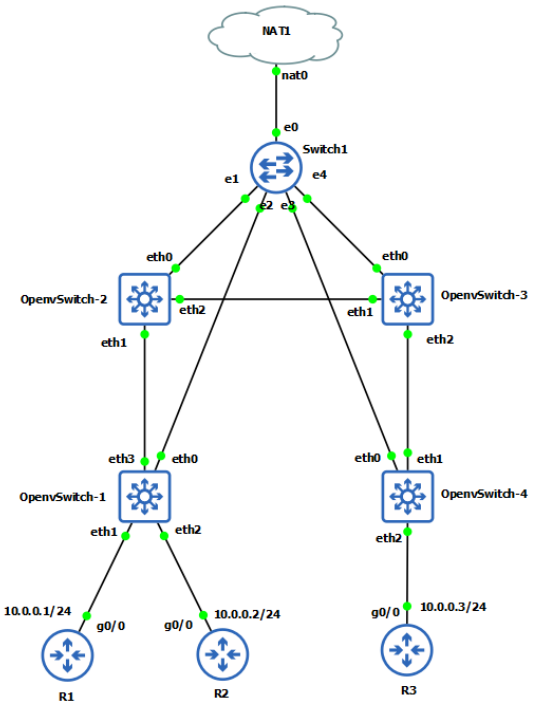


Figure 4. 2: Topology created in the GNS3 with four Open vSwitches and three Routers

The eth0 interface of Open vSwitch is the management interface which is used for communication with SDN controller and therefore, the eth0 interface needs to have an IP address. Here eth0 interface was configured to have an IP address from the DHCP pool of the GNS 3 Server VM. Before booting up the Open vSwitches, the configuration file needs to be edited to request the address from the DHCP pool.

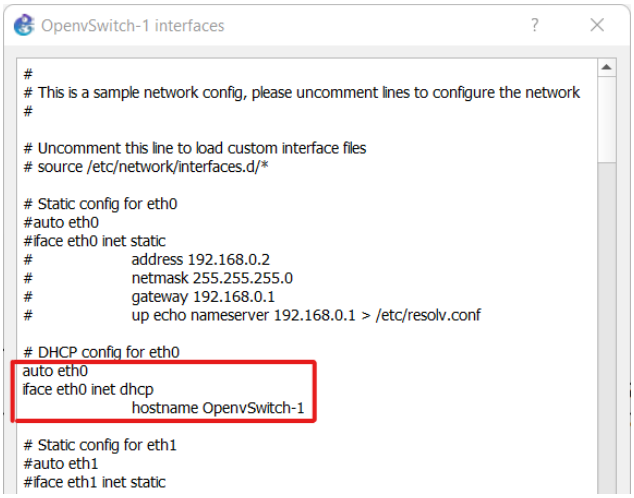


Figure 4. 3: Open vSwitch eth0 management interface configuration

The following figure displays the simplified setup of the implemented topology. The *eth0* interfaces of Open vSwitches (OVS) received the IP addresses from the DHCP pool of GNS3 server. These interfaces are connected to ONOS controller via switch and NAT interface.

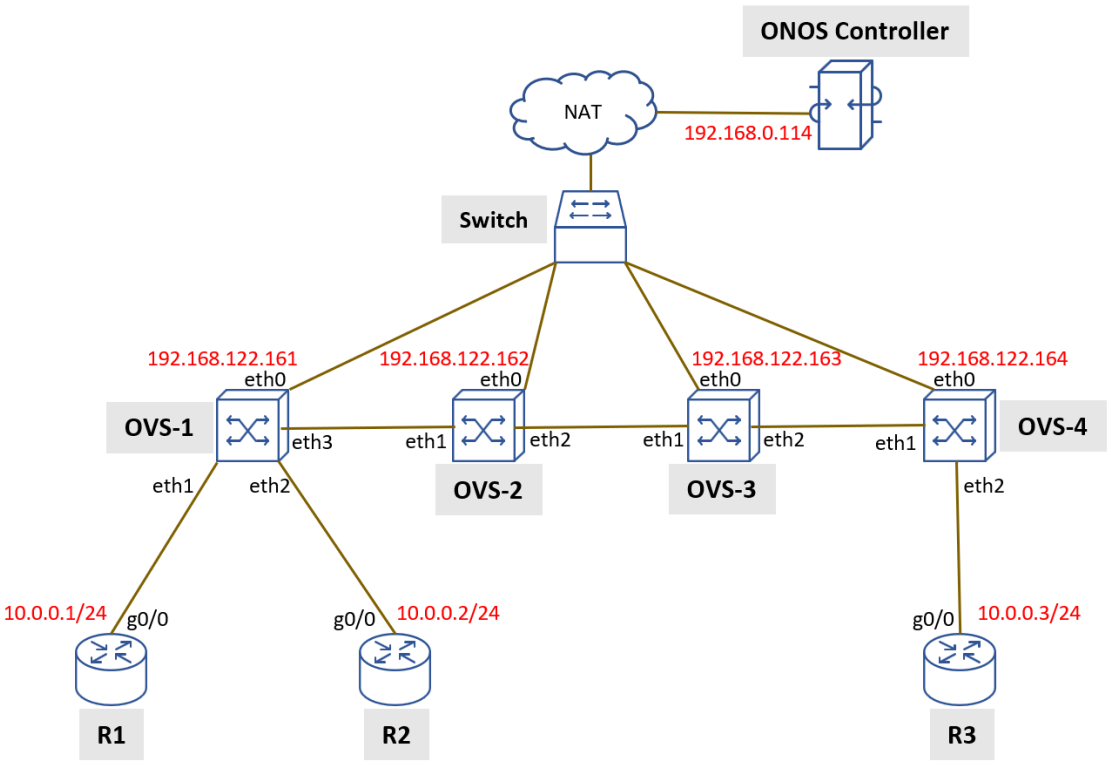


Figure 4. 4: Detailed connection diagram of created topology

Once the Open vSwitch is started and ready to establish connection with the SDN controller, it needs to be confirmed that all the other interfaces of the Open vSwitch are configured under the same Bridge interface. Ideally all the other interfaces of the Open vSwitch are configured under the Bridge *br0* interface, if not, these needs to be manually configured.

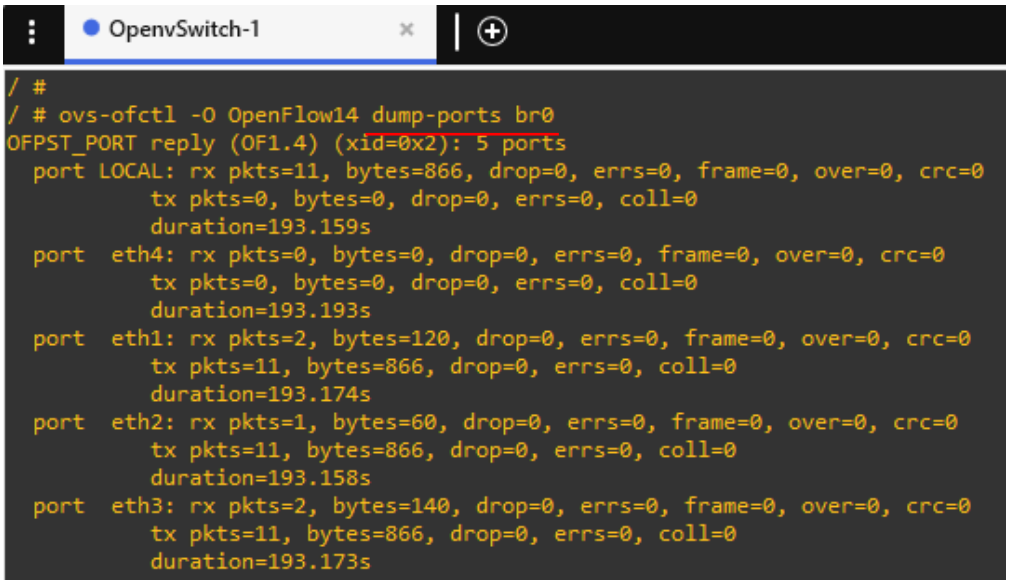


Figure 4. 5: List of ports configured under Bridge br0 on Open vSwitch-1

Bridge br0 is the logical 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, 192.168.0.114) and port number on which SDN controller was listening to OpenFlow protocol needs to be specified (here, 6653).

Before connecting the Open vSwitches to the SDN controller, OpenFlow protocol version (here OpenFlow protocol version 1.4) also needs to be specified along with the bridge interface being used. In this thesis, OpenFlow protocol version 1.3 and also 1.4 were utilised, which were the stable versions and supported all functionalities of implemented use cases.

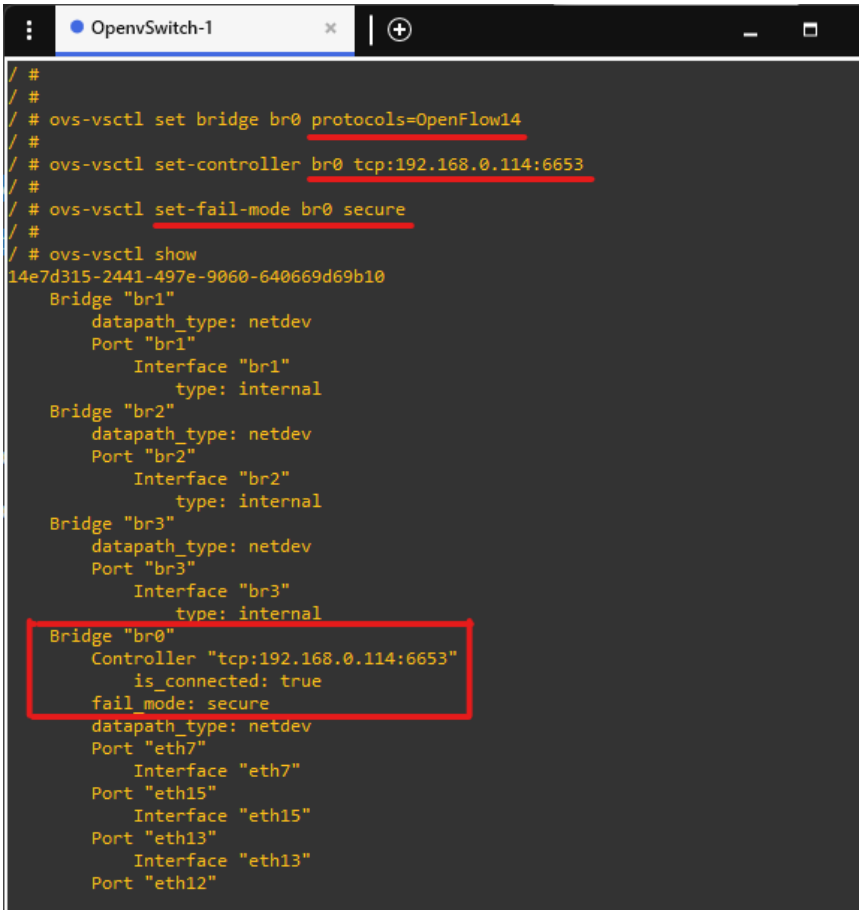


Figure 4. 6: Setting up Open vSwitch to communicate with the SDN controller

As seen in Figure 4.6, 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, two options can be utilized first, *ONOS CLI* and other *ONOS GUI*.

The ONOS CLI is an extension of Karaf's CLI. As a result, it is capable of leveraging features such as programmatic extensibility, the ability to load and unload bundles and SSH access. To access the ONOS CLI from the local machine itself, following command is used.

/opt/onos/bin/onos

To access the ONOS CLI from the host or remote machine, SSH keys need to be shared between the machines. The ONOS CLI listens on port number 8101. To access the ONOS CLI from the remote machine, following default login credentials are used.

ssh -p 8101 onos@192.168.0.114

Password: ***rocks***

Or to access the Karaf's CLI, following default login credentials are used.

ssh -p 8101 karaf@192.168.0.114

Password: ***karaf***

```
onos@root > devices                                                    18:47:33
id=of:00004e6d3cf34349, available=true, local-status=connected 9m13s ago, role=MASTER,
type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.3, serial=None, chassis=4e6d3cf
34349, driver=ovs, channelId=192.168.0.103:64636, datapathDescription=None, managementA
ddress=192.168.0.103, protocol=OF_14
id=of:00005a0521b2b145, available=true, local-status=connected 1m38s ago, role=MASTER,
type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.3, serial=None, chassis=5a0521b
2b145, driver=ovs, channelId=192.168.0.103:64716, datapathDescription=None, managementA
ddress=192.168.0.103, protocol=OF_14
id=of:0000a66b420f9043, available=true, local-status=connected 1m40s ago, role=MASTER,
type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.3, serial=None, chassis=a66b420
f9043, driver=ovs, channelId=192.168.0.103:64714, datapathDescription=None, managementA
ddress=192.168.0.103, protocol=OF_14
id=of:0000bedcd3680a49, available=true, local-status=connected 1m39s ago, role=MASTER,
type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.3, serial=None, chassis=bedcd36
80a49, driver=ovs, channelId=192.168.0.103:64715, datapathDescription=None, managementA
ddress=192.168.0.103, protocol=OF_14
```

Figure 4. 7: List of devices (Open vSwitches) connected to the ONOS controller from ONOS CLI

On the ONOS CLI, ***devices*** command is used to list out the information about the connected devices (i.e., Open vSwitches) to the ONOS controller. The list presents the detailed information of the devices like ID or label of the device, availability of the device, role of the ONOS controller (Master, Standby, None) for that device, protocol used at the Southbound interface, etc. In the above figure it is observed that four devices were connected to the ONOS controller using the OpenFlow protocol version 1.4.

Alternatively, ONOS GUI can also be used to confirm the connection from the controller side. The ***onos-web-gui*** feature application must be installed and activated on the ONOS. On the browser, the ONOS GUI listens on port number 8181. The base URL is /onos/ui; for example, to access the GUI on localhost, use: http://localhost:8181/onos/ui or http://192.168.0.114:8181/onos/ui. The default username and password for accessing the ONOS GUI are ***onos*** and ***rocks*** respectively.

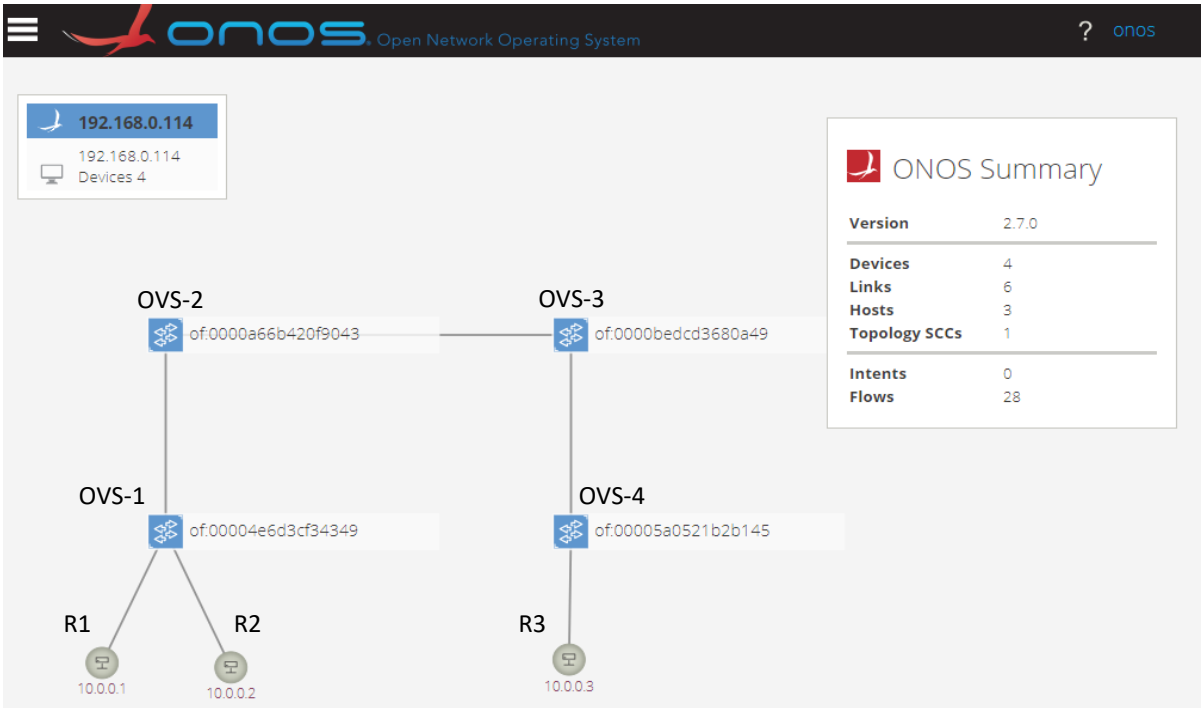


Figure 4. 8: Topology view of the created network from the ONOS GUI

A Wireshark application was listening on the link between the ONOS controller and one of the Open vSwitch to analyse the packets transferred between them. Southbound interface OpenFlow protocol was also captured on this link and figure 4.6 shows the Wireshark capture of the same. In the following Wireshark snippet, one can observe the initial OpenFlow packets exchanged between the Open vSwitch and ONOS controller for connection establishment as described in Chapter 2.

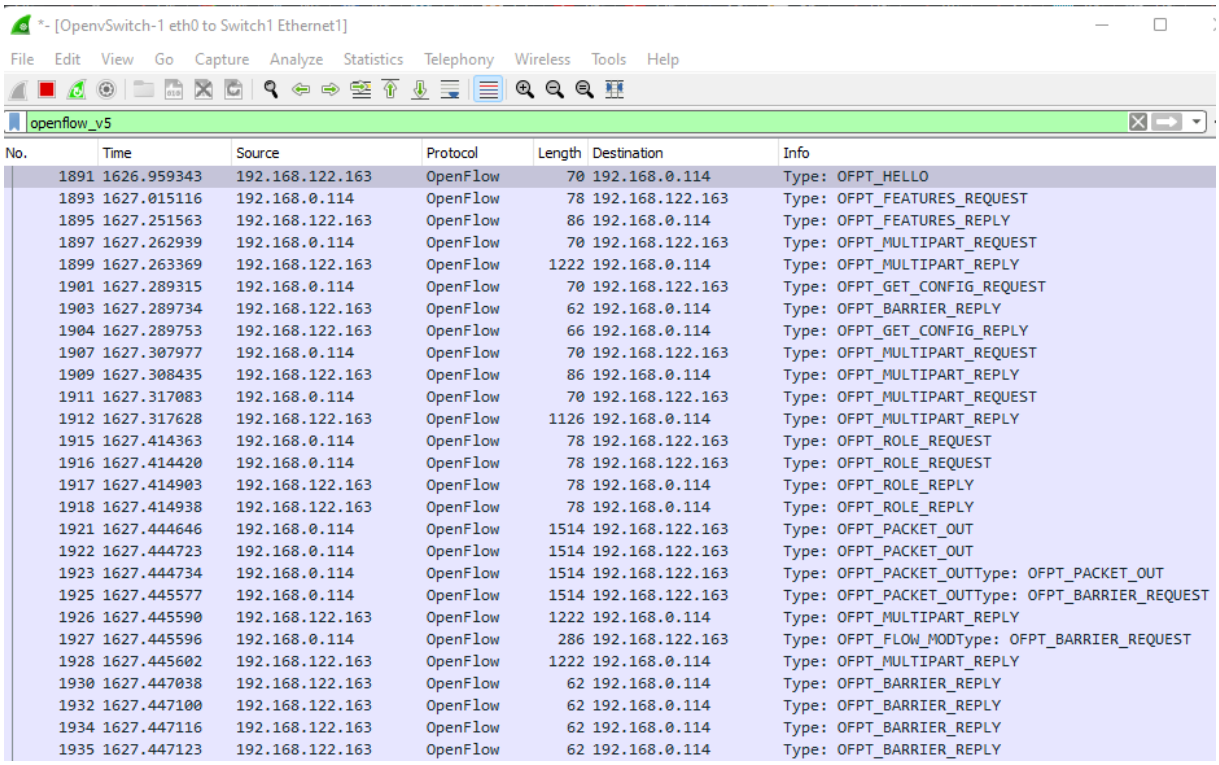


Figure 4. 9: OpenFlow protocol packets captured on the Wireshark between Open vSwitch and ONOS controller

Figure 4.10 displays the Wireshark packet snippet of OpenFlow Hello packet sent from Open vSwitch to the ONOS Controller for establishing the connection. In the following packet, one can confirm the OpenFlow protocol version 1.4 being used for communication between the ONOS controller and Open vSwitches.

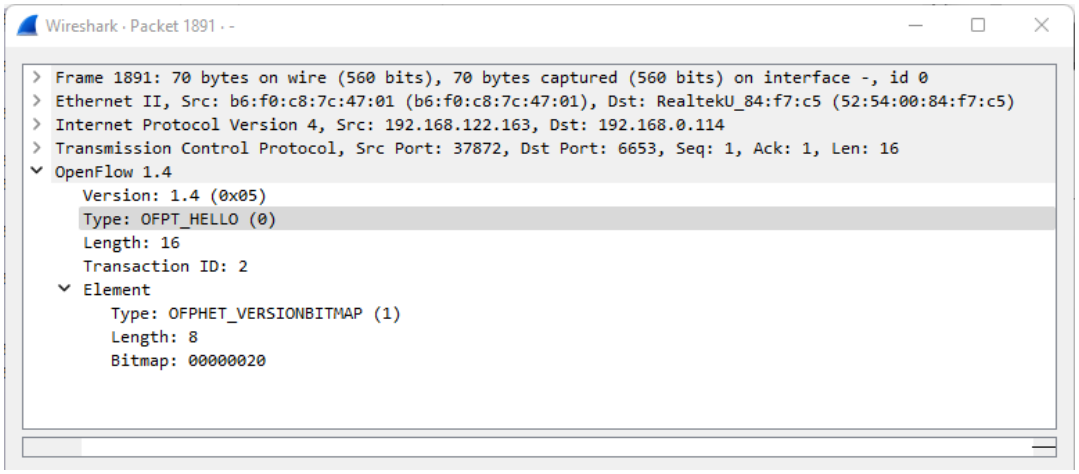


Figure 4. 10: OpenFlow handshake message- Hello packet captured on the Wireshark between Open vSwitch and ONOS controller

4.2.1 ONOS GUI

The ONOS GUI is a single-page web-application, providing a visual interface to the ONOS controller. The ONUS GUI provides a visual and thus more intuitive way to access relevant information about the ONOS controller and the topology connected to it. Information such as applications installed on the controller, number of devices connected, number of hosts, port numbers used in the topology, number of packets transferred between the links, and much more information is easily accessible through ONOS GUI. ONOS GUI listens on port number 8181 of the installed machine and can be accessed from the base URL: http://XX.YY.WW.ZZ:8181/onos/ui/index.html. The default username and password for accessing the ONOS GUI are ***onos*** and ***rocks*** respectively.

Following two features are required to be installed and activated on the ONOS for accessing the ONOS GUI.

App-ID: org.onosproject.gui *Feature: onos-web-gui*

App-ID: org.onosproject.gui2 *Feature: onos-web-gui2*

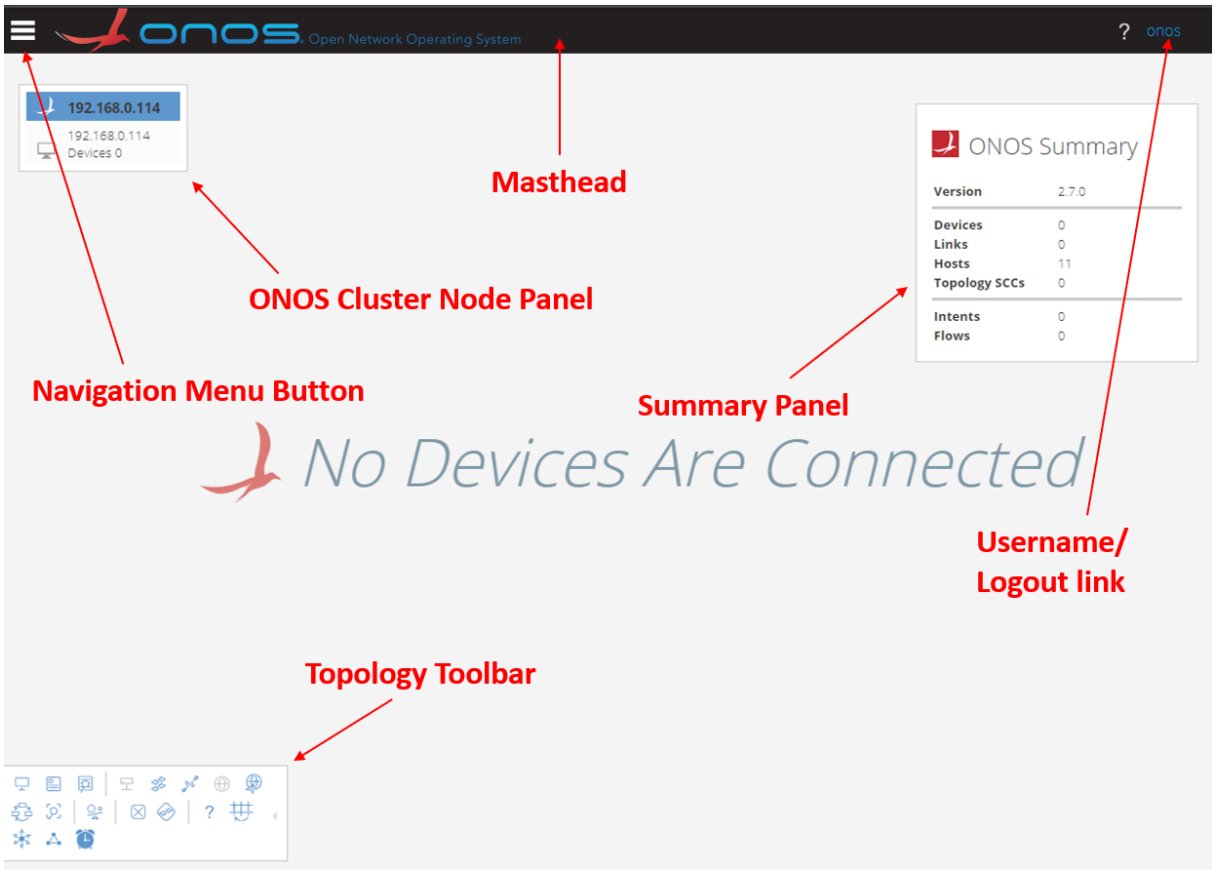


Figure 4. 11: Different components of the ONOS GUI

The ONOS Cluster Node Panel indicates the cluster members (controller instances) in the cluster. The Summary Panel gives a summary of properties of the network topology. The Topology Toolbar offers push-button / toggle-button actions that interact with the topology view. The Navigation Menu button provides access points to the various components of the network, from where one can view and configure various aspects of the network element.

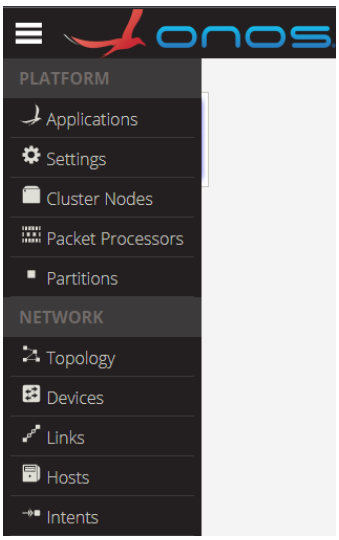


Figure 4. 12: Different Panels available to navigate on ONOS GUI

Table 4. 2: Description of different panels of ONOS GUI [70]

View	Description
Platform Category	
Applications	The Application view provides a listing of applications installed, as well as interaction with them on the network.
Settings	The Settings view provides information about all configurable settings of the components.
Cluster Nodes	The Cluster Nodes view provides a top-level listing of all the cluster nodes, (ONOS instances), in the network.
Packet Processors	The Packet Processors view shows the currently configured components that participate in the processing of packets sent to the controller.
Partitions	The Partitions view shows information about how the cluster partitions are configured.
Network Category	
Topology	The Topology view provides an interactive visualization of the network topology, including an indication of which devices (switches) are mastered by each ONOS controller instance.
Devices	The Device view provides a top-level listing of the devices in the network.
Flows	The Flow view provides a top-level listing of all flows for a selected device. (Note that this view would be able to access under the Device view.)
Ports	The Port view provides a top-level listing of all ports for a selected device. (Note that this view would be able to access under the Device view.)
Groups	The Group view provides a top-level listing of all groups for a selected device. (Note that this view would be able to access under the Device view.)
Meters	The Meter view provides a top-level listing of all meters for a selected device. (Note that this view would be able to access under the Device view.)
Links	The Link view provides a top-level listing of all the links in the network.
Hosts	The Host view provides a top-level listing of all the hosts in the network.
Intents	The Intent view provides a top-level listing of all the intents in the network.

The Application view of installed and activated applications on the ONOS controller describing the name, the APP ID, version, category of the application and the developer of that application. The top right corner buttons on the Application view provides the one click installation/ uninstallation, activate/ deactivate, upload and download OAR file of the selected application. ONOS app archive (OAR) files are compiled files of ONOS applications used for installation of applications.

Following is the Application view of some of the applications used during this thesis work.

Applications (169 Total)

Search All Fields

Title	App ID	Version	Category	Origin
BGP Router	org.onosproject.bgprouter	2.7.0	Traffic Engineering	ONOS Community
Basic Optical Drivers	org.onosproject.drivers.optical	2.7.0	Drivers	ONOS Community
Control Message Stats Provider	org.onosproject.openflow-message	2.7.0	Provider	ONOS Community
Default Drivers	org.onosproject.drivers	2.7.0	Drivers	ONOS Community
FIB Installer	org.onosproject.fibinstaller	2.7.0	Traffic Engineering	ONOS Community
Generic OVSDB Drivers	org.onosproject.drivers.ovsdb	2.7.0	Drivers	ONOS Community
Host Location Provider	org.onosproject.hostprovider	2.7.0	Provider	ONOS Community
Host Mobility	org.onosproject.mobility	2.7.0	Utility	ONOS Community
Host Probing Provider	org.onosproject.hostprobingprovider	2.7.0	Provider	ONOS Community
IPv6 RA Generator	org.onosproject.routeradvertisement	2.7.0	Traffic Engineering	ONOS Community
Intent Synchronizer	org.onosproject.intentsynchronizer	2.7.0	Utility	ONOS Community
LLDP Link Provider	org.onosproject.lldpprovider	2.7.0	Provider	ONOS Community
Multicast Forwarding	org.onosproject.mfwd	2.7.0	Traffic Engineering	ONOS Community
ONOS GUI2	org.onosproject.gui2	2.7.0	Graphical User Interface	ONOS Community
ONOS Legacy GUI	org.onosproject.gui	2.7.0	Graphical User Interface	ONOS Community
OVSDB Provider	org.onosproject.ovsdb-base	2.7.0	Provider	ONOS Community
OVSDB Southbound Meta	org.onosproject.ovsdb	2.7.0	Provider	ONOS Community
OVSDB host Provider	org.onosproject.ovsdbhostprovider	2.7.0	Provider	ONOS Community
OpenFlow Agent	org.onosproject.ofagent	2.7.0	Traffic Engineering	ONOS Community
OpenFlow Base Provider	org.onosproject.openflow-base	2.7.0	Provider	ONOS Community
OpenFlow Provider Suite	org.onosproject.openflow	2.7.0	Provider	ONOS Community
Openflow overlay	org.onosproject.workflow.ofoverlay	2.7.0	Utility	ONOS Community

Figure 4. 13: List of Applications installed and activated on ONOS controller

The Device view provides the information about the devices connected to the ONOS controller. It provides the detailed information like Device ID (mostly the MAC address of the device interface connecting to the ONOS controller), Master ONOS controller of that device, Number of ports of that device, vendor, hardware and software version of the device and southbound interface protocol with its version used for communication.

Upon selecting a specific device, the top right corner buttons on the Device view provides more information such as flows, ports, meters, and groups of the device. The following figure shows the list of four devices connected to single Master ONOS controller using OpenFlow protocol version 1.4.

ONOS Open Network Operating System

Devices (4 total)

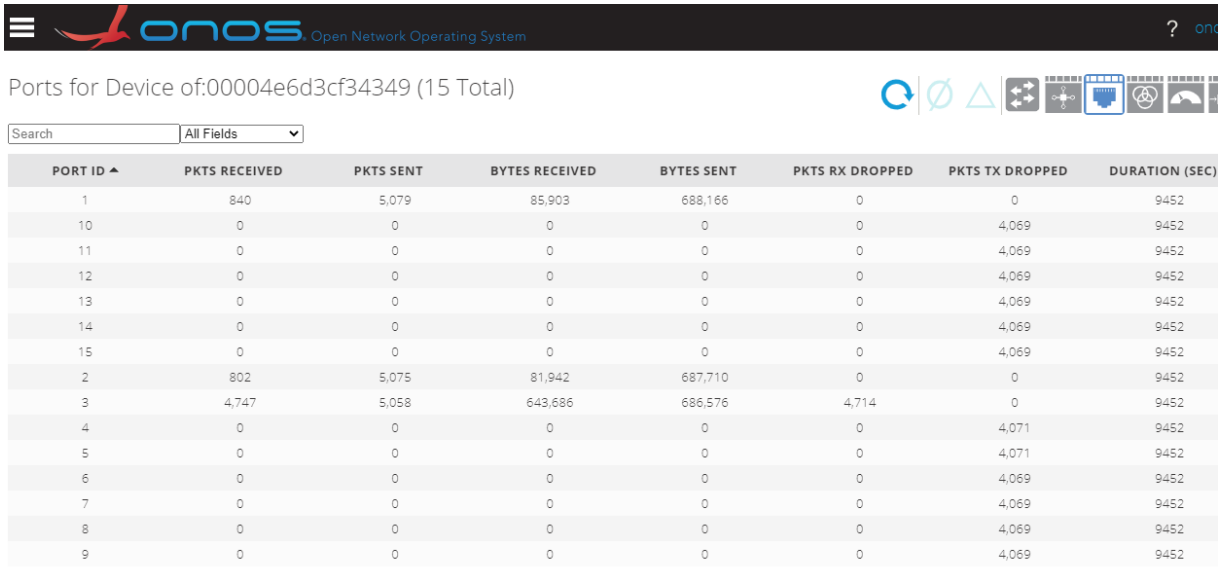
Search All Fields

FRIENDLY NAME	DEVICE ID	MASTER	PORTS	VENDOR	H/W VERSION	S/W VERSION	PROTOCOL
of:0000bedcd3680a49	of:0000bedcd3680a49	192.168.0.114	16	Nicira, Inc.	Open vSwitch	2.12.3	OF_14
of:0000a66b420f9043	of:0000a66b420f9043	192.168.0.114	16	Nicira, Inc.	Open vSwitch	2.12.3	OF_14
of:00005a0521b2b145	of:00005a0521b2b145	192.168.0.114	16	Nicira, Inc.	Open vSwitch	2.12.3	OF_14
of:00004e6d3cf34349	of:00004e6d3cf34349	192.168.0.114	16	Nicira, Inc.	Open vSwitch	2.12.3	OF_14

Figure 4. 14: List of Devices (Open vSwitches) controlled by ONOS controller

Upon selecting a specific device from the Device view and choosing the Port view of that device, one can monitor the traffic flowing through the different ports of that device. Different measurement formats are available on the Port view along with the duration of that port being active.

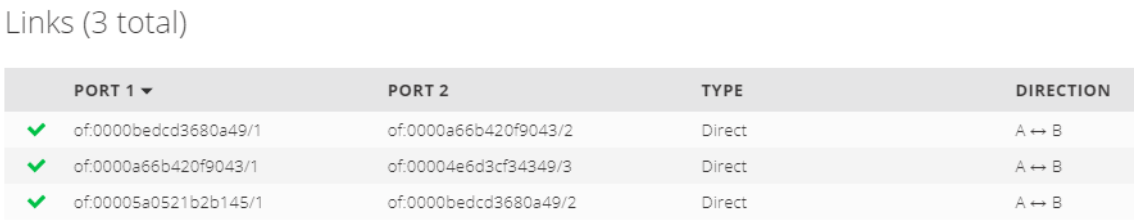
The following figure provides the information about the ports of one of the devices (Open vSwitch-1) connected in the network. Note that the total number of ports mentioned here are 15 whereas in the previous figure 4.14 (Device view) total number of ports for each device are 16. This is because Port 0 (interface eth0) is the management interface whose sole purpose is to accept the configuration from the ONOS controller, therefore this eth0 interface is not displayed in this list of ports. The interfaces eth1, eth2 and eth3 of the Open vSwitch-1 were being connected to other network elements in the network and hence the traffic flowing through these interfaces is observed in the following figure.



PORT ID	PKTS RECEIVED	PKTS SENT	BYTES RECEIVED	BYTES SENT	PKTS RX DROPPED	PKTS TX DROPPED	DURATION (SEC)
1	840	5,079	85,903	688,166	0	0	9452
10	0	0	0	0	0	4,069	9452
11	0	0	0	0	0	4,069	9452
12	0	0	0	0	0	4,069	9452
13	0	0	0	0	0	4,069	9452
14	0	0	0	0	0	4,069	9452
15	0	0	0	0	0	4,069	9452
2	802	5,075	81,942	687,710	0	0	9452
3	4,747	5,058	643,686	686,576	4,714	0	9452
4	0	0	0	0	0	4,071	9452
5	0	0	0	0	0	4,071	9452
6	0	0	0	0	0	4,069	9452
7	0	0	0	0	0	4,069	9452
8	0	0	0	0	0	4,069	9452
9	0	0	0	0	0	4,069	9452

Figure 4. 15: Details of different Ports of one of the Open vSwitch

The Link view displays the list of total Links in the network. These links are only between two devices and not between the host and the device. The following figure presents the list of three links present between the four devices deployed in the network. The exact port (Device ID/ Port) of the device and type of the link can be observed in the figure.



PORT 1	PORT 2	TYPE	DIRECTION
of:0000bedcd3680a49/1	of:0000a66b420f9043/2	Direct	A ↔ B
of:0000a66b420f9043/1	of:00004e6d3cf34349/3	Direct	A ↔ B
of:00005a0521b2b145/1	of:0000bedcd3680a49/2	Direct	A ↔ B

Figure 4. 16: List of all links connected in the SDN network

The Host view displays the list of total Hosts connected in the network. The detailed information of the hosts such as Host ID (MAC address of the host), VLAN ID if host is part of any VLAN, IP address and on which port of the device the host is connected in the network. The following figure displays the list of three hosts connected in the network with their respective IP addresses.

FRIENDLY NAME	HOST ID	MAC ADDRESS	VLAN ID	CONFIGURED	IP ADDRESSES	LOCATION
10.0.0.1	CA:01:0F:AE:00:08/None	CA:01:0F:AE:00:08	None	false	10.0.0.1	of:00004e6d3cf34349/1
10.0.0.2	CA:02:0F:BE:00:08/None	CA:02:0F:BE:00:08	None	false	10.0.0.2	of:00004e6d3cf34349/2
10.0.0.3	CA:03:17:6D:00:08/None	CA:03:17:6D:00:08	None	false	10.0.0.3	of:00005a0521b2b145/2

Figure 4. 17: List of Hosts connected in the SDN network

The following figure displays the Quick help and keyboard shortcuts used on the ONOS GUI.

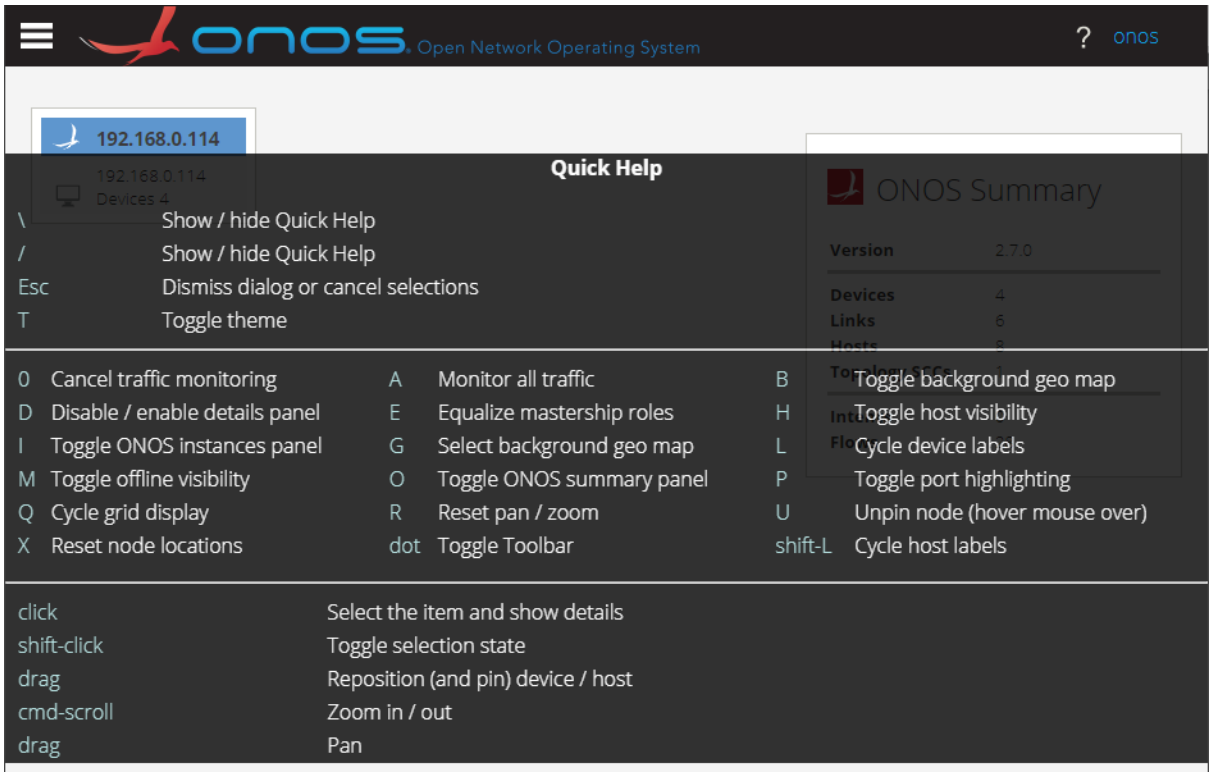


Figure 4. 18: Quick help and keyboard shortcuts used on the ONOS GUI

4.2.2 Creation and Installation of Flows

The SDN controller transmits the controlling commands to the network devices in form of forwarding flow rules. A ***flow*** can be defined as the information about packet forwarding in the network residing on the OpenFlow switch. Flow consists of a priority value, instructions for processing packets, packet match fields, counters, and flow timeouts. The flows are organized using flow tables. An OpenFlow switch consists of several flow tables containing the several flow rules.

Upon the occurrence of specific events, the SDN controller reactively or proactively updates the flow table. In a reactive approach, the controller does not populate the flow table with any rules at the beginning of the network operation. Whenever packet reach at switch during network operation, the switch requests, by sending the unknown/unmatched packet to the controller expecting a flow/rule as a response. This flow is added to the rule table on the switch, allowing for the processing of all packets matching the specific characteristics. In proactive approach, when network operation starts, the controller will install flow entries in the flow table in advance. The proactive approach was introduced to minimize the communication overhead involved between controllers and network devices. In order to maximize network performance, especially in large networks like data centres, choosing the correct flow rules is essential. The flow of an arriving packet is compared to flow entries (packet match-fields) in the flow table as it arrives at a switchport during traffic flow. If matching is missing in the flow table, the switch will communicate with the controller to update the flow table with entries that will let the packet get to its destination. Thus, a delay is created before the packet can be transmitted to the next hop as it involves the communication overhead between the SDN Controller and Open vSwitch.

The OpenFlow pipeline contains multiple flow tables, and each flow table contains multiple flow entries [5]. The flow tables are numbered starting from 0 and a flow table can have a maximum of 65535 flow entries in Open vSwitch. The OpenFlow pipeline processing begins from Flow Table 0 where the packet match-fields are checked. The outcome of the first flow table decides the packet processing by other flow tables. There must be at least one Flow Table in the Open vSwitch. A flow table entry is decided by the match-fields and priority value. The packet match-fields are extracted from the packet type are mostly packet headers like source and destination IP and MAC address. Also, ingress ports and the metadata fields can be used for packet match fields. The metadata field can be used to transfer the information to the other flow tables within the Open vSwitch.

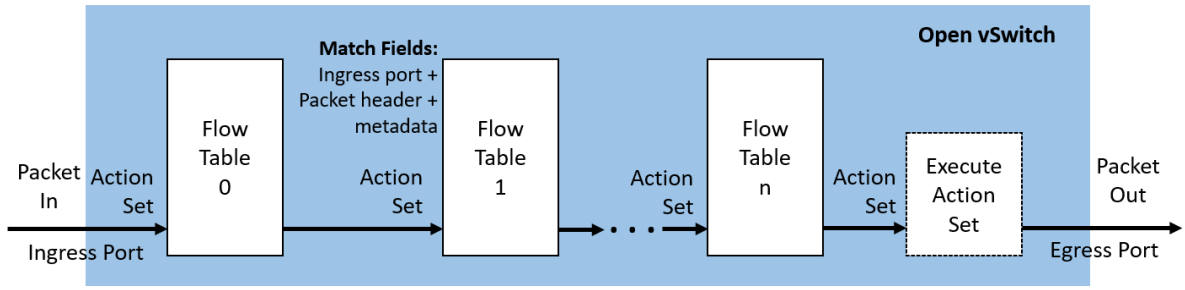


Figure 4. 19: Packet transmission through flow tables within an Open vSwitch

When the packet arrives at the ingress port of the Open vSwitch, it is checked for flow table match fields. When the match-fields are satisfied with the table entry requirement, the packet is processed by the flow table. Inside the flow table the packet is matched with the flow entries and the instruction set included in that flow entry is executed. The instruction might also include the packet to be processed by another flow table and if the instruction set does not include packet forwarding to the next flow table, packet pipeline processing stops. When packet pipeline processing stops, the packet is processed with its associated action set. Action set contains the information to forward the packet from the specified egress port or drop the packet. The priority value of each flow rule is set to provide the preference to specific service. In case of similar match-fields detected for multiple flow rules, the one with the highest priority flow rule will be executed.

An action set is connected with each packet. This set is empty at the beginning of packet processing. An action set contains a maximum of one action of each type. A flow table can modify the action set with the help of instructions linked with a particular match [5].

If Open vSwitch contains just a single flow table, a flow table match field value is set to *ANY* so that it matches all possible values in the packet header. If a packet does not satisfy any match fields, flow entry in the flow table can specify how to process unmatched packet. For example, the packet can be dropped or send the packet to the controller for processing.

The ONOS controller and Open vSwitch together provide three options to install flows on the Open vSwitch. First option through the REST API on the ONOS controller, second option through the commands on ONOS CLI and third option through the commands on Open vSwitch CLI. By default, the flow table 0 is created in the Open vSwitch with default flows directing different type of traffic towards the ONOS controller for processing. Following figure shows the installed flow table 0 on the Open vSwitch CLI before configuring any flow rules. The command ***ovs-ofctl -O OpenFlow14 dump-flows br0*** was used to display all the flows installed on the Open vSwitch. In this command, the OpenFlow protocol version (*OpenFlow14*) used for traffic transmission needs to be specified along with the logical management interface (*br0*) of the Open vSwitch. The following figure displays the output of this command on the Open vSwitch-1 in the created test network.

```
table=0, n_packets=0, n_bytes=0, send_flow_rem priority=5,arp actions=CONTROLLER:65535,clear_actions
table=0, n_packets=461, n_bytes=62696, send_flow_rem priority=40000,dl_type=0x88cc actions=CONTROLLER:65535,clear_actions
table=0, n_packets=461, n_bytes=62696, send_flow_rem priority=40000,dl_type=0x8942 actions=CONTROLLER:65535,clear_actions
table=0, n_packets=45, n_bytes=3150, send_flow_rem priority=5,ipv6 actions=CONTROLLER:65535,clear_actions
table=0, n_packets=6, n_bytes=684, send_flow_rem priority=5,ip actions=CONTROLLER:65535,clear_actions
table=0, n_packets=7, n_bytes=420, send_flow_rem priority=40000,arp actions=CONTROLLER:65535,clear_actions
table=0, n_packets=0, n_bytes=0, send_flow_rem priority=5,ip,nw_dst=224.0.0.0/4 actions=CONTROLLER:65535,clear_actions
```

Figure 4. 20: Flow rules configured on an Open vSwitch

As seen in the above figure, seven flows were already installed on the Open vSwitch before configuring any flows manually. These flows were by default installed by Open vSwitch after getting connected to ONOS controller for smooth transmission of traffic in the network. Note that all flows were directing traffic towards the ONOS controller *(actions=CONTROLLER:65535)* for processing of the received packet. CONTROLLER:65535 means that the switch should send only the first 65535 bytes of the packet to the controller. All the flows were installed under the flow table 0 and action set value was set to clear.

These basic flows were installed to exchange the traffic of basic packet types. These packets contain the header field of Address Resolution Protocol (ARP), IPv4 or IPv6 addresses, Multicast addresses (*nw_dst=224.0.0.0/4*), Link Layer Discovery Protocol (LLDP) (*dl_type=0x88cc*) and Open Network OS (*dl_type=0x8942*).

The following figure displays the same default flow rules from the ONOS CLI.

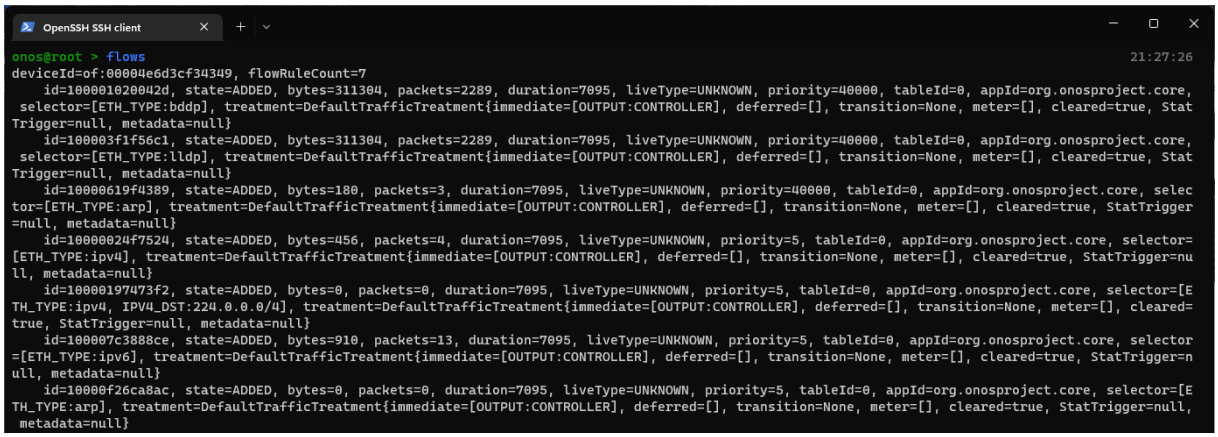


Figure 4. 21: Configured flow rules from the ONOS CLI

To install the flow manually from the Open vSwitch CLI, ***ovs-ofctl add-flow*** command is used. Along with these command few other parameters are required to be passed to install the flows successfully. These must have parameters are flow table number, priority value, packet match fields, action set. The following figure displays the installation of two flow rules, which were installed through the Open vSwitch CLI.

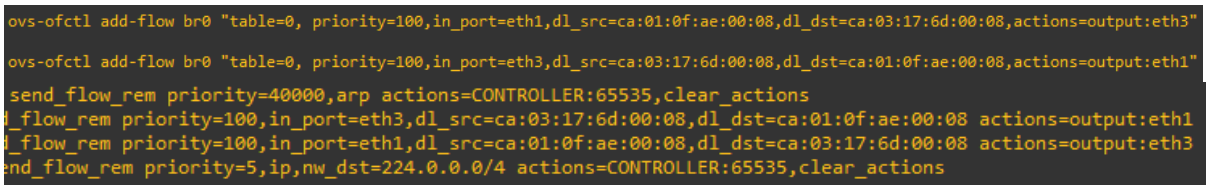


Figure 4. 22: Configuration of flow rules from the Open vSwitch CLI

The flows were installed to transmit the traffic between the Router R1 (mac address: ca:01:0f:ae:00:08) and Router R3 (mac address: ca:03:17:6d:00:08) in the created test network. The following set of commands were used to install the flows on Open vSwitch-1.

ovs-ofctl add-flow br0 "table=0, priority=100, in_port=eth1, dl_src=ca:01:0f:ae:00:08, dl_dst=ca:03:17:6d:00:08, actions=output:eth3"

ovs-ofctl add-flow br0 "table=0, priority=100, in_port=eth3, dl_src=ca:03:17:6d:00:08, dl_dst=ca:01:0f:ae:00:08, actions=output:eth1"

Two flows were installed on the Open vSwitch-1 to transmit and receive the traffic between the routers R1 and R3. Along with add-flow command, several parameters were described to receive the desired output. These parameters were listed in the above set of commands: flow table number (*table=0)*, priority value (*priority=100)*, packet match-fields (*in_port=eth1, dl_src=ca:01:0f:ae:00:08, dl_dst=ca:03:17:6d:00:08,*), action set (*actions=output:eth3)*. In these flow rules, packet match-fields have three conditions to satisfy: input port, source mac address and destination mac address. Upon arrival of any packet at switchport eth1 of Open vSwitch-1, the packet was checked for these packet match fields. When these all three conditions are matched, appropriate performance is carried out as mentioned in the action set, i.e. to forward the packet from switchport eth3 of Open vSwitch-1. Thus, the Open vSwitch will forward the matched packet from eth3 without communicating with the ONOS controller. If any of the match field was not satisfied with the match condition, the packet would be forwarded to the ONOS controller for further processing.

Another method to add flows is through REST API on the ONOS controller. In this method the flow configuration file in JSON [71] format was created. The following figure shows the snippet of pseudocode for flow configuration *flows.json* file.

```
Flows
    {
      Declare priority value of flow : { 0 and 65535 }
      Set the flow entry state : { permanent=true, timeout-0 }
      Set Device ID : { MAC address of logical management interface (Bridge br0) of Open vSwitch }

      # Action to be taken by open vSwitch after the match-fields are matched
      Treatment:
          instructions: { send out the packet from port number,
                          send packet to flow table number, etc. }

      # Specifying the match-fields for flow
      Selector:
          criteria: { source and destination MAC address,
                      source and destination IP address,
                      ingress port, VLAN ID, etc. }
      }
```

Figure 4. 23: Pseudocode for flow rule configuration from the ONOS REST API

The flow was created to transmit the traffic between the Router R2 (mac address: ca:02:0f:be:00:08) and Router R3 (mac address: ca:03:17:6d:00:08) in the created test network using the configuration file.

The selector criteria field basically sets the packet match-field values. These values were set to match the input port, source and destination MAC address and source and destination IP address. The treatment field contains the instructions when the packet match field values are matched. Herein the instruction was set to forward the packet from the interface eth3 of the Open vSwitch-1.

Since the ONOS controller was downloaded and installed as a package, flows.json file was required to be transferred to the server using the cURL specifying the location of file on local machine as shown below.

curl -X POST -H "content-type:application/json" http://192.168.0.114:8181/onos/v1/flows -d @/opt/onos/config/flows.json --user onos:rocks

The successful installation of this flow rule can be observed on the Open vSwitch-1 CLI as seen in the following figure.

```
 send_flow_rem priority=40000,dl_type=0x88cc actions=CONTROLLER:65535,clear_actions
 send_flow_rem priority=40000,dl_type=0x8942 actions=CONTROLLER:65535,clear_actions
end_flow_rem priority=5,ipv6 actions=CONTROLLER:65535,clear_actions
d_flow_rem priority=5,ip actions=CONTROLLER:65535,clear_actions
d_flow_rem priority=40000,arp actions=CONTROLLER:65535,clear_actions
flow_rem priority=100,in_port=eth3,dl_src=ca:03:17:6d:00:08,dl_dst=ca:01:0f:ae:00:08 actions=output:eth1
flow_rem priority=100,in_port=eth1,dl_src=ca:01:0f:ae:00:08,dl_dst=ca:03:17:6d:00:08 actions=output:eth3
low_rem priority=100,in_port=eth2,dl_src=ca:02:0f:be:00:08,dl_dst=ca:03:17:6d:00:08 actions=output:eth3
low_rem priority=100,in_port=eth3,dl_src=ca:03:17:6d:00:08,dl_dst=ca:02:0f:be:00:08 actions=output:eth2
flow_rem priority=5,ip,nw_dst=224.0.0.0/4 actions=CONTROLLER:65535,clear_actions
```

Figure 4. 24: Successful installation of flow rules from the ONOS REST API

For debugging process of flow installation, the flows can be added with the following command used from the ONOS CLI. Temporary flows are installed for the testing purpose and these flows are removed after the test is complete. The flows per device and number of iterations arguments are required to be set with the *add-test-flows* command.

add-test-flows

DESCRIPTION
onos:add-test-flows
Installs a number of test flow rules - for testing only
SYNTAX
onos:add-test-flows [options] flowPerDevice numOfRuns
ARGUMENTS
flowPerDevice
Number of flows to add per device
numOfRuns
Number of iterations
OPTIONS
-j, --json
Output JSON
--help
Display this help message

Figure 4. 25: Command template to add flow from ONOS CLI

The successful performance testing of the flows can be seen in the following figure. In this test scenario, two flows were created on each device and removed after testing. Three such iterations of flow installation and removal were performed. The following snippet shows the test run to be successful and the time taken by each iteration can be observed adjacent to the Run number.

```
onos@root > add-test-flows 2 3
Run 0:
..batch add request
..completed 4450 ± 100 ms
..cleaning up
Run 1:
..batch add request
..completed 4956 ± 100 ms
..cleaning up
Run 2:
..batch add request
..completed 4553 ± 100 ms
..cleaning up
Run is success.
  Run 0 : 8 ms
  Run 1 : 4 ms
  Run 2 : 5 ms
```

Figure 4. 26: Successful flow test execution on the ONOS CLI

Along with all these different flow creation and installation methods, ONOS controller has been built to provide the great overview of possible functions related to the flows management through the RESTful API. On this RESTful API, REST methods such as GET, POST and DELETE can be found to manage the flows on the Open vSwitches. This RESTful API document can be found at the easy to access link, http://192.168.0.114:8181/onos/v1/docs/#/flows

This REST API contains the different features related to the ONOS controller components management. Each of these features comprises of multiple REST methods to GET overview of that component, POST new instructions for that component or DELETE some rules from that component. Each of these REST methods contains the different arguments such as implementation notes, parameters, response messages, curl, request URL, response code, response headers, etc.

The following figure shows the HTTP methods offered at ONOS RESTful API for managing the *Flows* component of the ONOS controller.

Figure 4. 27: REST methods available to configure and manage flow rules

These are the different possibilities offered by ONOS controller and Open vSwitch for creation, installation, and management of the *flow rules*. Another such significant ONOS component is *Intents.*

4.2.3 Creation and Installation of Intents

ONOS controller consists of subsystem named *Intent subsystem.* ONOS refers to the collection of components as a subsystem or sometimes as a service. This Intent subsystem works on the Intent Framework. The Intent Framework is a subsystem that allows applications to specify their network control desires in the form of policy and these policy-based directives are *Intents*. An Intent is a model object that defines an application's request to modify the network's functioning to the ONOS core. The ONOS core receives the intent specifications and translates them with the help of intent compilation, into installable intents, which are primarily actionable operations on the network environment. These actions are executed by intent installation process, which eventually make some changes in the network environment, such as flow rules being installed on the switches, tunnel links being provisioned and many more [72].

When an intent is delivered by an application, it is sent into the compiling phase. Upon successful compiling, an intent would be forwarded to the installing phase or in case of unsuccessful compilation, it would be sent to failed state. After successful installation of an intent, it is finally passed to the installed state or in event of installation failure, an intent is sent to the recompile state. An intent might be withdrawn if an application decides that it no longer requires the intent.

An Intent should consist of Intent ID, Application ID, Intent key, Priority value and Network resources. An Intent might also contain other attributes such as Ingress port, Egress port, Encapsulation type, etc depending on Intent types. ONOS supports various types of Intent installation. Followings are some of the types of Intents.

1. Host to host intents
2. Point to point intents
3. Single point to multi point intents
4. Multi point to single point intents
5. Flow rule intents
6. Flow objective intents

ONOS controller consists of three different techniques for creation of intents in the network environment. First technique is through the commands on ONOS CLI, second technique is through the ONOS GUI, and the third technique is through the REST API on the ONOS controller. The following figure displays the syntax for creating the host-to-host intent from the ONOS CLI. All other types of intents can also be created from the ONOS CLI, with few alterations to the displayed syntax.

add-host-intent

DESCRIPTION
onos:add-host-intent
Installs host-to-host connectivity intent

SYNTAX
onos:add-host-intent [options] one two

ARGUMENTS
one
One host ID
two
Another host ID

OPTIONS

-j, --json Output JSON	**-d, --ethDst** Destination MAC Address	**-l, --lambda** Lambda
--ipDst Destination IP Prefix	**--setEthSrc** Rewrite Source MAC Address	**--ipProto** IP Protocol
-k, --key Intent Key	**-s, --ethSrc** Source MAC Address	**-p, --priority** Priority (defaults to 100)
-b, --bandwidth Bandwidth	**--setEthDst** Rewrite Destination MAC Address	**--tcpDst** Destination TCP Port
-t, --ethType Ethernet Type	**--tcpSrc** Source TCP Port	**--ipSrc** Source IP Prefix

Figure 4. 28: Command template to add intent from ONOS CLI

The use of *add-host-intent* command on the ONOS CLI creates the host-to-host intent between the two hosts whose hosts IDs are provided in the command. Other optional attributes such as Intent key, priority value, ethernet type, source and destination MAC addresses, bandwidth, etc can also be passed along with the intent creation command. The following figure shows the use of this command.

```
onos@root > add-host-intent CA:01:0F:AE:00:08/None CA:03:17:6D:00:08/None                    13:29:33
Host to Host intent submitted:
HostToHostIntent{id=0xb00009, key=0xb00009, appId=DefaultApplicationId{id=2, name=org.onosproject.cli},
priority=100, resources=[CA:01:0F:AE:00:08/None, CA:03:17:6D:00:08/None], selector=DefaultTrafficSelecto
r{criteria=[]}, treatment=DefaultTrafficTreatment{immediate=[NOACTION], deferred=[], transition=None, me
ter=[], cleared=false, StatTrigger=null, metadata=null}, constraints=[LinkTypeConstraint{inclusive=false
, types=[OPTICAL]}], resourceGroup=null, one=CA:01:0F:AE:00:08/None, two=CA:03:17:6D:00:08/None}
```

Figure 4. 29: Configuration of intent from the ONOS CLI

A host-to-host intent was created between the Router R1 (mac address: ca:01:0f:ae:00:08) and Router R3 (mac address: ca:03:17:6d:00:08) in the created test network as seen in figure 4.2. The above figure shows the intent was submitted for compilation and further installation, if all the provided attributes are valid. If the other necessary attributes required for installation of intent are not declared in the command, those are created by ONOS itself. The following figure shows the successful installation of the host-to-host intent.

```
onos@root > intents
Id: 0xb00009
State: INSTALLED
Key: 0xb00009
Intent type: HostToHostIntent
Application Id: org.onosproject.cli
Leader Id: 192.168.0.114
Resources: [CA:01:0F:AE:00:08/None, CA:03:17:6D:00:08/None]
Treatment: [NOACTION]
Constraints: [LinkTypeConstraint{inclusive=false, types=[OPTICAL]}]
Source host: CA:01:0F:AE:00:08/None
Destination host: CA:03:17:6D:00:08/None
```

Figure 4. 30: Configured intents from the ONOS CLI

Another method to create the intents in the network environment is through the ONOS GUI. The ONOS controller provides the easy to create and install the intents between the network elements. To create an intent from the ONOS GUI, press the Shift key on keyboard and select the desired devices displayed on the Topology view. A dialogue box named Selected Items will pop up on the right side of the screen displaying the host IDs of the selected hosts. At the bottom of this dialogue box two buttons are present, one to create flow and other to show related traffic. After selecting the desired hosts, click on the create flow button to create the intent and then show related traffic button to display the related traffic flowing between the Open vSwitches. The following figure shows the creation of one such intent.

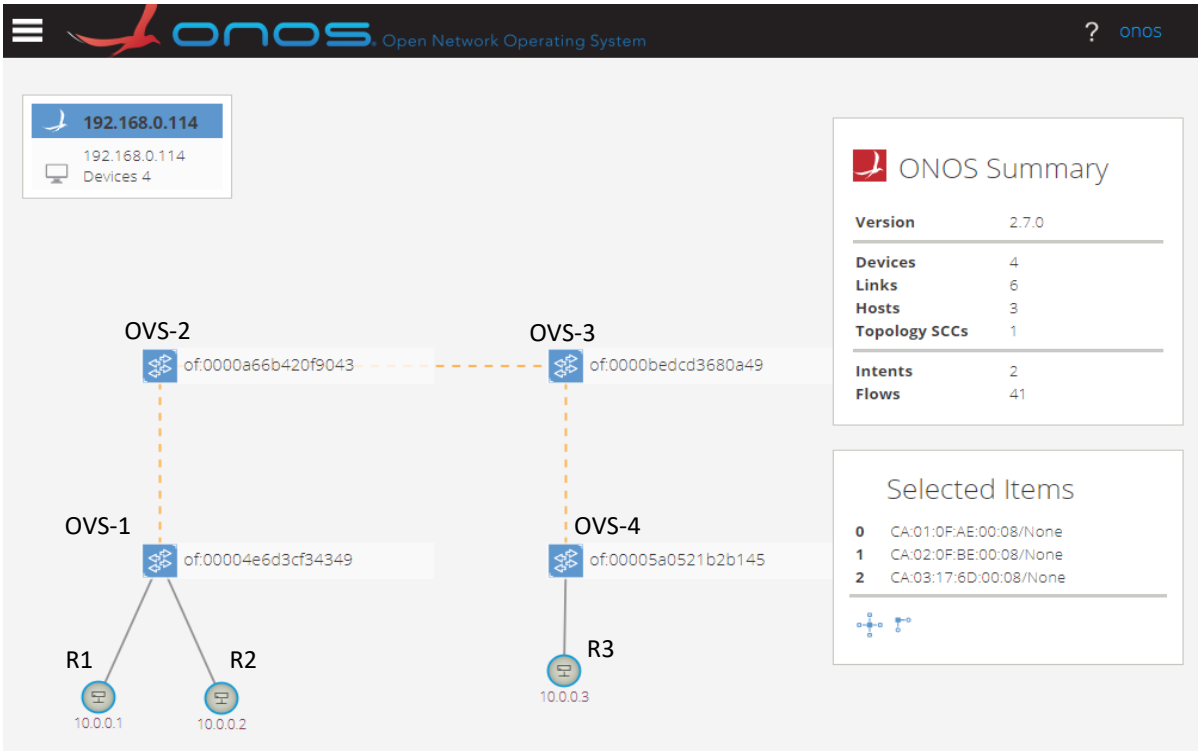


Figure 4. 31: Configuration of intents from the ONOS GUI

For creation of virtual overlay networks, multiple endpoints are required to be configured in same service. For creation of such networks, SinglePoint-to-MultiPoint and MultiPoint-to-SinglePoint intent are used. Hence, the creation of one such intent was tested. MultiPoint-to-SinglePoint intent was created between the three routers Router R1 (MAC address: ca:01:0f:ae:00:08, IP address: 10.0.0.1), Router R2 (MAC address: ca:02:0f:be:00:08, IP address: 10.0.0.2) and Router R3 (MAC address: ca:03:17:6d:00:08, IP address: 10.0.0.3) in the created network. The following figure shows the successful installation of MultiPoint-to-SinglePoint intent from the ONOS GUI.

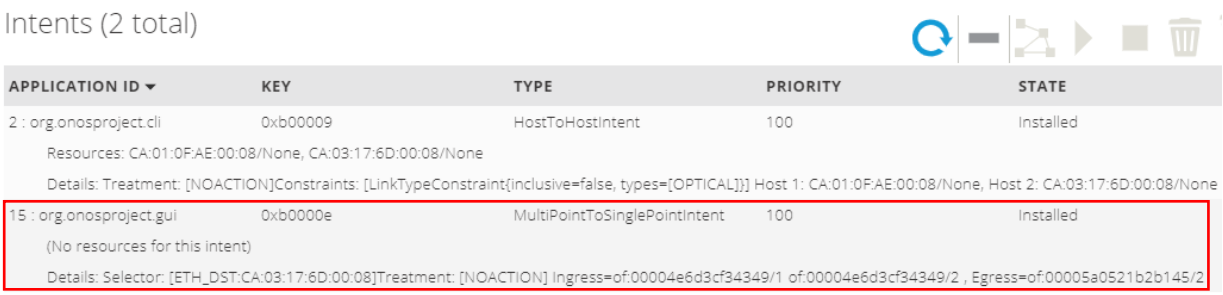


Figure 4. 32: List of different configured intents from the ONOS GUI

If MultiPoint-to-SinglePoint intent is installed from the ONOS CLI, ingress ports of the switch and egress port of the switch are required to be provided rather than the host IDs.

Another method to create the intents is through REST API on the ONOS controller. In this method the intent configuration file is created in JSON format. The following figure shows the snippet of pseudocode for intent configuration *intents.json* file.

```
Intents
    {
    Declare the type of intent : { Host-to-Host, SinglePoint-to-MultiPoint, MultiPoint-to-SinglePoint }
    Set the application ID : { org.onosproject.ovsdb }
    Declare priority value of intent : { 0 and 65535 }

    # Specify the information of devices
    For Host-to-Host intent: { MAC address of the first and second host }

    For SinglePoint-to-MultiPoint intent: { Device ID with ingress port number & Device IDs with egress port numbers }

    For MultiPoint-to-SinglePoint intent: { Device IDs with ingress port numbers & Device ID with egress port number }
    }
```

Figure 4. 33: Pseudocode for intent configuration from the ONOS REST API

A host-to-host intent was created between the Router R2 (mac address: ca:02:0f:be:00:08) and Router R3 (mac address: ca:03:17:6d:00:08) in the created test network using the *intents.json* configuration file. The priority value of 1000 was set, which will basically set the priority value of flow rule developed for this intent. Since this was a host-to-host intent, the MAC address of hosts was specified. Since the ONOS controller was downloaded and installed as a package, intents.json file required to be transferred to the server using the cURL specifying the location of file on local machine as shown below.

curl -X POST -H "content-type:application/json" http://192.168.0.114:8181/onos/v1/intents -d @/opt/onos/config/intents.json --user onos:rocks

The successful installation of this intent can be observed through the ONOS CLI as seen in the following figure.

```
onos@root > intents
Id: 0xb0000e
State: INSTALLED
Key: 0xb0000e
Intent type: MultiPointToSinglePointIntent
Application Id: org.onosproject.gui
Leader Id: 192.168.0.114
Common ingress selector: ETH_DST:CA:03:17:6D:00:08
Treatment: [NOACTION]
Ingress connect points and individual selectors
 -> Connect Point: of:00004e6d3cf34349/2   Selector: Inherited
 -> Connect Point: of:00004e6d3cf34349/1   Selector: Inherited
Egress connect points and individual selectors
 -> Connect Point: of:00005a0521b2b145/2   Selector: Inherited

Id: 0xb00011
State: INSTALLED
Key: 0xb00011
Intent type: HostToHostIntent
Application Id: org.onosproject.ovsdb
Leader Id: 192.168.0.114
Resources: [CA:02:0F:BE:00:08/None, CA:03:17:6D:00:08/None]
Treatment: [NOACTION]
Constraints: [LinkTypeConstraint{inclusive=false, types=[OPTICAL]}]
Source host: CA:02:0F:BE:00:08/None
Destination host: CA:03:17:6D:00:08/None
```

Figure 4. 34: List of configured intents from the ONOS CLI

Along with all these different intent creation and installation methods, ONOS controller has been developed to provide the great overview of possible functions related to the intent management through the RESTful API similar to the flows management. On this RESTful API, HTTP methods such as GET, POST and DELETE can be found to manage the flows on the Open vSwitches. This RESTful API document can be found at the easy to access link, *http://192.168.0.114:8181/onos/v1/docs/#/intents*

This REST API contains the different features related to the ONOS controller components management. Each of these features comprises of multiple HTTP methods to GET overview of that component, POST new instructions for that component or DELETE some rules from that component. Each of these HTTP methods contains the different arguments such as implementation notes, parameters, response messages, curl, request URL, response code, response headers, etc.

The following figure shows the HTTP methods offered at ONOS RESTful API for managing the *Intents* on the ONOS controller.

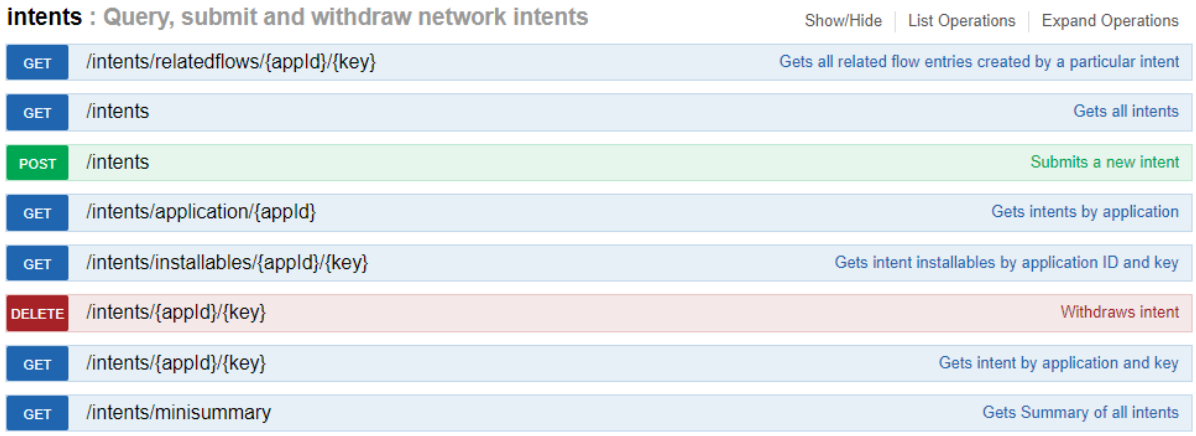


Figure 4. 35: REST methods available to configure and manage intents in the network

These were the different techniques offered by the ONOS controller for creation, installation, and management of the *Intents*. All these techniques were used to create and install the different types of instants between the network elements on the created network environment. The following figure provides the view of all these intents being installed successfully.

```
onos@root > intents -s
Intent type                Total     Installed     Withdrawn       Failed    InstallReq    Compiling    Installing   Recompiling   WithdrawReq   Withdrawing  UnknownState
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
All                            3             3             0            0             0            0            0             0             0             0             0
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
MultiPointToSinglePoint        1             1             0            0             0            0            0             0             0             0             0
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
HostToHost                     2             2             0            0             0            0            0             0             0             0             0
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Figure 4. 36: List of all the configured intents in the network

4.3 Implementation with Mininet

The ONOS controller was integrated and tested with another network emulated environment, Mininet [54]. Mininet provides easy to create virtual network consisting of software switches, hosts, and controller, each of these running real kernel and application code on a single machine. It provides a simple network testbed for creating software-defined networks and developing OpenFlow applications. It also includes a CLI that is topology aware and OpenFlow aware, for debugging and analysing the complete network.

In order to create the testbed for this thesis, Mininet VM image (Mininet's latest version 2.3.0) [73] was downloaded and installed. This pre-packaged Mininet VM image was based on the Ubuntu 20.04 OS. This Mininet VM image includes Mininet itself, all OpenFlow binaries and tools pre-installed, and tweaks to the kernel configuration to create larger virtual networks. To login to this Mininet VM, username: *mininet* and password: *mininet* were used.

Once logged in, a normal Linux CLI was presented. By using ***sudo mn*** command, a small network of two hosts, one switch and one controller was created. After successful creation of this network, directly a Mininet CLI was presented as seen in the following figure. Mininet's CLI allows to control and manage entire virtual network from a single console.

```
mininet@mininet-vm:~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=35821>
<Host h2: h2-eth0:10.0.0.2 pid=35823>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2
:None pid=35828>
<Controller c0: 127.0.0.1:6653 pid=35814>
```

Figure 4. 37: Initiating network in the Mininet

Mininet is configured to install this small network by using ***sudo mn*** command, which basically is a start command for Mininet. As seen in the above figure, a virtual network of the mentioned network nodes was created successfully. A ***dump*** command on Mininet CLI provided the interface information about the created network nodes. Various other commands control and manage the network can be found with ***help*** command on the Mininet CLI.

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['52.4 Gbits/sec', '52.5 Gbits/sec']
```

Figure 4. 38: Use of iperf in the Mininet

One such command was ***iperf*** which was used for testing the TCP bandwidth between the created hosts. As seen in the above figure, results of the above command shows the high bandwidth between the created hosts.

By default, Mininet runs Open vSwitch in OpenFlow mode, which requires an OpenFlow controller. The other SDN controllers already installed in this Mininet VM are OpenFlow reference controller, Open vSwitch's ovs-controller, NOX and Ryu. After testing all these controllers, NOX and Ryu controllers were showing few errors. Another option Mininet provides is to connect to the SDN controller running outside the Mininet VM.

The Mininet is capable of creating and booting up the virtual network at great speed. Even the large-scale networks are created within few seconds of time. The following figure displays the total time taken for creating the network, adding, and configuring all the network nodes, adding links between them, and then destroying the network of 64 hosts, 21 switches and one controller in just 8 seconds.

```
mininet@mininet-vm:~$ sudo mn --topo tree,depth=3,fanout=4 --test none
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29
 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h
56 h57 h58 h59 h60 h61 h62 h63 h64
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21
*** Adding links:
(s1, s2) (s1, s7) (s1, s12) (s1, s17) (s2, s3) (s2, s4) (s2, s5) (s2, s6) (s3, h1) (s3, h2) (s3, h3) (s3,
h4) (s4, h5) (s4, h6) (s4, h7) (s4, h8) (s5, h9) (s5, h10) (s5, h11) (s5, h12) (s6, h13) (s6, h14) (s6, h1
5) (s6, h16) (s7, s8) (s7, s9) (s7, s10) (s7, s11) (s8, h17) (s8, h18) (s8, h19) (s8, h20) (s9, h21) (s9,
h22) (s9, h23) (s9, h24) (s10, h25) (s10, h26) (s10, h27) (s10, h28) (s11, h29) (s11, h30) (s11, h31) (s11
, h32) (s12, s13) (s12, s14) (s12, s15) (s12, s16) (s13, h33) (s13, h34) (s13, h35) (s13, h36) (s14, h37)
(s14, h38) (s14, h39) (s14, h40) (s15, h41) (s15, h42) (s15, h43) (s15, h44) (s16, h45) (s16, h46) (s16, h
47) (s16, h48) (s17, s18) (s17, s19) (s17, s20) (s17, s21) (s18, h49) (s18, h50) (s18, h51) (s18, h52) (s1
9, h53) (s19, h54) (s19, h55) (s19, h56) (s20, h57) (s20, h58) (s20, h59) (s20, h60) (s21, h61) (s21, h62)
 (s21, h63) (s21, h64)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29
 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h
56 h57 h58 h59 h60 h61 h62 h63 h64
*** Starting controller
c0
*** Starting 21 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 ...
*** Stopping 1 controllers
c0
*** Stopping 84 links
............................................................................................
*** Stopping 21 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21
*** Stopping 64 hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29
 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h
56 h57 h58 h59 h60 h61 h62 h63 h64
*** Done
completed in 7.885 seconds
```

Figure 4. 39: Rapid network creation and destruction on Mininet

For creating this network, the tree topology was utilized as seen in the command used. The other two arguments, *depth=3* and *fanout=4* are applied to define the structure of the tree topology. By depth of 3, Mininet understands to create network of three layers of switches or three rows and by fanout of 4, it understands to create four nodes under each network node at each layer. Hence, layer one will have 1 switch, layer two will have 4 switches, layer three will have 16 switches and layer four consists of all 64 hosts. Another argument of *--test none* was utilized thereby the Mininet understands that no tests to be executed and only to set up and tear down the requested topology.

The same network was also tested for checking the connectivity between the hosts. The ICMP ping was tested between these 64 hosts and in total 4096 pings were exchanged between them, and this test was completed in 79.360 seconds.

Mininet supports the creation of many different topologies and running different tests on the network. When Mininet is instructed to create the network, by default, the switches and the controller are put in the root namespace of the machine whereas only the hosts are put in their own separate namespaces.

However, the switches can be initiated in their own namespace, by passing the ***--innamespace*** argument while starting the Mininet. This option does provide an example of how to isolate different switches. In terms of functionality this results in a situation in which the switches will talk to the controller through a separately bridged control connection instead of using a loopback interface. This option does not support the creation of network with Open vSwitches [74].

If the first string typed into the Mininet CLI is a controller, switch or host name, the command is executed on that network node. Only the network is virtualised, each host process sees the same set of processes and directories as that seen by the root network namespace.

The following figures display the example of executing the commands on host h1 and switch s1 from the Mininet CLI. The host h1 was created in its own namespace while switch s1 was created in machine's namespace.

```
mininet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        ether 46:4c:c2:4b:a7:83  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 4. 40: Accessing created host's CLI from the Mininet

```
mininet> s1 ovs-ofctl dump-ports tcp:127.0.0.1:6654
OFPST_PORT reply (xid=0x2): 3 ports
  port LOCAL: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
  port  "s1-eth1": rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
  port  "s1-eth2": rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
```

Figure 4. 41: Accessing created switch's CLI from the Mininet

Mininet supports the creation of desired custom topologies. Custom topologies can be easily defined using a simple Python API. The syntax and example of this Python API is as shown in the following figure.

```python
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def build( self ):
        "Create custom topo."

        # Add hosts
        h1 = self.addHost('h1',ip='10.10.1.1')
        h2 = self.addHost('h2',ip='10.10.2.1')
        h3 = self.addHost('h3',ip='10.10.3.1')
        h4 = self.addHost('h4',ip='10.10.4.1')

        # Add switches
        s1 = self.addSwitch('s1', dpid="0000000000000001", protocols='OpenFlow13', datapath ='user')
        s2 = self.addSwitch('s2', dpid="0000000000000002", protocols='OpenFlow13', datapath ='user')
        s3 = self.addSwitch('s3', dpid="0000000000000003", protocols='OpenFlow13', datapath ='user')
        s4 = self.addSwitch('s4', dpid="0000000000000004", protocols='OpenFlow13', datapath ='user')

        # Add links
        self.addLink(h1, s1)
        self.addLink(h2, s2)
        self.addLink(h3, s3)
        self.addLink(h4, s4)

        self.addLink(s1, s2)
        self.addLink(s2, s3)
        self.addLink(s3, s4)

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Figure 4. 42: Template for creating custom topology on the Mininet

A custom topology of 15 Open vSwitches and 12 hosts was created by using this Python API and connected to the ONOS controller installed outside the Mininet VM. As seen in the following figure, the remote controller arguments need to be passed with the controller's IP address and location of the custom topology Python file while initiating the Mininet.

```
mininet@mininet-vm:~$ sudo mn --controller=remote,ip=192.168.0.114 --custom ~/mininet/custom/mytopo.py
--topo mytopo
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.0.114:6653
*** Adding hosts:
h1 h2 h3 h4 h6 h7 h9 h10 h12 h13 h14 h15
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (h6, s6) (h7, s7) (h9, s9) (h10, s10) (h12, s12) (h13, s13) (h14, s
14) (h15, s15) (s1, s5) (s2, s4) (s2, s5) (s2, s6) (s3, s5) (s4, s8) (s5, s7) (s5, s8) (s5, s9) (s6, s8
) (s7, s11) (s8, s10) (s8, s11) (s8, s12) (s9, s11) (s10, s14) (s11, s13) (s11, s14) (s11, s15) (s12, s
14)
*** Configuring hosts
h1 h2 h3 h4 h6 h7 h9 h10 h12 h13 h14 h15
*** Starting controller
c0
*** Starting 15 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 ...
*** Starting CLI:
```

Figure 4. 43: Creation Mininet network with custom topology

The topology was created and connected to the ONOS controller successfully. The created network was tested with the ICMP ping messages with 100% success rate.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h6 h7 h9 h10 h12 h13 h14 h15
h2 -> h1 h3 h4 h6 h7 h9 h10 h12 h13 h14 h15
h3 -> h1 h2 h4 h6 h7 h9 h10 h12 h13 h14 h15
h4 -> h1 h2 h3 h6 h7 h9 h10 h12 h13 h14 h15
h6 -> h1 h2 h3 h4 h7 h9 h10 h12 h13 h14 h15
h7 -> h1 h2 h3 h4 h6 h9 h10 h12 h13 h14 h15
h9 -> h1 h2 h3 h4 h6 h7 h10 h12 h13 h14 h15
h10 -> h1 h2 h3 h4 h6 h7 h9 h12 h13 h14 h15
h12 -> h1 h2 h3 h4 h6 h7 h9 h10 h13 h14 h15
h13 -> h1 h2 h3 h4 h6 h7 h9 h10 h12 h14 h15
h14 -> h1 h2 h3 h4 h6 h7 h9 h10 h12 h13 h15
h15 -> h1 h2 h3 h4 h6 h7 h9 h10 h12 h13 h14
*** Results: 0% dropped (132/132 received)
```

Figure 4. 44: ICMP ping test between the endpoints in the Mininet network

The following figure shows the view of the created topology from the ONOS controller GUI.

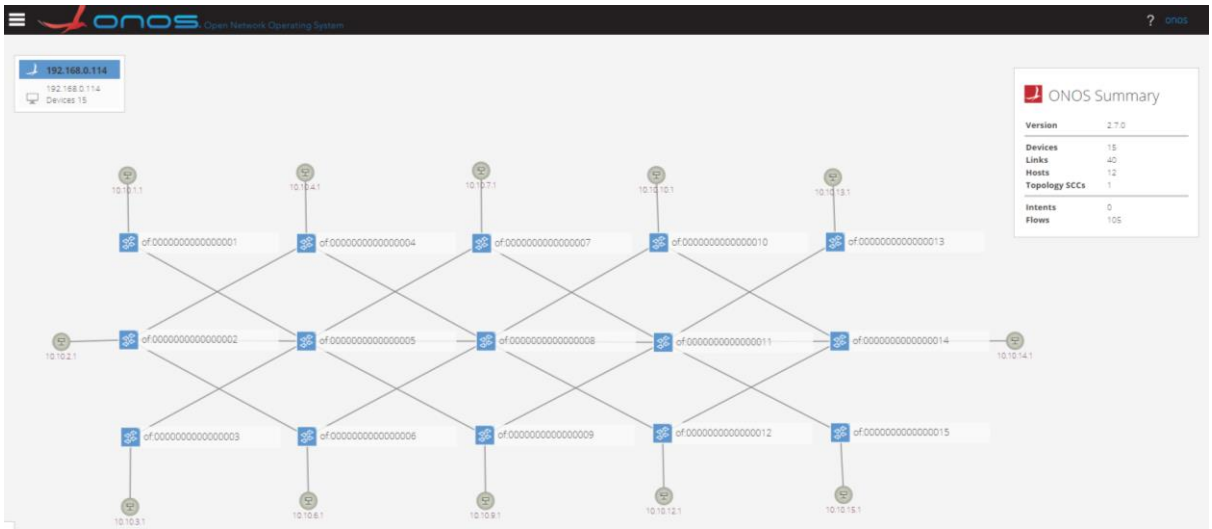


Figure 4. 45: View of created custom Mininet topology from the ONOS GUI

ONOS controller contains a built-in workflow named ***onos.py*** to integrate the ONOS controller with the Mininet emulator [75]. If Mininet has been installed on a machine, ONOS can also be installed on the same machine, and both can be integrated. With Mininet and onos.py, one can easily start up an ONOS controller and a desired virtual network can be created for any topology in a single VM.

This testbed is ideal for the developers who always search for an easy, end-to-end workflow that automatically creates a single or multiple nodes ONOS controller and a virtual Mininet network, all in a single VM running Ubuntu natively. This simplifies development setup because the entire emulated network lives in a single development environment and share a single Linux kernel which eventually is more efficient than multiple VMs development setup.

Additionally, onos.py creates the network in which one can easily change the number of nodes in the ONOS cluster, as well as other components like the delay or bandwidth between the nodes. This workflow provides a single, unified console via the ***mininet-onos*** where one can execute both Mininet and the ONOS commands. This workflow is also suitable to handle the port forwarding, so one can easily connect to the GUI or to the Karaf's CLI by connecting to the correct ports on the VM.

4.4 Problems identified

During the installation of the components used in this Thesis, only a few challenges were faced. Similarly, while implementing the use cases and testing the functionality of SDN controller and the network devices, only a few problems were identified. Following are the problems encountered and possible fixes for the same:

- Installation of SDN controllers is easier when they are downloaded as a packet rather than directly installing from source code and then building the software.
- New versions of OpenDaylight controller do not support the GUI features and layer 2 switch applications since these projects were separated from OpenDaylight. Since the 10th version (Neon) of OpenDaylight controller these projects were separated due to the scarcity of developers.
- On repeated use of same Open vSwitches for multiple use cases, ONOS controller confuses and configures rules from one network to another network. The reason behind this is the Open vSwitches used have the same MAC addresses in both the networks. This can be taken care by clearing all configurations and data on the ONOS controller using command ***wipe-out please*** on ONOS CLI.
- Unlike previous versions of OpenFlow protocol, latest version of this protocol needs to be explicitly configured on all the network devices. Before connecting the Open vSwitch to the controller, Open vSwitch needs to be specified with 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.
- Some of the functioning services of SDN controllers are renamed or completely changed for better functionality of the controller and the documentation available to configure these services are associated with previous versions and hence these documentations are outdated.
- Some of the features which require modification or upgradation according to the new versions of controller's software are not modified and which leads to the absence of those features in the newer versions of the controller. For example, to enable IPv6 on ONOS controller, the features listed on [76] are outdated and the relevant modified features were found manually and listed in use case-3 section of this report.
- Open-source communities are not much active presently as compared to the past. Hence, any queries or doubts about the functionality of controller are not resolved.
- Open vSwitch application available on GNS3 Marketplace (named, Open vSwitch with management interface) has some errors and needs to be fixed before using.

4.5 Use Case-1: Testing the ONOS Controller with Isolated Layer 2 Overlay Networks

From time-to-time Internet Service Providers (ISPs) are puzzled over better VPN services to provide. Where L2VPN offer lower operational cost and higher compatibility regarding VPN tunnels, on the other hand L3VPN offer large scalability and better traffic engineering. The L2VPN services can run over the existing IP core with minimal configuration which provides great opportunity for the ISPs to deploy a new service over the existing infrastructure. Another benefit of L2VPN is packet forwarding information can be learned through the data plane itself without involving the control plane.

L2VPNs like Virtual Private LAN Services (VPLS) [77] offer the service providers support for the multipoint communication and these services have robust security features. VPLS is a shared packet switched network that provides multiple pseudo-wire (PW) [78] connections over the core network. Layer 2 services across a WAN that imitates as an Ethernet LAN in many aspects are offered by the VPLS. Regardless of the location of the sites, all VPLS-connected sites seem to be connected on the same LAN. This network creates a private connection since only customer devices from the same VPLS are allowed to use the connection. ISPs network allows customer devices that are a part of the same VPLS instance to interact and communicate with one another as though they were doing so over a common LAN.

VPLS is a way of providing an L2VPN. Additionally, VPLS is favoured because of some of its features, such as protocol independence and cost-effective operating characteristics [79]. The main goal of VPLS is to connect networks that operate at a global scale as if they are networked on the same LAN. VPLS offers multipoint-to-multipoint Ethernet connectivity over a core Multi-Protocol Label Switching (MPLS) or IP network. The lower operational cost of VPLS can be provided to its better resource utilization.

Originally, VPLS flat architecture was designed, which worked very well for small to medium size of networks. However, in case of larger networks, flat architectures faced significant scalability problems in both data and control planes because of the requirement of a full mesh of PWs. Another architecture, hierarchical VPLS (H-VPLS) architecture was suggested as a solution to this problem. The H-VPLS architecture offers a feasible solution to the scalability issue by decreasing the number of PWs [80]. To overcome the problems of discovery of neighbours, security, and flexibility, initially MPLS was used to implement VPLS. Eventually, two implementations were standardized, first, BGP for auto-discovery and signalling [77] and second, Label Distribution Protocol (LDP) [81] for signalling.

New technologies were proposed such as SDN based VPLS to increase scalability and enhance the security of the service. Software-Defined VPLS (SD-VPLS) promises improved tunnel management, enhanced security, and better scalability. The SDN's principal feature of separate data plane and control plane aligned significantly with VPLS L2VPN service. The significant tunnel establishment delay reduction was observed in SD-VPLS compared to the other VPLS architectures [82]. Thus, the use of SDN in context of VPLS is expected to enhance the capabilities of VPLS. Particularly, SDN can be utilized with VPLS to improve security, scalability, and network programmability, which furthermore increases flexibility and agility. Thus, SD-VPLS can be said one of the most recent technological developments in VPLS [80].

4.5.1 Introduction

In this use case ONOS controller was implemented with the VPLS L2VPN service. The aim was to connect multiple end-users in an OpenFlow network to create the layer 2 broadcast overlay network. Virtually separate paths were created to isolate the traffic flow between the endpoints. VPLS application is by default included in the ONOS platform.

To establish VPLS connectivity between two or more hosts in the network, following things needs to be taken into consideration.

1. A VPLS ID is required to be specified.
2. The ONOS configuration should have at least two Open vSwitch's interfaces configured in it.
3. A VPLS ID should at least contain two such interfaces in it.
4. The network should have at least two hosts connected to it.

When 1st, 2nd and 3rd conditions are satisfied, hosts attached to the VPLS are able to send and receive ARP request messages as broadcast messages. Before establishing unicast communication, it is essential to make sure that all hosts get discovered properly. When 4th condition gets satisfied, i.e., ONOS discovers at least two hosts of the same VPLS ID in the network, unicast communication is established. Hosts configured in the same VPLS can send in packets tagged with the same VLAN ID, different VLAN IDs or no VLAN IDs at all.

VPLS can be installed and configured at setup-time, before pushing the ONOS bits from a management machine to target machines or even at the run-time, while ONOS is running. ONOS VPLS application, **org.onosproject.vpls** needs to be installed and activated before executing VPLS configuration. And this can be carried out from the ONOS CLI or ONOS GUI. ONOS VPLS application supports two encapsulation types, VLAN and MPLS or even without the configuration of VLAN or MPLS this application creates VPLS networks.

An OpenFlow network of six endpoints was created in the GNS3 application for testing this use case. Three hosts (H1, H2 and H3) and three servers (Server1, Serve2 and Server3) were deployed and just configured with the respective IP addresses on the active interfaces. The Open vSwitches and the endpoints were running on the GNS3 VM server application and ONOS controller was installed on the separate Ubuntu VM running outside the GNS3 application. (All components and applications version are same as mentioned in the Table 4.1) NAT interface was used to connect the created network to the ONOS controller as seen in the following figure. OpenFlow protocol version 1.4 was used to transfer the control messages between ONOS controller and Open vSwitches.

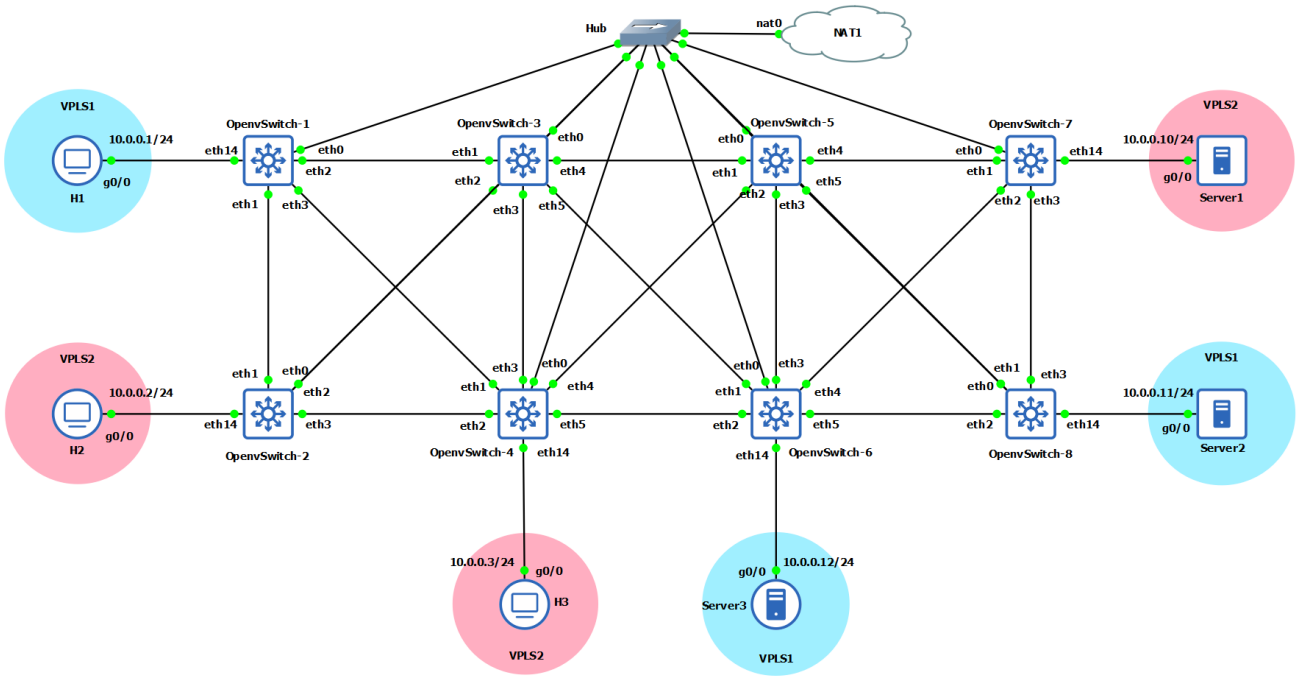


Figure 4. 46: Network created in the GNS3 for L2VPN VPLS

The endpoints were specifically configured to be in the same subnet network for testing the VPLS application. Before configuring the VPLS on the created network, endpoints were able to communicate with every other endpoint. For creating the VPLS networks, different endpoints were selected in the network. To establish the VPLS network two different VPLS were created with VPLS ID-1 and VPLS ID-2. Since there are six endpoints and two different VPLS overlay networks in this network, six Open vSwitch's interfaces were configured into the ONOS configuration. These six interfaces were, Open vSwitch-1's eth14 interface (H1's interface), Open vSwitch-2's eth14 interface (H2's interface), Open vSwitch-4's eth14 interface (H3's interface), Open vSwitch-7's eth14 interface (Server1's interface), Open vSwitch-8's eth14 interface (Server2's interface) and Open vSwitch-6's eth14 interface (Server3's interface). The endpoints H1, Server2 and Server3 were configured into VPLS ID-1 (endpoints highlighted in blue in above figure) and the endpoints H2, H3 and Server1 were configured into VPLS ID-2 (endpoints highlighted in red in above figure).

The VPLS network was tested to be configured by both the configuration methods, through ONOS CLI and through ONOS REST API. Both the configuration methods were successful in implementing the VPLS network as desired. The following figure shows the successful creation and configuration of two VPLS, i.e., VPLS ID-1 (VPLS1) and VPLS ID-2 (VPLS2) using ***vpls list*** command on ONOS CLI. The ***vpls show*** command displays the more details about the created VPLS network which can be seen in the following figure.

```
onos@root > vpls list
VPLS1
VPLS2
onos@root > vpls show
----------------
VPLS name: VPLS2
Associated interfaces: [h2, h3, s1]
Encapsulation: NONE
State: ADDED
----------------
VPLS name: VPLS1
Associated interfaces: [s3, h1, s2]
Encapsulation: NONE
State: ADDED
----------------
```

Figure 4. 47: List of configured VPLS on the ONOS CLI

As aimed, two separate VPLS were created consisting of configured endpoints. None of the encapsulation method was utilized to keep the network simple and test the VPLS network with minimal configuration. The status of *State: ADDED* confirms that the requested VPLS network was successfully configured from the ONOS controller side.

The following figure displays the view of created VPLS network from the ONOS GUI which was accessed through http://192.168.0.106:8181/onos/ui.

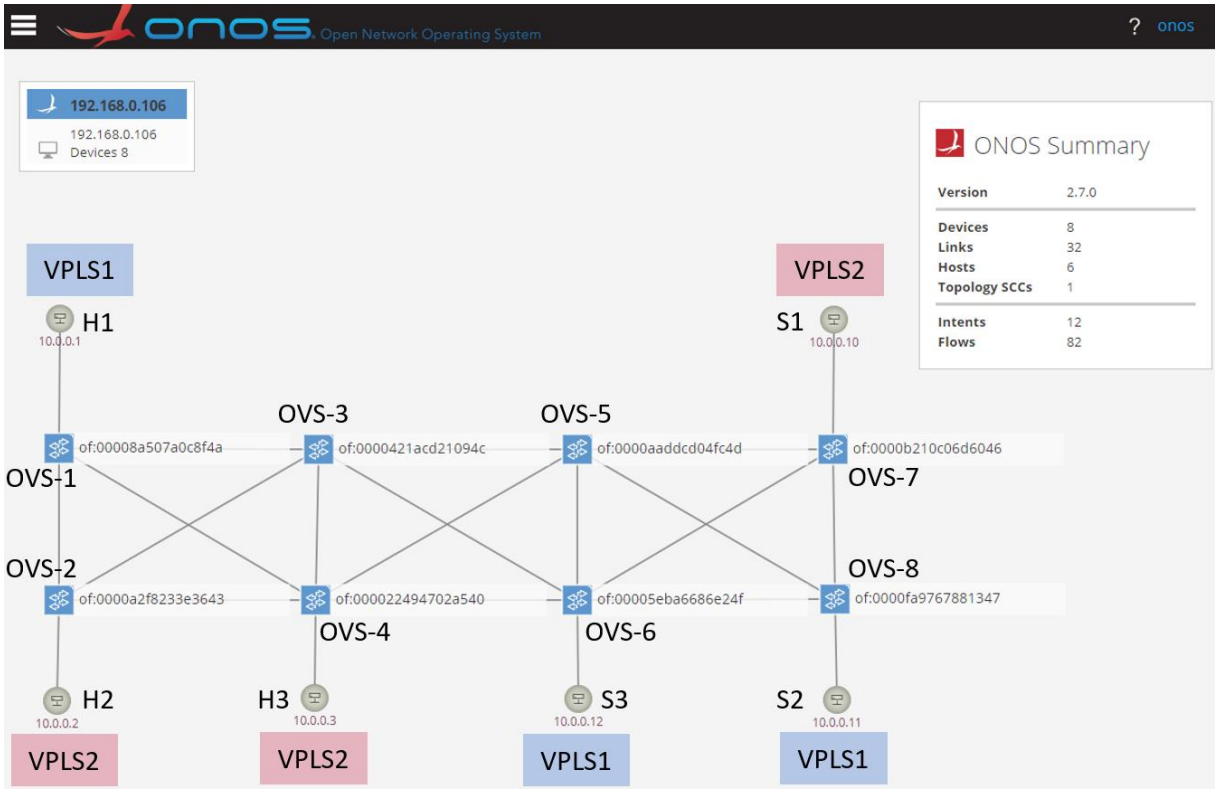


Figure 4. 48: Topology view of the created VPLS network from the ONOS GUI

The successful configuration of VPLS network on ONOS controller results in generation of different intents operations. ONOS generates two types of intents,

1. Single-Point to Multi-Point intents for broadcast traffic, from one ingress point to every egress point, configured in the same VPLS ID.
2. Multi-Point to Single-Point intents for unicast traffic, from every egress point to one ingress point in the same VPLS ID.

For each endpoint within the same VPLS ID, which is a source of broadcast traffic, a single-point to multi-point intent is installed. The ingress point of each intent (single-point) is the switchport where the source endpoint for that intent is connected to. The egress points (multi-point) are any other switchports where destination endpoints of same VPLS ID are connected. The intent ingress selector is defined using the switchport ingress, the destination broadcast MAC address (*FF:FF:FF:FF:FF:FF*) and the VLAN ID of the source endpoint if VLAN is configured. The egress selectors are defined as the ingress selector (broadcast MAC address), but with the VLAN IDs of the destination endpoints if VLANs are configured.

Switchport	VLAN ID	Src MAC	Dest MAC	Src IP	Dest IP	OF: Match-fields	OF: Action
1	10	*	FF:FF:FF:FF:FF:FF	*	*	Ingress port Dest MAC VLAN ID	Egress port 2,3

Figure 4. 49: Processing of broadcast traffic in Open vSwitch with OpenFlow Match-fields and Action

Unicast traffic is transferred through multi-point to single-point intents. For each endpoint within the same VPLS ID, which is destination of the unicast traffic, a multi-point to single-point intent is installed. The ingress points (multi-point) are any switchports where the source endpoints of same VPLS ID are connected. The egress point (single-point) of each intent is the switchport where the destination endpoint is connected to. The intent ingress selector is defined using the switchport ingress, the MAC address of the destination endpoint and the VLAN ID of the source endpoint if VLAN is configured. The egress selector is defined as the ingress selector (matching on the destination MAC address), but with the VLAN ID of the destination endpoint if VLAN is configured.

Intents for unicast traffic are generated only if two or more interfaces are configured in the same VPLS ID and two or more endpoints attached to those configured interfaces send some traffic into the network and get discovered by the Host Service. The reason for this is Unicast intents are not configured by the operator, but instead are generated by the host service after the endpoints are discovered. The following figure displays the generated 12 intents for the created VPLS network with two VPLS IDs. 12 intents are generated because each configured switchport in the network is associated with one broadcast single-point to multi-point intent as well as one unicast multi-point to single-point intent.

Intents (12 total)

APPLICATION ID	KEY	TYPE	PRIORITY	STATE
88 : org.onosproject.vpls	VPLS1-brc-of:0000fa9767881347-14-FF:FF:FF:FF:FF:FF	SinglePointToMultiPointIntent	1100	Installed
88 : org.onosproject.vpls	VPLS1-brc-of:00005eba6686e24f-14-FF:FF:FF:FF:FF:FF	SinglePointToMultiPointIntent	1100	Installed
88 : org.onosproject.vpls	VPLS2-brc-of:0000a2f8233e3643-14-FF:FF:FF:FF:FF:FF	SinglePointToMultiPointIntent	1100	Installed
88 : org.onosproject.vpls	VPLS2-brc-of:0000b210c06d6046-14-FF:FF:FF:FF:FF:FF	SinglePointToMultiPointIntent	1100	Installed
88 : org.onosproject.vpls	VPLS2-brc-of:000022494702a540-14-FF:FF:FF:FF:FF:FF	SinglePointToMultiPointIntent	1100	Installed
88 : org.onosproject.vpls	VPLS1-brc-of:00008a507a0c8f4a-14-FF:FF:FF:FF:FF:FF	SinglePointToMultiPointIntent	1100	Installed
88 : org.onosproject.vpls	VPLS1-uni-of:00005eba6686e24f-14-CA:06:11:AD:00:08	MultiPointToSinglePointIntent	1200	Installed
88 : org.onosproject.vpls	VPLS1-uni-of:00008a507a0c8f4a-14-CA:01:11:1D:00:08	MultiPointToSinglePointIntent	1200	Installed
88 : org.onosproject.vpls	VPLS2-uni-of:000022494702a540-14-CA:05:11:9D:00:08	MultiPointToSinglePointIntent	1200	Installed
88 : org.onosproject.vpls	VPLS2-uni-of:0000a2f8233e3643-14-CA:02:11:2D:00:08	MultiPointToSinglePointIntent	1200	Installed
88 : org.onosproject.vpls	VPLS2-uni-of:0000b210c06d6046-14-CA:03:11:3D:00:08	MultiPointToSinglePointIntent	1200	Installed
88 : org.onosproject.vpls	VPLS1-uni-of:0000fa9767881347-14-CA:04:11:4D:00:08	MultiPointToSinglePointIntent	1200	Installed

Figure 4. 50: List of intents configured by the VPLS application

The following figure shows the detailed information about the intents installed for H1 in the created network. For the broadcast traffic of from H1, broadcast single-point to multi-point intent was installed and for the unicast traffic of from H1, unicast single-point to multi-point intent was installed. It can be observed from the figure that the broadcast traffic as ingress point of Open vSwitch-1's eth14 interface (H1's interface), and egress points of Open vSwitch-8's eth14 interface (Server2's interface) and Open vSwitch-6's eth14 interface (Server3's interface). Similarly, unicast traffic as egress point of Open vSwitch-1's eth14 interface (H1's interface), and ingress points of Open vSwitch-8's eth14 interface (Server2's interface) and Open vSwitch-6's eth14 interface (Server3's interface).

```
Id: 0x100013
State: INSTALLED
Key: VPLS1-brc-of:00008a507a0c8f4a-14-FF:FF:FF:FF:FF:FF
Intent type: SinglePointToMultiPointIntent
Application Id: org.onosproject.vpls
Leader Id: 192.168.0.106
Common ingress selector: ETH_DST:FF:FF:FF:FF:FF:FF
Treatment: [NOACTION]
Constraints: [PartialFailureConstraint]
Ingress connect points and individual selectors
 -> Connect Point: of:00008a507a0c8f4a/14   Selector: Inherited
Egress connect points and individual selectors
 -> Connect Point: of:00005eba6686e24f/14   Selector: Inherited
 -> Connect Point: of:0000fa9767881347/14   Selector: Inherited
```

```
Id: 0x100015
State: INSTALLED
Key: VPLS1-uni-of:00008a507a0c8f4a-14-CA:01:11:1D:00:08
Intent type: MultiPointToSinglePointIntent
Application Id: org.onosproject.vpls
Leader Id: 192.168.0.106
Common ingress selector: ETH_DST:CA:01:11:1D:00:08
Treatment: [NOACTION]
Constraints: [PartialFailureConstraint]
Ingress connect points and individual selectors
 -> Connect Point: of:00005eba6686e24f/14   Selector: Inherited
 -> Connect Point: of:0000fa9767881347/14   Selector: Inherited
Egress connect points and individual selectors
 -> Connect Point: of:00008a507a0c8f4a/14   Selector: Inherited
```

Figure 4. 51: Intents configured by the VPLS application

These intents are further converted into forwarding flow rules for Open vSwitches with match-fields as ingress point (in_port) of packet and destination MAC address. The following figure shows the flow rules installed on the Open vSwitch-1. The first flow was installed by unicast intent, for data traffic from H1 to Server3, the second flow was installed by unicast intent, for data traffic from Server2 and Server3 to H1, the third flow was installed by broadcast intent, for data traffic from Server2 and Server3 to H1, the forth flow was installed by broadcast intent, for data traffic from H1 to Server2 and Server3 and the fifth flow was installed by unicast intent, for data traffic from H1 to Server2.

```
 send_flow_rem priority=1200,in_port=eth14,dl_dst=ca:06:11:ad:00:08 actions=output:eth3
 send_flow_rem priority=1200,in_port=eth2,dl_dst=ca:01:11:1d:00:08 actions=output:eth14
d_flow_rem priority=1100,in_port=eth2,dl_dst=ff:ff:ff:ff:ff:ff actions=output:eth14
d_flow_rem priority=1100,in_port=eth14,dl_dst=ff:ff:ff:ff:ff:ff actions=output:eth3
 send_flow_rem priority=1200,in_port=eth14,dl_dst=ca:04:11:4d:00:08 actions=output:eth3
```

Figure 4. 52: Flow rules installed on the Open vSwitch-1 by the ONOS controller as per configured VPLS

4.5.2 Configuration and Working

L2VPN service VPLS runs on top of ONOS controller as a distributed application. When running multiple ONOS controllers as a cluster, VPLS applications from each ONOS instance share their status and use common data structures. ONOS controller takes care of maintaining all applications in synchronisation. On initialisation of multiple VPLS applications, the VPLS Operation Manager from each ONOS instance sends a request to LeadershipService to know which instance is the leader among the running VPLS applications. Only the leader node can execute VPLS operations and generate Intents on the VPLS network.

There are several components associated for managing and controlling the VPLS application. These components are VPLS Manager, VPLS Operation Manager, VPLS Store, VPLS Neighbour Handler, VPLS Config Manager.

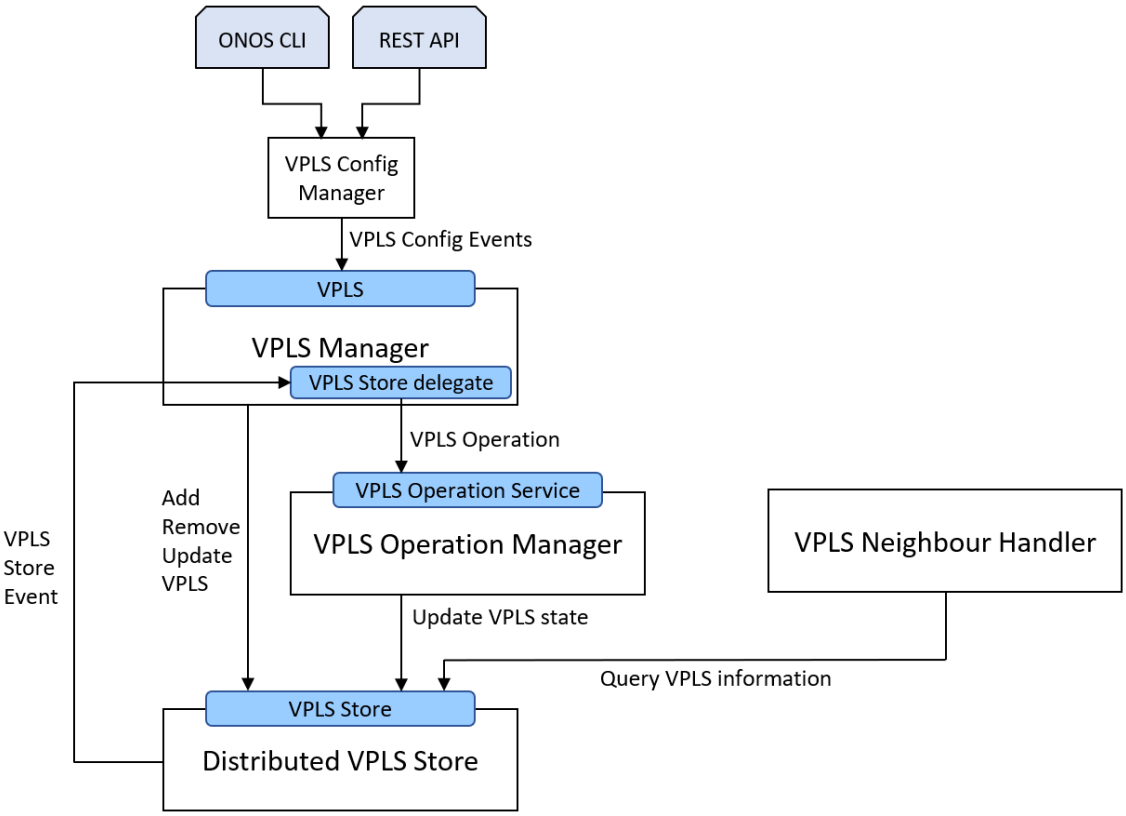


Figure 4. 53: Components of VPLS application

VPLS configuration can be submitted from ONOS CLI or ONOS REST API. When the VPLS configuration is submitted to the VPLS application, it arrives at the VPLS Config Manager. Any host event information such as removal or addition of any host in the network is taken care by the VPLS Config Manager. The VPLS Config Manager component administers the update events of ONOS network configurations and optimizes changes from VPLS network config event to invoke VPLS API. This VPLS API is provided by the VPLS Manager for managing VPLS. The API provides create, update, read and delete functionality. VPLS Manager also helps other application to interact with VPLS Store and VPLS Operation Manager. It is also responsible for handling the host events such as attach or detach of hosts from the VPLS network. VPLS Manager component contains another component in itself named, VPLS Store Delegate. This component is responsible of handling the VPLS store events and generate new VPLS operations according to the store event type and VPLS status. After generation of an operation, it is forwarded to VPLS Operation Manager directly. The VPLS Operation Manager manages modification of all operations for VPLS. It converts the VPLS operations to sets of Intent operations and provide correct order of Intent operations. On completion or failure of VPLS operation, it updates the success or error status to VPLS Store. The VPLS Store component stores all VPLS information and transfers VPLS information to network config system. VPLS Neighbour Handler handles the neighbour messages of any VPLS from every interface [83]. This is the general workflow of VPLS application for managing and controlling the VPLS network.

The use case was tested for VPLS network configuration by both the configuration methods, through ONOS CLI and through ONOS REST API. Both the configuration methods were successful in configuring the VPLS network as desired.

First VPLS configuration method tested in this use case was configuration through ONOS CLI. ONOS CLI provides commands to set up VPLS network through its command line interface. Note that these commands are not available by default and are activated along with the installation and activation of VPLS application. To begin with switchports configuration, the **interface-add** command was used along with arguments defining the Open vSwitch's ID and port number and name of the endpoint connected to that switchport. Six switchports as discussed earlier in this chapter were configured through the ONOS CLI as seen in the following figure.

```
onos@root > interface-add of:00008a507a0c8f4a/14 h1
Interface added
onos@root > interface-add of:0000a2f8233e3643/14 h2
Interface added
onos@root > interface-add of:000022494702a540/14 h3
Interface added
onos@root > interface-add of:0000b210c06d6046/14 s1
Interface added
onos@root > interface-add of:0000fa9767881347/14 s2
Interface added
onos@root > interface-add of:00005eba6686e24f/14 s3
Interface added
```

Figure 4. 54: Configuration of VPLS interfaces from the ONOS CLI

The VPLS command on ONOS CLI provides various options for configuration and management of VPLS network. These options are *create, list, set-encap, clean, delete, add-if* (adding interface), *rem-if* (removing interface) and *show*. First the VPLS networks are created with appropriate VPLS IDs and then the relevant switchport interfaces are configured into the VPLS IDs. The *set-encap* command provides the option for configuring the encapsulation type used for that VPLS ID such as encapsulation with VLAN, MPLS or none. Likewise other commands can be used to modify or remove the component from the VPLS network. The *vpls list* and *vpls show* commands are used to check the VPLS configuration and status as seen in the following figure.

```
onos@root > vpls create VPLS1
onos@root > vpls add-if VPLS1 h1
onos@root > vpls add-if VPLS1 s2
onos@root > vpls add-if VPLS1 s3
onos@root >
onos@root > vpls create VPLS2
onos@root > vpls add-if VPLS2 h2
onos@root > vpls add-if VPLS2 h3
onos@root > vpls add-if VPLS2 s1
onos@root >
onos@root > vpls list
VPLS1
VPLS2
onos@root > vpls show
----------------
VPLS name: VPLS2
Associated interfaces: [h2, h3, s1]
Encapsulation: NONE
State: ADDED
----------------
VPLS name: VPLS1
Associated interfaces: [s3, h1, s2]
Encapsulation: NONE
State: ADDED
----------------
```

Figure 4. 55: Configuration of VPLS from the ONOS CLI

The configuration method from ONOS CLI does not provide the option to configure the MAC addresses of the endpoints. MAC address being the vital entity for this L2VPN service, VPLS application sometimes won't be configured correctly if MAC addresses are mismatched. Due to rigorous use of same ONOS controller and same Open vSwitches having same MAC addresses for various use cases, it was observed that the ONOS CLI method of configuration was configuring random MAC addresses to the endpoints. Therefore, it was identified that ONOS REST API configuration method was more suitable in configuration of successful VPLS network compared to the configuration through the ONOS CLI method.

Another VPLS configuration method tested in this use case was configuration through ONOS REST API. For the configuration through REST API, the configuration file in JSON format named, ***network-cfg-vpls.json*** was created. The following figure shows the snippet of pseudocode for VPLS configuration file.

```
VPLS configuration
        {
            # Specify the list of Open vSwitch ports where hosts are connected
            Device ID : { MAC address of logical management interface (Bridge br0) of Open vSwitch }
            Interfaces : { name, MAC address and VLAN ID of the host device }

            Set the application ID of VPLS : { org.onosproject.vpls }

            # Create the list of VPLS overlay network
            Set VPLS ID : { name for VPLS }
            Add the interfaces to the created VPLS ID : { name of the host device participating in the same VPLS ID }
            Set encapsulation method for VLAN: { VLAN, MPLS or none }
        }
```

Figure 4. 56: Pseudocode for configuration of VPLS from the ONOS REST API

The list of Open vSwitch ports defining the endpoint location containing the Device IDs and interfaces was created. The interfaces were configured with the appropriate host (endpoint) names and their respective MAC addresses. If VLAN IDs are also used in the VPLS network to distinguish endpoints, VLAN IDs can also be configured under each interface. VPLS application, **org.onosproject.vpls** was configured along with list of VPLS IDs. To set up the VPLS network for this use case, two different VPLS networks were created with VPLS ID-1 (VPLS 1) and VPLS ID-2 (VPLS 2). The endpoints, H1's, Server2's and Server3's interfaces were configured into VPLS 1 and the endpoints, H2's, H3's and Server1's interfaces were configured into VPLS 2. If any encapsulation method is used in the VPLS network, encapsulation type can be defined under each VPLS IDs.

If ONOS controller is downloaded as a source code, after placing VPLS application files in appropriate directory, it needs to be built again. Whereas, if the ONOS controller is downloaded and installed as a package, network-cfg-vpls.json file needs to be transferred to the server using the cURL specifying the location of file on local machine as shown below.

curl -X POST -H "content-type:application/json" http://192.168.0.106:8181/onos/v1/network/configuration -d @/opt/onos/config/network-cfg-vpls.json --user onos:rocks

The successful configuration of the VPLS network results in creation of intents on the VPLS network as discussed earlier in the chapter. These intents further generate some flow rules in the flow tables of involved Open vSwitches. According to the working of VPLS network in this use case, five flow rules (incoming & outgoing of broadcast messages and incoming & outgoing of Unicast messages) were configured on each involved Open vSwitch. The following figure displays these five flow rules generated on the Open vSwitch-1 in the created VPLS network.

Finally, the working of created VPLS network was tested from the endpoints. The successful configuration and implementation of the VPLS network can be observed through the ONOS CLI. The endpoints configured in the same VPLS IDs were able to send traffic to each other while the traffic was not reaching to the endpoints configured in another VPLS IDs. The following figure shows the implemented VPLS networks.

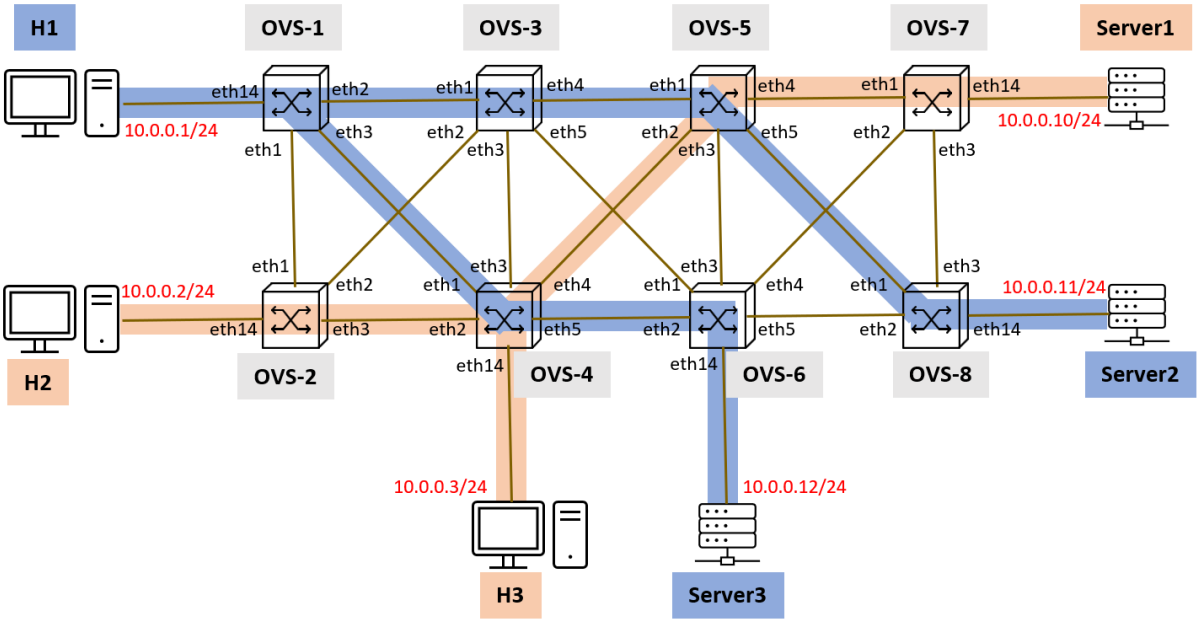


Figure 4. 57: Implemented VPLS networks

Hence, the VPLS network configuration was successfully carried out through a single REST API in ONOS controller. While legacy technologies require the manual configuration of multiple devices in the network, SD-VPLS tries to make the process easier for network operators.

4.6 Use Case-2: Testing the Network with Multiple ONOS Controllers

Software-defined networks with single centralised point for controlling the network were facing the major problems of efficiency, scalability, high availability, redundancy, and security [84]. These challenges led the network designers to implement the multi-controller architectures. For multi-controller architectures, physically-distributed architecture was proposed, with further different types as logically-centralised, logically-distributed architectures, flat and hierarchical architectures.

In physically-distributed network architecture, as the name suggests, the controllers are located at different physical location which further increases the network scalability. This architecture also defines the controller aspects like how to place controllers and which type of communication to use among them. A logically-centralised type of architecture means to distribute the control authority among the controllers so that they appear as a single controller for the underlying network. All the controllers have the complete view of the network and are responsible of the same tasks in the network. Unlike controllers of logically-centralised architecture, the controllers in logically-distributed architecture are responsible for controlling the network devices present in their own scope of network [22].

The flat and hierarchical type of architectures define the aspects of the control plane. In a flat type of architecture, all the controllers are placed on just one layer having just one layer of control plane. All the controllers have the complete view of the network and are responsible of the same tasks in the network. In flat architecture, the network provides more resilience to failures but the task of managing all the controllers becomes difficult. In a hierarchical type of architecture, the controllers are arranged in multiple layers having two or three layers of control plane [22]. Here the controllers have different sectional view of the network that they are responsible for controlling.

ONOS controller supports the physically-distributed and logically-centralised type of architecture. ONOS is built to tackle the problems of scalability, high-availability, and performance. ONOS's distributed core relieves the application developers' tasks of cluster coordination and state management by offering an available set of core building blocks for dealing with different types of distributed control plane state [23].

4.6.1 Introduction

In this use case, a cluster of three ONOS controllers was formed and tested on the network. A single instance of ONOS installed on a virtual machine consumes 3-4 GB of RAM and a minimum of 3 CPUs, and due to limited availability of resources on the host machine, implementation of this use case was not possible using the standard installation process. Hence, the ONOS controllers were installed on the Docker containers for this use case. Three Docker instances were created to build the cluster of three ONOS controllers. The network of Open vSwitches was created using the Mininet emulator.

ONOS is built to deploy the collection of servers, which can communicate with each other to perform the same tasks, distribute responsibilities, and control the same underlying network. The physically-distributed and logically-centralised architecture of ONOS answers the problems of network scalability and high availability. The distributed ONOS environment provides the fault-tolerance and resilience even if a controller instance from the cluster fails.

ONOS controller cluster is formed on the Atomix cluster [85]. An Atomix framework is designed for implementation of scalable networks and fault-tolerant distributed systems. The Atomix framework is part of ONOS project for cluster formation and management since ONOS version 1.14 (Owl). An Atomix cluster is responsible for ONOS controllers cluster management, data storage and service discovery [86]. The detailed information of downloading and installing Atomix as well as ONOS docker can be found on the ONOS wiki page [87]. After installation and initiating the docker instances for Atomix and ONOS containers, appropriate configuration for each docker instance is required to be performed. This configuration must be corresponding to the number of nodes running in the ONOS controller cluster formed. For instance, if cluster is formed of five ONOS controllers, five different configuration files would have to be created. Similarly, if cluster is formed of three Atomix instances, three different configuration files would have to be created. The number of ONOS controllers in a cluster and the number of Atomix instances in a cluster should always be equivalent, but it is recommended to have at least three of each instance in their respective clusters.

In this use-case, cluster of three Atomix instances and cluster of three ONOS instances were created. Each of these instances was running on different docker container. The following figure shows the implementation of these clusters along with the IP address of each docker container.

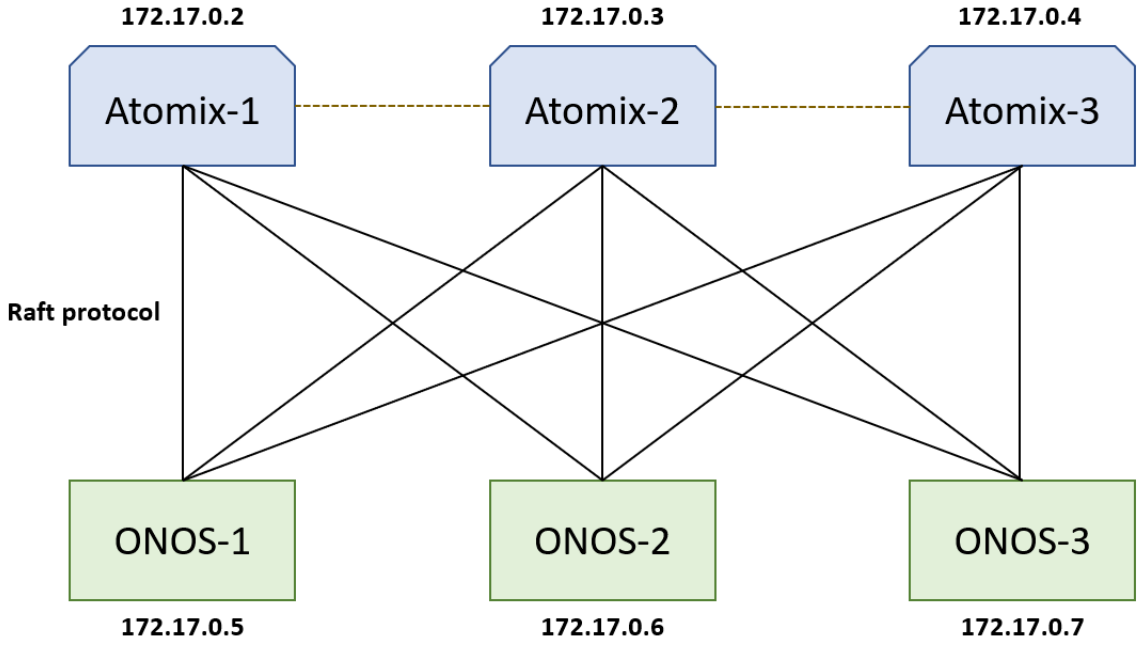


Figure 4. 58: Cluster formation of Atomix and ONOS

The configuration files of Atomix instances are required to be installed on all running docker instances for Atomix cluster. The following figure shows the pseudocode for the configuration of Atomix cluster.

```
Atomix Cluster
        {
            Set the cluster ID : { onos }
            Set the node ID : { atomix-number }
            Set the IP address of atomix container : { x.x.y.y }
            Set the port number to lister to onos : { 5679 }

            # List of atomix containers part of the cluster
            Discovery:
                    { node ID and IP address of atomix container }

            # Specify the management group information for configuration of managing primitives
            type : { raft }
            partitions : { number of partitions }
            members : { node IDs of atomix container }

            # Specify the partition group information for configuration of primitive storage and replication
            type : { raft }
            partitions : { number of partitions }
            members : { node IDs of atomix container }
        }
```

Figure 4. 59: Pseudocode for configuration of Atomix cluster

The cluster configuration in Atomix configuration file specifies the details about all the nodes of Atomix instances running in the cluster. Since ONOS functions on Raft protocol [9] for cluster formation, this protocol must be defined in the management-group for managing the Atomix cluster. The partition-groups lists the members of the cluster, storage level and number of partitions. ONOS recommends to make partition group of more than twice the number of Atomix nodes present in the cluster [88]. The configuration files of ONOS controller are installed on individual running docker instance for ONOS controller cluster. The following figure shows the pseudocode for the configuration of Code cluster.

```
ONOS Cluster
        {
            Set the name : { onos }
            Set the node ID : { onos-number }
            Set the IP address of onos container : { x.x.y.y }
            Set the port number : { 9876 }

            # Specify the information about atomix cluster
            Storage:
                {
                    IP address of atomix container : { raft }
                    node ID of atomix container : { raft }
                    port number of atomix container : { 5679 }
                }
        }
```

Figure 4. 60: Pseudocode for configuration of ONOS cluster

Since, the ONOS controller cluster creation and management is handled by the Atomix framework, the ONOS cluster configuration files contain just the information about the Atomix cluster. On initiation of ONOS instance in the network, the location is informed to the Atomix. The *Atomix group membership* primitive is used to determine the changes of initiation or failure of any ONOS controller. Atomix then informs the other ONOS instances about the new ONOS instance via Atomix cluster. ONOS instances connect to each other and share the information about underlying network through the peer-to-peer communication. A partition group can be defined as a set of shards implementing a specific distributed systems protocol, Raft protocol in case of ONOS [88].

On successful installation and configuration of both, Atomix and ONOS controller clusters are formed, and the details about the creation of partitions can be checked from the ONOS GUI as seen in the following figure.

Partitions (7 total)

NAME	TERM	LEADER	MEMBERS
7	0	atomix-2	atomix-2, atomix-3, atomix-1
6	0	atomix-2	atomix-2, atomix-3, atomix-1
5	0	atomix-2	atomix-2, atomix-3, atomix-1
4	0	atomix-2	atomix-2, atomix-3, atomix-1
3	0	atomix-1	atomix-2, atomix-3, atomix-1
2	0	atomix-1	atomix-2, atomix-3, atomix-1
1	0	atomix-1	atomix-2, atomix-3, atomix-1

Figure 4. 61: Details about formed Atomix cluster from ONOS GUI

In this use case, cluster of three Atomix instances with group of 7 partitions was created as configured. The network was controlled by the cluster of three ONOS controllers. For this use case, the underlying network was created in the Mininet environment. A Spine-Leaf topology was deployed consisting of six Open vSwitches and 20 hosts.

In the following figure, three ONOS controllers can be observed in ONOS Cluster Node Panel on ONOS GUI. Three ONOS controllers are displayed in three different colours (navy blue, blue & red) having three different IP addresses (172.17.0.5, 172.17.0.6 & 172.17.0.7) respectively.

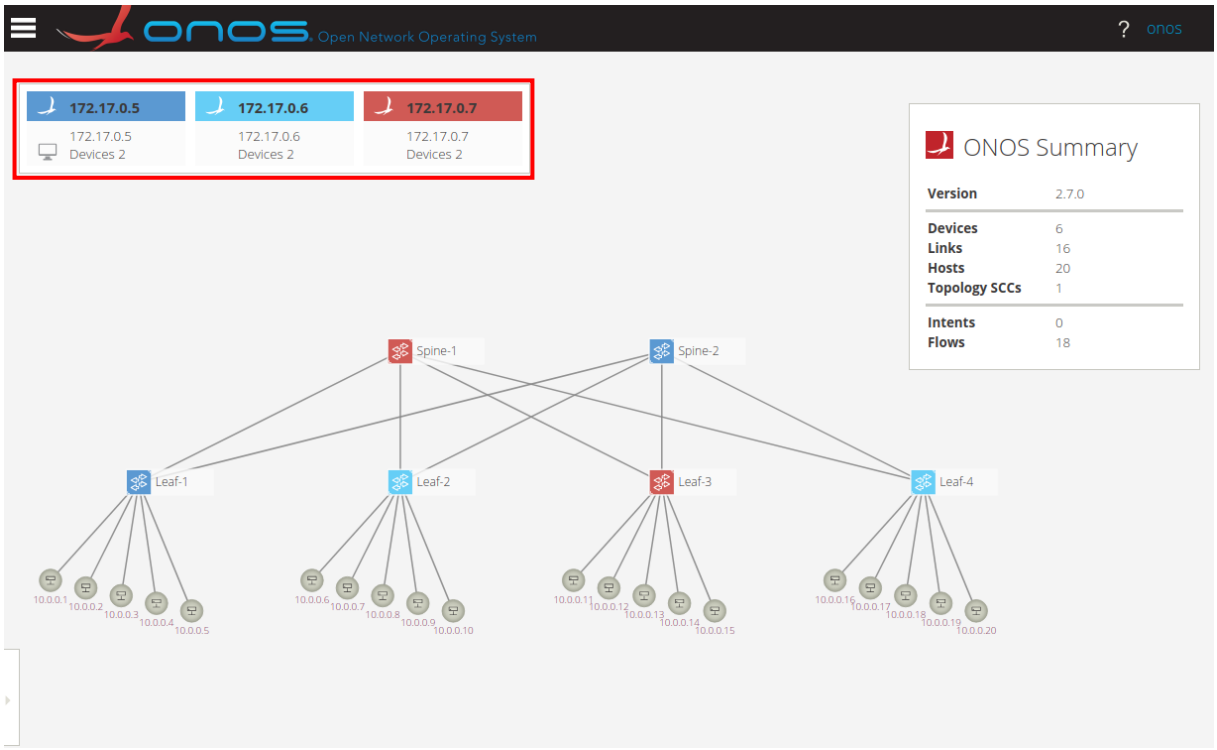


Figure 4. 62: ONOS controllers cluster view on the ONOS GUI

ONOS controller cluster formed here is physically-distributed (different IP locations) and logically-centralised (controlling the same network), which proves the distributed type of architecture discussed earlier in this chapter. Cluster of these ONOS controllers acts together as a unified and coherent distributed system. The six Open vSwitches are equally distributed among the three ONOS controllers. The switches have chosen a master controller for themselves by process explained in next sub-chapter.

The details about the ONOS controller cluster can be found under Cluster Nodes tab on ONOS GUI as shown in the following figure. The ONOS controllers communicate with each other using the TCP port number mentioned in the ONOS controller cluster formation configuration files.

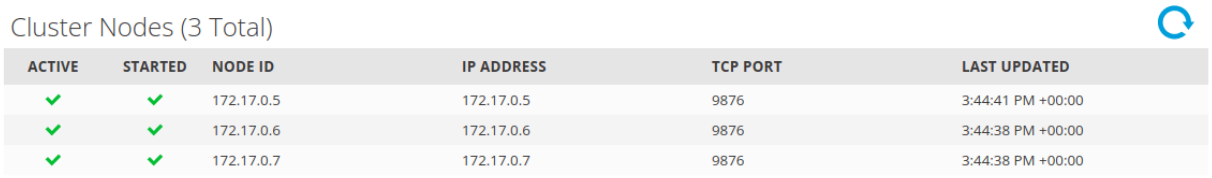


Figure 4. 63: Cluster Nodes of three ONOS controllers

4.6.2 Master Controllers with their Devices

The ONOS controller refers to the Open vSwitches as Devices. The switches connected in the network are open to choose a master controller for themselves from the available controllers in the cluster. A switch can have one and only one master controller, whereas a master controller can have multiple switches associated with itself. The function of master controller is to read and write the control instructions on the associated Open vSwitches. If the associated master controller shuts down, ONOS elects a new master from the remaining available controllers that the switch can communicate with.

All the ONOS controllers present in the cluster are able to track the state of all Open vSwitches in the network. A controller can be associated with any role out of three roles defined by the OpenFlow. These roles can be Master role, Standby role or None role.

1. A Master role of controller has full control over the switch and can read and write instructions on that switch.
2. A Standby role of controller has partial control over the switch and can only read instructions from that switch.
3. A None role of controller cannot read instructions from the switch and may or may not know the switch in the network.

When the new switch is connected in the network, all the controllers present in the cluster start collecting information from that switch. The first controller to collect the information that none of the controller is associated master to that switch and setup a control channel with that switch, becomes a master controller for that switch. Other controllers that discover this switch later are given the Standby role. If any switches are shut down or lose connection in the network, the controller is associated with the None role.

The following figure shows the list of devices (Open vSwitches) connected in the network with their respective Master controller.

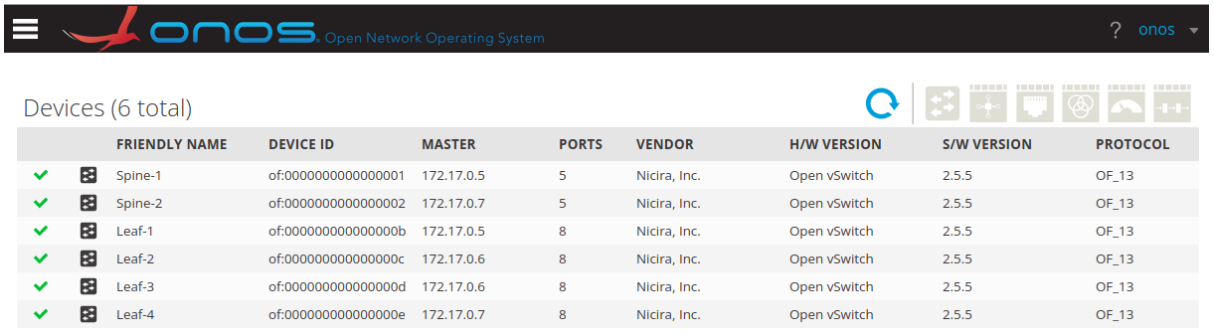


Figure 4. 64: List of Open vSwitches with their associated Master controller

If any master controller is shut down or losses connectivity from the network, a new master is elected for the switches, which were associated with that master controller. A new master is elected from the list of Standby role controllers for that individual switch. This list of Standby controllers is arranged according to the Node IDs of controllers in preference order. This enables switch to elect next best master controller in case of failure of current master controller.

The following figure shows the failure of one controller and re-election of new master controller for the switches. In the created network, a controller (172.17.0.5) was explicitly shut down and it was observed that the switches associated with that controller re-elect the new master controller (172.17.0.6).

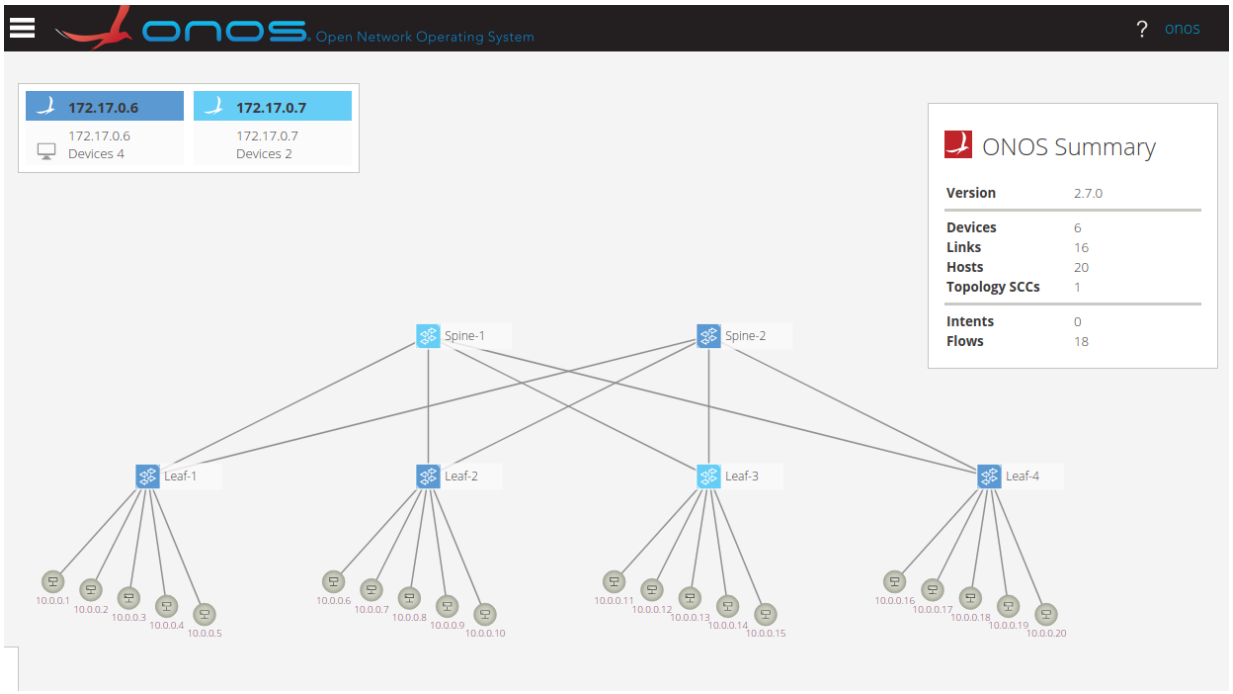


Figure 4. 65: Failure of one controller from the cluster

4.6.3 Proof and Validation of Link Failover Functioning

Previous chapters discuss the failure of controller and devices in the network, this sub-chapter will discuss the failure of links between the devices.

The links between the devices are discovered on a hop-by-hop basis by using probe frames. The probe frames are forwarded from all ports of a device containing device ID. The other end of link is discovered when the other device on another end receives this probe frame and sends it back to the controller. The *Link subsystem* of ONOS communicates with *Device subsystem* using the *LLDPLinkProvider*. The LLDPLinkProvider's *LinkDiscovery* object performs link discovery through LLDP messages [89]. After the links are discovered, the data transmission between the devices takes place mostly via Reactive Forwarding.

The Reactive Forwarding concept of SDN enables the controller to configure the path between the devices when the actual data has been transmitted from either of the device. ONOS contains an application named ***org.onosproject.fwd*** which is required to be installed and activated for data transmission to take place in the network. When the endpoints generate some traffic in the network, the first packet is encapsulated as a PacketIn message of OpenFlow and forwarded to the ONOS controller. With the help of Reactive Forwarding application, ONOS controller updates the flow rules on the Open vSwitches. It uses the Dijkstra algorithm for finding the best possible path between the endpoints [90]. The use of Intent based service in the network increases the resilience of the network.

The Intents are managed by the Intent subsystem which is explained in detail in chapter 4.2.3. The Intent subsystem acquires the information and updates about the network topology from the Topology subsystem. When the new Intents are formed between the endpoints, the Intent subsystem creates the path between the endpoints based on the information received from the Topology subsystem.

In case of link failure, which is part of Intent path, the Intent subsystem makes changes in the path based on the information received from the Topology subsystem. The controller expects the link failure information to be transferred to it through the control messages of OpenFlow protocol. On receiving this information, the Intent subsystem decides either the creation of new intents or updating the affected intents in the network [91]. This information is further results in changes in flow rules of the Open vSwitches.

In this way when a link between the Open vSwitches is shut down, ONOS detects this event and the information about the updated paths is forwarded to the Open vSwitches. To test this mechanism, the created network environment for the multiple controllers use case was utilized. An intent was created between the endpoints (10.0.0.2 and 10.0.0.18) as seen in the following figure.

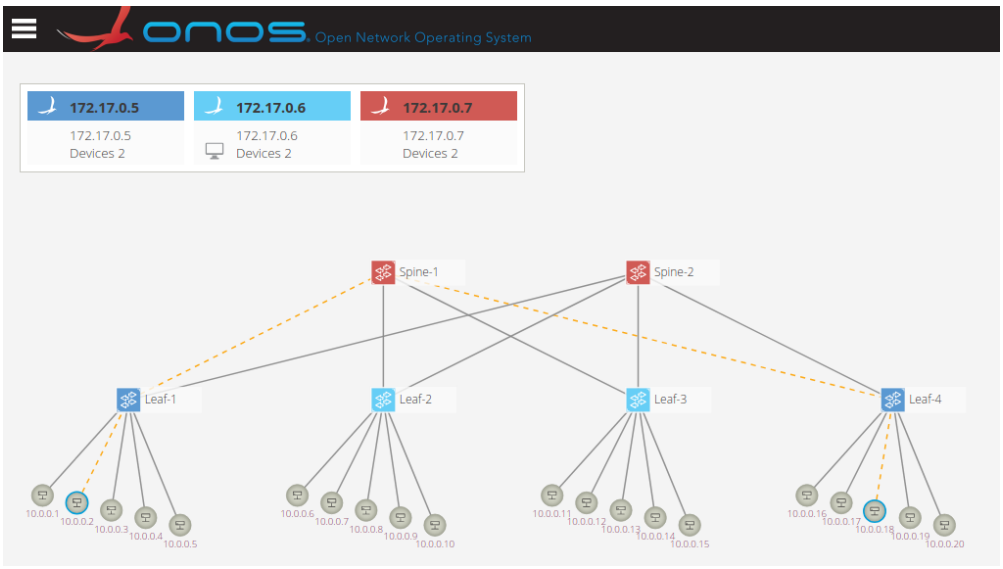


Figure 4. 66: Route from Leaf-1 to Leaf-4 created after installing an intent

The created intent was able to find the shortest path between the selected endpoints through Leaf1-Spine1-Leaf4 path as seen in the above figure. One of the links, which was part of this path, link between the Spine1 and Leaf1 was manually shut down using the Mininet command. The ONOS controller was successful in installing the new intent as seen in the following figure and updating the revised flow rules on the associated Open vSwitches.

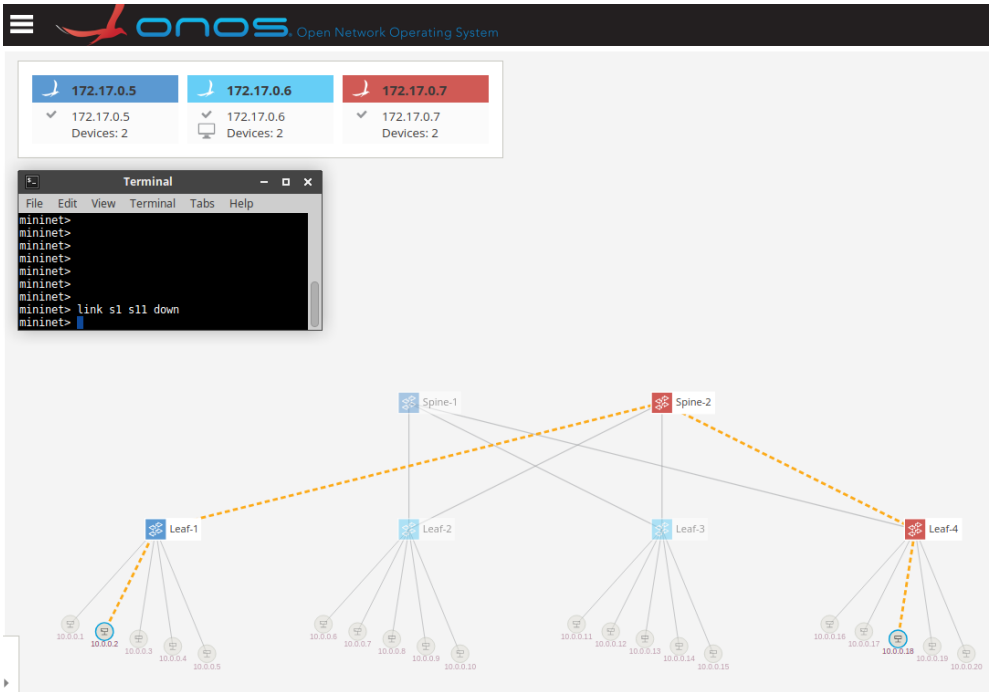


Figure 4. 67: Route change after failure of link between Spine-1 and Leaf-1

The above figure 4.67 shows the installation of new path via Leaf1-Spine2-Leaf4 Open vSwitches. The same process was repeated and another link which was part of this path, link between the Spine2 and Leaf4 was manually shut down using the Mininet command. And the ONOS controller was successful in installing again the new intent and updating the revised flow rules on the associated Open vSwitches. The following figure shows the updated path from the ONOS GUI and the ***paths*** command can be used on the ONOS CLI to find the path between the endpoints.

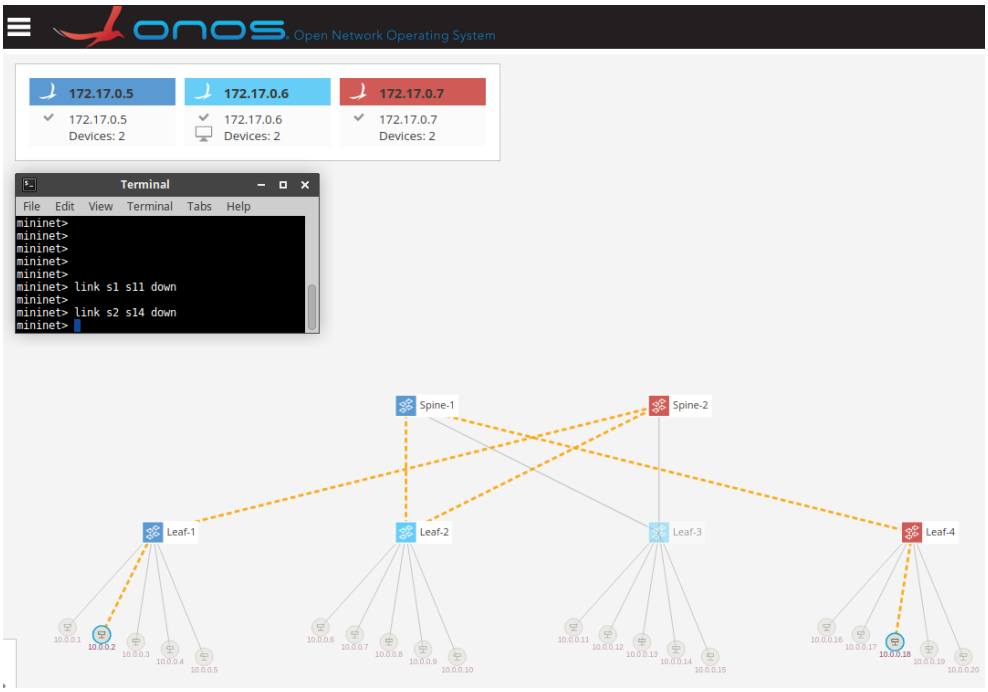


Figure 4. 68: Route change after failure of link between Spine-2 and Leaf-4

The above figure 4.68 shows the installation of new path via Leaf1-Spine2-Leaf2-Spine1-Leaf4 Open vSwitches. This shows that intents-based service increases the resilience of the network and intents are more powerful than simply installing flows. Hence, proves and validates the link failover mechanism in the network controlled by ONOS controller.

4.7 Use Case-3: Testing the ONOS Controller with IPv6 Addressing

With the rapid rise of Internet users, due to the Internet of Things (IoT) smart devices more and more devices are connected to the Internet. The IPv6 addressing scheme was introduced to overcome the restriction of limited addresses of IPv4 addressing scheme. The lack of IPv4 addresses demands the Internet Service Providers (ISPs) to migrate to the IPv6 addressing scheme. Numerous aspects are taken into consideration by service providers such as optimum cost of migration, security, quality of service, monitoring, device configurability of vertically integrated switches and routers. Software-defined Networking with its concept of centralised controlling and managing the network, promises to ease this migration to IPv6 addressing scheme with possibility to reduce human errors to benefit the new generations of networking [92]. SDN also enables the programmable networking and introduces the open standards which help ISPs to move out of specific vendor lock-ins.

While SDN increases the controllability of networking devices through programming and virtualization using open protocols, IPv6 improves the efficiency of the internet protocol, including routing. A software controlled IPv6 network is required to replace the already used IPv4 addresses to answer the convergence in telecommunications focusing on the migration to latest generation digital packet-based communications. However, it is not feasible to immediately migrate the current network to SDN and IPv6-based networking. The migration is a gradual process thus, to ensure a seamless transition, a good plan must be developed that considers consumer demand, capital and operating expense and traffic engineering [93].

4.7.1 Introduction

IPv6 was introduced in OpenFlow version 1.2 and some additional features were added in OpenFlow version 1.3. IPv6 address mechanism is supported by ONOS platform since version Blackbird 1.1.0. IPv6 feature is explicitly required to be configured on the ONOS controller. For activating IPv6 on ONOS controller, following configurations and applications are required to be installed and activated through ONOS CLI or ONOS GUI.

Applications:

1. Proxy ARP — APP ID: org.onosproject.proxyarp
2. Reactive Forwarding — APP ID: org.onosproject.fwd
3. IPv6 RA (Route Advertisement) Generator — APP ID: org.onosproject.routeadvertisement
4. Host location Provider — APP ID: org.onosproject.hostprovider

Configurations:

HostLocationProvider component

onos> cfg set org.onosproject.provider.host.impl.HostLocationProvider requestIpv6NdpRsRa true

onos> cfg set org.onosproject.provider.host.impl.HostLocationProvider requestIpv6ND true

NeighbourResolutionManager component

onos> cfg set org.onosproject.net.neighbour.impl.NeighbourResolutionManager ndpEnabled true

IPv6 Reactive Forwarding component

onos> cfg set org.onosproject.fwd.ReactiveForwarding ipv6Forwarding true

onos> cfg set org.onosproject.fwd.ReactiveForwarding matchIpv6Address true

The detailed steps for enabling IPv6 addressing scheme on ONOS controller mentioned here [76] are outdated and the above configuration was manually found through the component settings on the ONOS GUI.

For testing the ONOS controller functionality with the IPv6 addressing this use case was implemented. In this use case, the network was created consisting of four Open vSwitches and four endpoints (hosts) in the GNS3 emulation software. The network was simultaneously tested for both IPv6 and IPv4 addressing scheme. The NAT interface was used to connect the Open vSwitches with the ONOS controller which was installed and running on VM outside the GNS3 environment. All the software versions are same as mentioned in Table 4.1. The following figure shows the implemented network consisting of four endpoints, two endpoints (H1 and H4) just using IPv6 addressing scheme and two endpoints (H2 and H3) using both IPv6 and IPv4 addressing scheme.

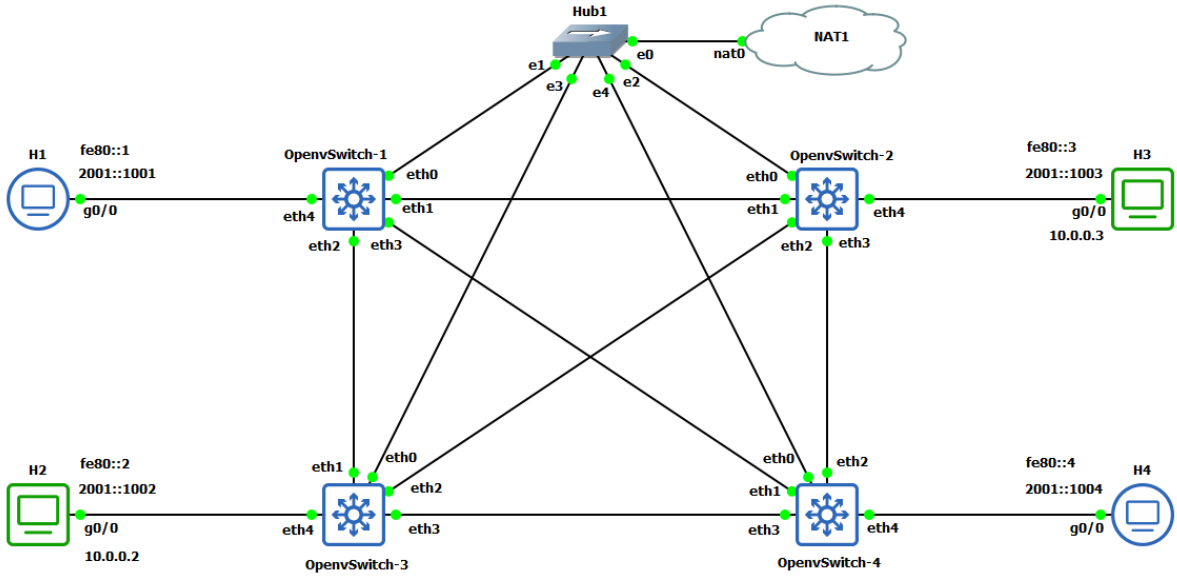


Figure 4. 69: Topology created in the GNS3 with IPv4 and IPv6 endpoints

After successful configuration and booting up the network in GNS3 software, ONOS controller was able to identify the IPv6 as well as IPv4 endpoints in the network. The following figure displays the view of created network from the ONOS GUI.

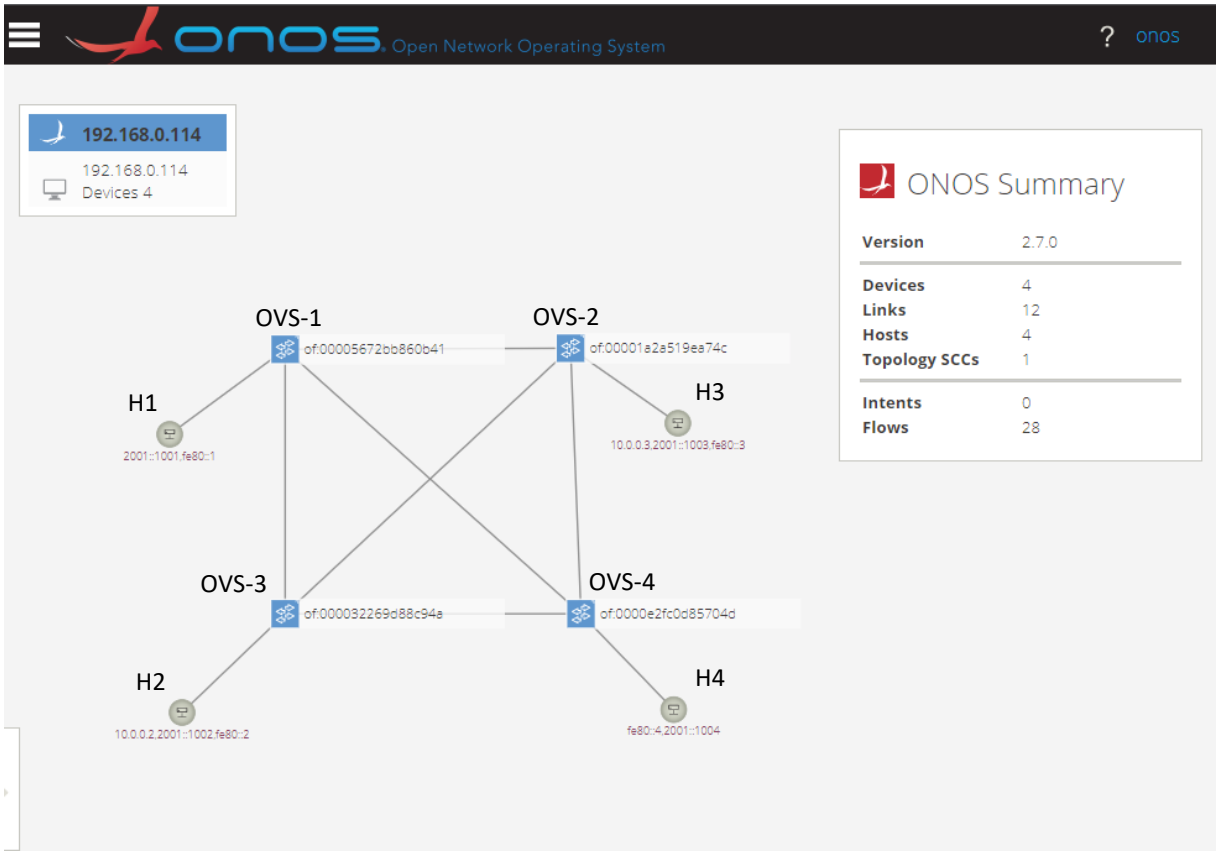


Figure 4. 70: Topology view on ONOS GUI with IPv4 and IPv6 endpoints

As seen in the above figure, the ONOS controller accurately identifies both the IPv6 and IPv4 addressing schemes and also Link-local and Global Unicast IPv6 addresses for all the endpoints.

When any new IPv6 data packet arrives at the any switchport for the first time, Open vSwitch sends the Neighbour Solicitation packet encapsulated in OpenFlow packet to the ONOS controller. The Neighbour Solicitation packet contains the Target Address (TA) of the IPv6 data packet. ONOS controller forwards this Neighbour Solicitation packet to all the connected Open vSwitches in the network. The Open vSwitch, which is connected to endpoint of the mentioned Target Address (TA) send back the Neighbour Advertisement packet to the ONOS controller mentioning that requested Target Address (TA) is connected to itself. ONOS controller forwards this Neighbour Advertisement packet to the Open vSwitch which generated the Neighbour Solicitation packet. All the Neighbour Discovery packets are encapsulated in OpenFlow packet and exchanged between Open vSwitches and ONOS controller.

The Neighbour Discovery of IPv6 addresses on ONOS controller was captured in the network using the Wireshark application. The ICMPv6 pings were forwarded in the network and subsequently related packets were captured.

In the following figure 4.71, the sequence of packet transmission in the network is displayed. Source and destination endpoints (H1 and H4) of ICMPv6 packet, all Open vSwitches and the ONOS controller nodes are listed at the top of the figure along with their IP addresses. (Note: the IP addresses of Open vSwitches are the addresses of eth0 management interface connected with the ONOS controller). All the transmissions are numbered according to the sequence of occurrence in the figure and followed by the explanation. The IPv6 packets exchanged between endpoints and Open vSwitches are highlighted in red colour, the OpenFlow packets exchanged between Open vSwitches and ONOS controller are highlighted in blue colour and the IPv6 packets exchanged between Open vSwitches are highlighted in green colour. The Neighbour Solicitation and the Neighbour Advertisement packets contains the Target Address (TA).

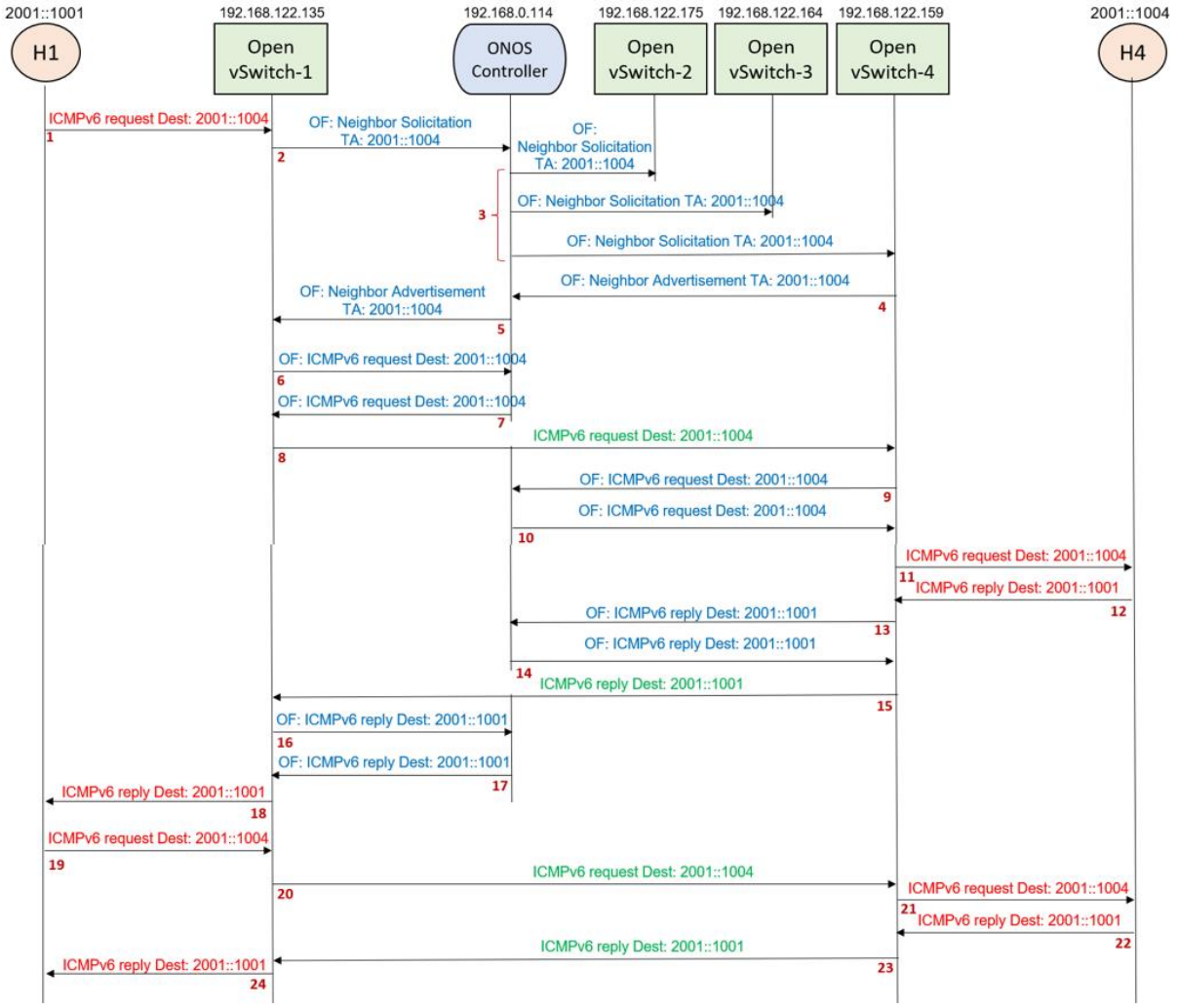


Figure 4. 71: Packets exchanged between network devices for Neighbour Discovery

The ICMPv6 data packets were transmitted in this use case as seen in the above figure. The ICMPv6 request packet was generated between the endpoints H1 (IPv6 address: 2001::1001) and H4 (IPv6 address: 2001::1004) containing Source Address: 2001::1001 and Destination Address: 2001::1004. This ICMPv6 request packet (1) was forwarded to eth4 interface of Open vSwitch-1. Since the Open vSwitch-1's flow table doesn't have any rule for this IPv6 address, the Open vSwitch-1 sent the Neighbour Solicitation packet encapsulated in OpenFlow packet (2) to the ONOS controller containing the Target Address (TA) of the H4. ONOS controller forwarded this Neighbour Solicitation packet (3) to all the connected Open vSwitches in the network. Since H4 is connected to Open vSwitch-4, it sent the Neighbour Advertisement (4) packet mentioning that requested Target Address (TA) of H4 is connected to itself and send this packet encapsulated in OpenFlow packet to the ONOS controller. ONOS controller forwards this Neighbour Advertisement packet (5) to the Open vSwitch-1.

Since the IPv6 packet of new type (ICMPv6 request) had arrived at the switchport, the Open vSwitch-1 forwarded the ICMPv6 request packet encapsulated in OpenFlow packet (6) type OFPT_PACKET_IN to the ONOS controller. The ONOS controller sent the OpenFlow packet (7) type OFPT_PACKET_OUT to the Open vSwitch-1 containing information to forward the ICMPv6 request packet from port 3 (eth3 interface). Also, the flow rule was listed in the flow table of Open vSwitch-1 for this destination address of H4. The Open vSwitch-1 forwarded the ICMPv6 request packet (8) to the Open vSwitch-4. The Open vSwitch-4 also didn't have any flow rules for this destination address and it forwarded ICMPv6 request packet encapsulated in OpenFlow packet (9) type OFPT_PACKET_IN to the ONOS controller. The ONOS controller sent the OpenFlow packet (10) type OFPT_PACKET_OUT to the Open vSwitch-4 containing information to forward the ICMPv6 request packet from port4 (eth4 interface). Also, the flow rule was listed in the flow table of Open vSwitch-4 for this destination address of H4. The Open vSwitch-4 forwarded the ICMPv6 request packet (11) from the port 4 to the endpoint H4. H4 generated the ICMPv6 reply packet (12) containing Source Address: 2001::1004 (H4) and Destination Address: 2001::1001 (H1) and forwarded to Open vSwitch-4.

After the ICMPv6 reply packet arrived at Open vSwitch-4, Open vSwitch-4 checked its flow rules and didn't find any flow rules listed for the destination address of H1 and this ICMPv6 reply packet was forwarded to the ONOS controller encapsulated in OpenFlow packet (13). ONOS controller replied with OpenFlow packet (14) to forward the ICMPv6 reply packet from port 1 and correspondingly updated the flow rule for this destination address on the Open vSwitch-4. Same process was done by the Open vSwitch-1 when the ICMPv6 reply packet arrived at its switchport.

Flow rules were installed for both the destination addresses of H1 and H4 on both the switches, Open vSwitch-1 and Open vSwitch-4 connected to these endpoints. When the next ICMPv6 request packet arrived at the Open vSwitch-1 for the destination of H4, the flow tables were checked for the matching forwarding flow rule. The flow rule was found with the match-fields of ingress port, source, and destination MAC address and therefore, the packet was directly forwarded to Open vSwitch-4.

The following figure displays the Neighbour Solicitation packet encapsulated in OpenFlow packet. This OpenFlow packet was captured using the Wireshark listening on the link between the Open vSwitch-1 and ONOS controller. At the initial phase of Neighbour Discovery, the Neighbour Solicitation packet encapsulated in OpenFlow packet was forwarded by Open vSwitch-1 to ONOS controller to get the information about the destination address. The following packet is corresponding to the packet number 2 in figure 4.71. The encapsulated OpenFlow packet contained the match-parameters providing the information to ONOS controller that the Neighbour Solicitation packet was received on switchport 4 (eth4) by the Open vSwitch-1.

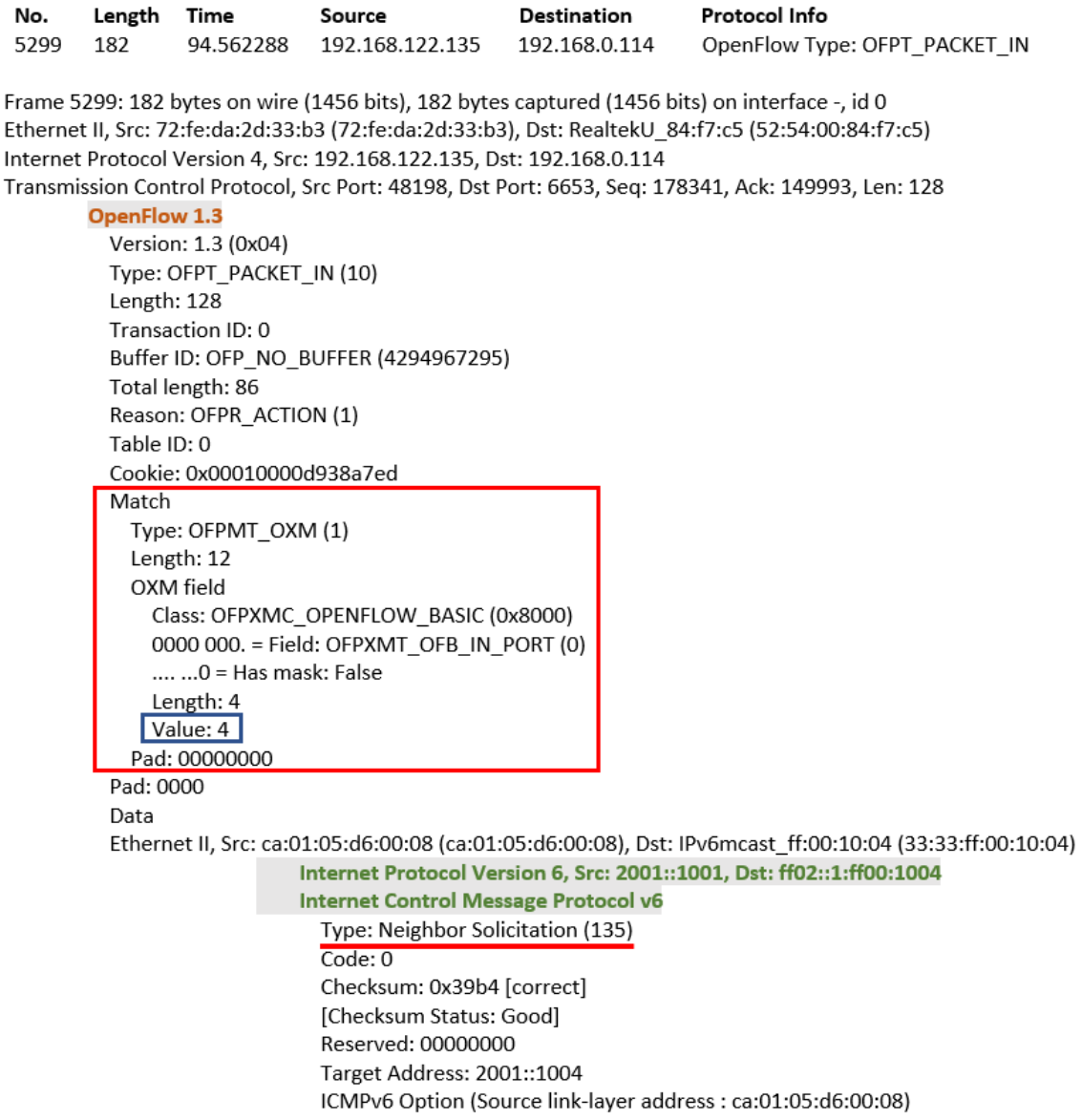


Figure 4. 72: Neighbour Solicitation packet captured on the Wireshark listening between Open vSwitch-1 and ONOS controller

After Neighbour Discovery, the Open vSwitch-1 sent the ICMPv6 request packet encapsulated in OpenFlow packet to the ONOS controller. This OpenFlow OFPT_PACKET_IN packet was sent by Open vSwitch-1 to get the information about ICMPv6 request packet processing. The ONOS controller replied with the OpenFlow OFPT_PACKET_OUT packet containing the appropriate action to be taken by the Open vSwitch-1. The following figure displays one such OpenFlow OFPT_PACKET_OUT packet. This OpenFlow packet was captured using the Wireshark listening in the link between the Open vSwitch-1 and ONOS controller. The ONOS controller replied to the Open vSwitch-1 with the OpenFlow OFPT_PACKET_OUT packet containing the action-parameter. This action-parameter informed the Open vSwitch-1 to forward the ICMPv6 request packet from the port 3 (eth3). The following packet is corresponding to the packet number 7 in figure 4.71.

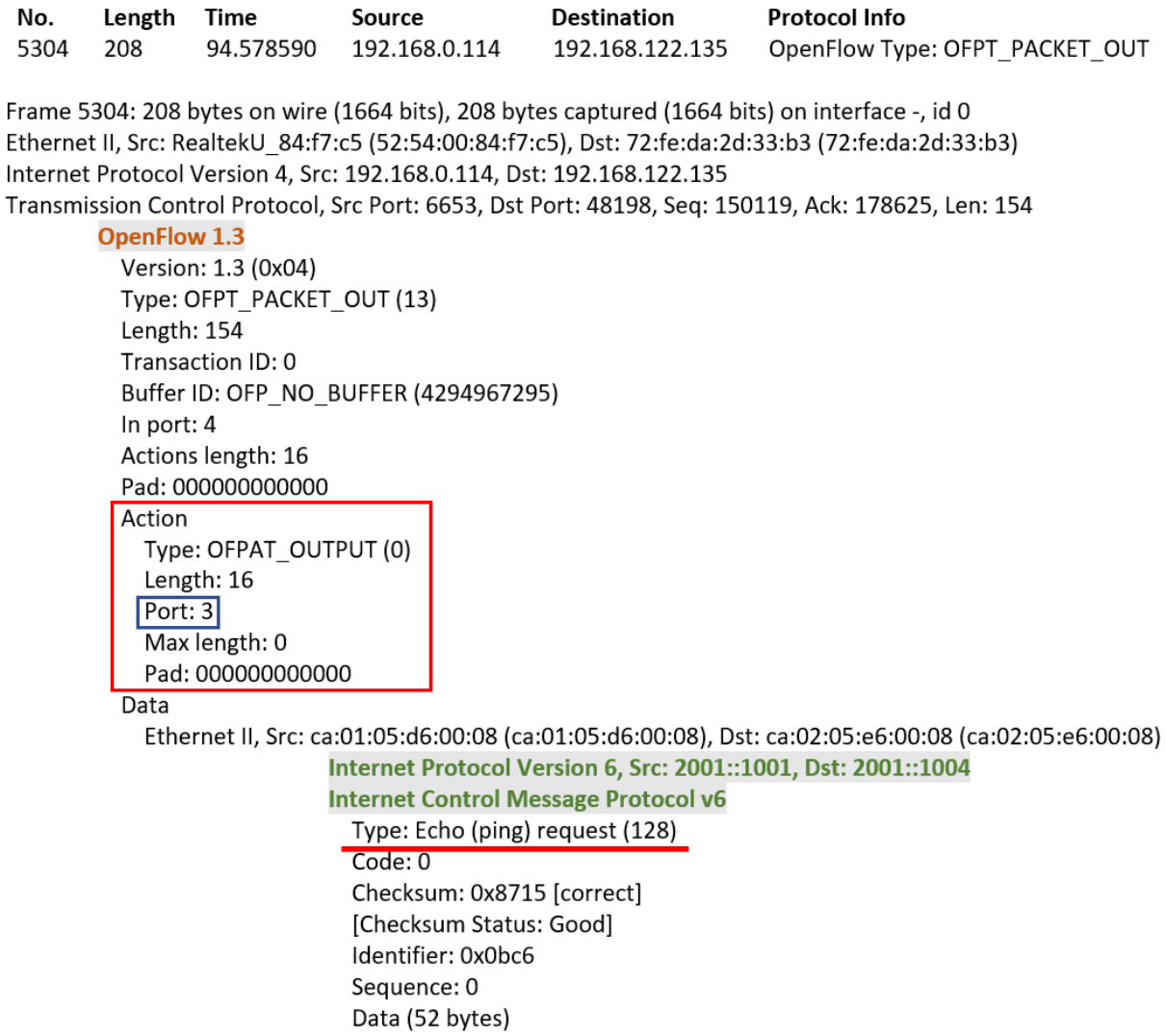


Figure 4. 73: OpenFlow OFPT_PACKET_OUT packet captured on the Wireshark listening between Open vSwitch-1 and ONOS controller

The Reactive Forwarding application enabled with IPv6 forwarding and matchIpv6Address component takes care of installing the flow rules on Open vSwitches when the IPv6 packets are transmitted in the network. When the first packet arrives at the Open vSwitch, that packet is encapsulated in the OpenFlow packet and sent to the ONOS controller. ONOS controller then installs the appropriate flow rules on the Open vSwitch with the help of Reactive Forwarding application.

The alternative to this method is installing intents between the endpoints which further installs the flow rules on Open vSwitches. This method installs the flow rules on Open vSwitches even before any data packet for the associated traffic is arrived at the Open vSwitch.

4.7.2 IPv6 over IPv4 tunnelling

One of the issues faced by ISPs while migrating the network from IPv4 to IPv6 addressing scheme is integrating the IPv6 subnets with IPv4 subnets. Migrating a IPv4 addressing network to IPv6 addressing is not a single step process. The migration takes place in specific stages, beginning with developing small subnets of IPv6 addresses within the IPv4 network and progressively transferring all the services to IPv6 in the network [94] [95] [96]. But during this transition, the IPv6 subnets should be able to communicate over the IPv4 infrastructure. The IPv6 subnets can be configured to transmit packets over IPv4 network by *IPv6 over IPv4 tunnelling* [97]. IPv6 over IPv4 tunnels are point-to-point tunnels created by encapsulating IPv6 packets within IPv4 headers to carry them over IPv4 network.

In this use case, another network was implemented in GNS3 environment to test the functioning of ONOS controller for IPv6 over IPv4 tunnelling. The isolated IPv6 subnets were created and IPv4 network was configured between these subnets. The Open vSwitches and ONOS controller were located in the IPv4 network. The Cisco routers (Cisco IOS Software, 7200 Software (C7200-ADVENTERPRISEK9-M), Version 12.4(24)T5) were utilized to configure the IPv6 over IPv4 tunnels.

The following figure displays the created network in the GNS3 environment. Two isolated IPv6 subnets (2010::/104 and 2020::/104) were created containing a Router and an endpoint in each subnet. An IPv4 network (highlighted in yellow) consisting of four Open vSwitches and connecting to the ONOS controller through NAT interface was created between these IPv6 subnets. The IPv6 over IPv4 tunnel was configured between the Router R1 and Router R2.

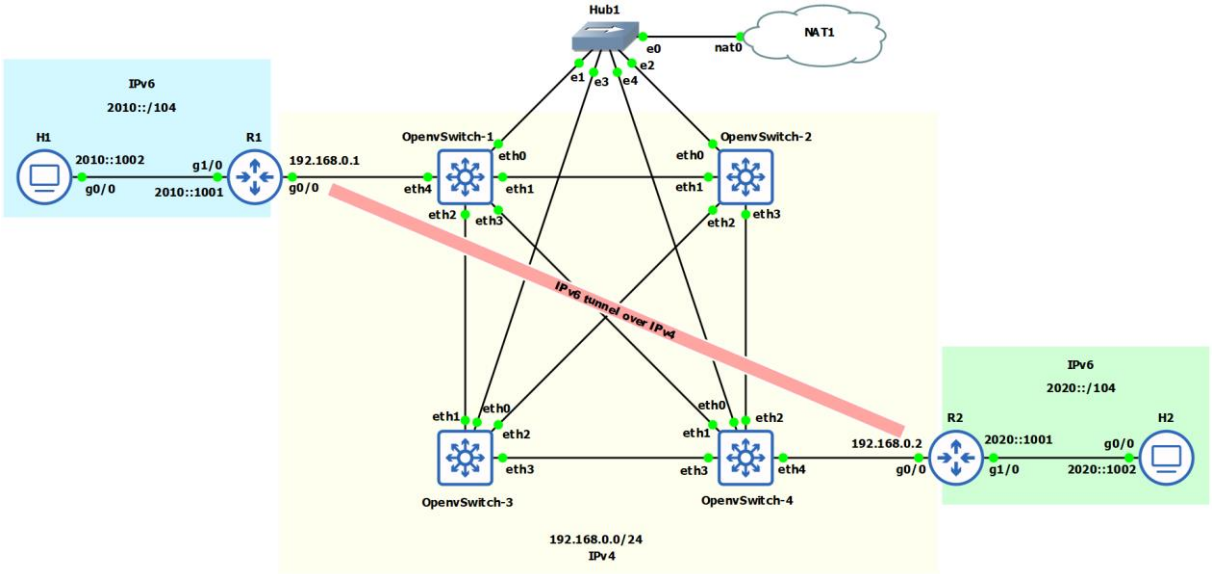


Figure 4. 74: Topology created in the GNS3 with IPv6 tunnelling over IPv4 network

The installation of required applications and configurations for enabling IPv6 was done on the ONOS controller. The tunnel interfaces were configured with the IPv6 address 2000::1/104 at router R1's interface g0/0 and 2000::2/104 at router R2's interface g0/0. Tunnel mode 6to4 (ipv6ip) and RIPng routing protocol [98] was configured to advertise the IPv6 addresses through the tunnel. The following snippet shows the Tunnel interface configuration on router R1.

```
R1#sh run | sec interface Tunnel0
interface Tunnel0
 no ip address
 ipv6 address 2000::1/104
 ipv6 rip sdn enable
 tunnel source GigabitEthernet0/0
 tunnel destination 192.168.0.2
 tunnel mode ipv6ip
```

Figure 4. 75: IPv6 Tunnel configuration on the router R1

After successful configuration of tunnel and enabling IPv6 services on ONOS controller, ONOS controller was able to set up the IPv6 tunnel over IPv4 network. The tunnel was configured with the source IPv4 address of router R1's interface g0/0 and the destination IPv4 address of router R2's interface g0/0 as seen in figure 4.75. The successful installation of tunnel can be confirmed from either of the routers console at the end interfaces of tunnel. The following figure displays all the IPv6 routes learnt by the router R1. The highlighted route is the route to IPv6 subnet located at the other end of the tunnel and therefore, the tunnel creation was confirmed. The route to the IPv6 subnet located at the other end of the tunnel were successfully advertised over the tunnel and learned by the router R1 through RIPng routing protocol.

```
R1# sh ipv6 route
IPv6 Routing Table - Default - 6 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, M - MIPv6, R - RIP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external
       O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
       ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
C   2000::/104 [0/0]
     via Tunnel0, directly connected
L   2000::1/128 [0/0]
     via Tunnel0, receive
C   2010::/104 [0/0]
     via GigabitEthernet1/0, directly connected
L   2010::1001/128 [0/0]
     via GigabitEthernet1/0, receive
R   2020::/104 [120/2]
     via FE80::C0A8:2, Tunnel0
L   FF00::/8 [0/0]
     via Null0, receive
```

Figure 4. 76: Routes advertised over the configured tunnel as seen on Router R1

Host PCs in both the IPv6 subnets were configured with the gateway address of respective routers i.e., host H1 (2010::1002) with gateway of router R1 (2010::1001) and host H2 (2020::1002) with gateway of router R2 (2020::1001). After the successful creation and installation of IPv6 tunnel over IPv4 network, the hosts were able to transfer traffic to each other.

The ICMPv6 ping request was generated from the host H1 to the destination host H2. This request was forwarded from the host H1 to router R1. Router R1 encapsulated the received ICMPv6 ping request in IPv4 packet and forwarded it to the Open vSwitch-1. The Open vSwitch-1 not knowing how to process the received packet from router R1, switch encapsulated the received packet into OpenFlow packet and sent it to the ONOS controller. This OpenFlow OFPT_PACKET_IN packet was sent by Open vSwitch-1 to get the information about received packet request packet processing. The ONOS controller replied with the OpenFlow OFPT_PACKET_OUT packet containing the appropriate action to be taken by the Open vSwitch-1. The following figure 4.77 displays one of the OpenFlow OFPT_PACKET_OUT packet. This OpenFlow packet was captured using the Wireshark listening in the link between the Open vSwitch-1 and ONOS controller. The ONOS controller replied to the Open vSwitch-1 with the OpenFlow OFPT_PACKET_OUT packet containing the action-parameter. This action-parameter commands the Open vSwitch-1 to forward the received packet from the port 3 (eth3).

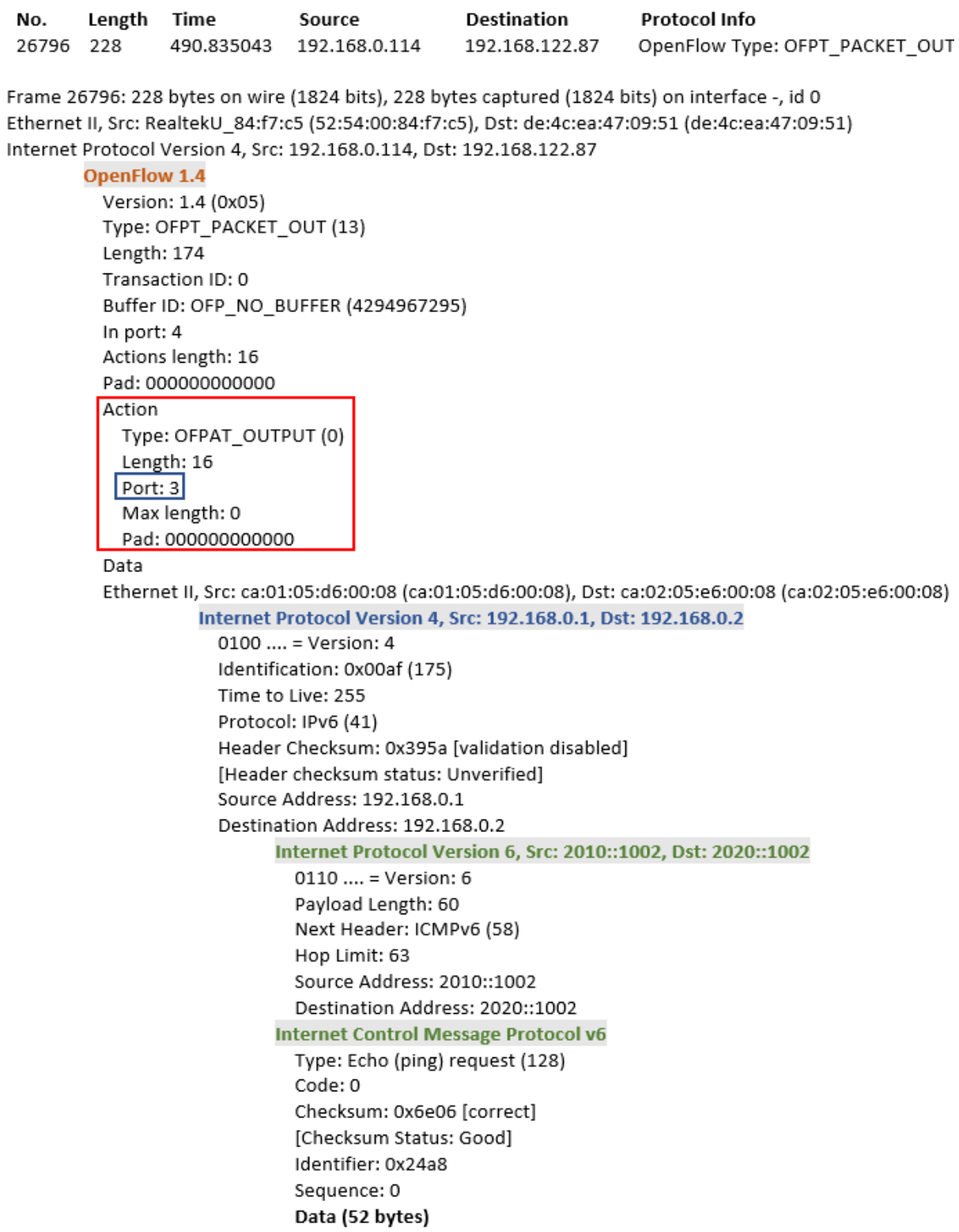


Figure 4. 77: IPv6 packet encapsulated in IPv4 packet over tunnel and that packet encapsulated in OpenFlow packet captured on Wireshark

The successful creation and installation of IPv6 tunnel over IPv4 network can be seen in the above figure. The IPv6 data packets are encapsulated in the IPv4 packets and transferred over the tunnel. Open vSwitches use OpenFlow protocol encapsulating the tunnel packets in itself to inquire the ONOS controller about forwarding of the packets. ONOS controller with its available features, is compatible with various services of IPv6 addressing scheme and can benefit the migration of IPv4 networks to the IPv6 networks.

4.8 Use Case-4: Integrating Software-defined Network with the Legacy Networks

After evaluating various aspects of SDN through ONOS controller, concluding chapter was to evaluate the process of integrating software-defined network with the legacy networks consisting of coupled control and data plane network devices.

The legacy networks consist of several Autonomous Systems (AS) and are mostly heterogeneous in nature which leads to the problem of multi-domain heterogeneous network migration planning. Implementing software-defined network with the legacy network comes with other challenging problems like lack of a high maturity level of SDN-based standards, insufficient availability of skilled human resources, technical as well as business perspectives, higher costs of network migration to replace or upgrade network devices. Due to these reasons, most of the ISPs evade the idea to migrate to technologies like SDN.

However, to overcome the management and controlling issues of legacy networks, SDN is the feasible solution. The migration can be kicked off by first transferring the Private customer networks into the SDN and later gradually implementing SDN in core public networks. In the Public networks, the migration can be initiated by setting up hybrid networks of legacy networks with SDN and eventually transferring the network completely into SDN [99].

4.8.1 Introduction

With the applications developed for ONOS controller, it supports the migration process of legacy networks to the SDN. ONOS controller is capable of performing and exchanging the routing information with the legacy routers using the SDN-IP application. SDN-IP is an ONOS application developed and implemented over ONOS controller to connect to the external networks on the legacy networks using the standard Border Gateway Protocol (BGP) [100]. The SDN network acts as an Autonomous System (AS) to the legacy routers in the network. The SDN network consists of Open vSwitches, ONOS controller and BGP-Speakers. BGP-Speakers are used to establish the peering with the external gateway routers by acquiring route information from the ONOS controller. This configured SDN network serves as a transit network communicating with other legacy networks.

In this use case a SDN network was implemented in the GNS3 network emulator consisting of 8 Open vSwitches, one BGP-Speaker and the ONOS controller. The following figure displays the implemented network in GNS3 emulator. The ONOS controller was running inside the GNS3 emulator in this use case to set up the iBGP peering with the BGP-speaker (BGP-SP1). The eth0 management interfaces of all Open vSwitches were connected to the ONOS controller through a hub. Different Autonomous Systems were implemented (coloured ovals), each with one border gateway router and a host PC. *(More clear figure can be found in Appendix.)*

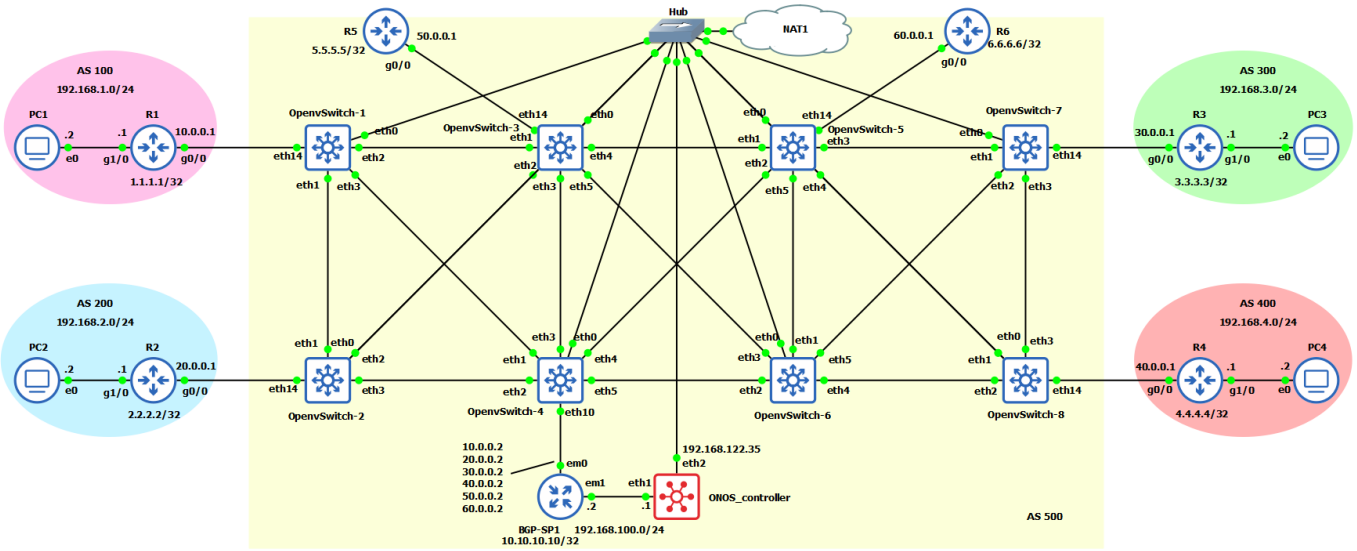


Figure 4. 78: Topology created in the GNS3 with different legacy networks and SDN network

To test the integration process of SDN network with the legacy networks through ONOS controller, this use case was created consisting of five different AS. Four IPv4 legacy networks were created, AS 100 with border gateway router R1, AS 200 with border gateway router R2, AS 300 with border gateway router R3 and AS 400 with border gateway router R4. The Cisco routers (Cisco IOS Software, 7200 Software (C7200-ADVENTERPRISEK9-M), Version 12.4(24)T5) were utilized as border gateway routers for these legacy networks. The SDN network was created in separate AS 500 consisting of 8 Open vSwitches, BGP-Speaker (BGP-SP1) and two internal routers (R5 and R6). The BGP-Speaker was implemented on Quagga suite [101] on FreeBSD platform [102] version clang 8.0.1 working with FRRouting (version 7.2) [103] open source Internet routing protocol suite. The Open vSwitches and ONOS controller's specifications are as listed in the Table 4.1.

The border gateway routers were configured with BGP to share the routing information of their respective internal networks. These border gateway routers peer with BGP-Speaker in the SDN network and use eBGP to exchange routing information with the adjacent external network routers through BGP-Speaker. Due to limited memory resources availability on the host machine, only a single BGP-Speaker was implemented in the SDN network, however multiple BGP-Speakers can be implemented to increase the high availability of the network. The BGP-Speaker uses iBGP to communicate with the other BGP-Speakers, internal routers, and SDN-IP application on ONOS controller . The internal routers R5 and R6 are configured are advertising to advertise two different subnets located inside the SDN network. The R5 internal router advertises 50.0.0.0/24 subnet, which is also advertised to the external border routers and the R6 internal router advertises 60.0.0.0/24 subnet, which is just advertised to the internal routers present inside the SDN network (AS 500). In other words, R6 router is configured to communicate and exchange routes with R5 internal router whereas, R5 router is configured to communicate and exchange routes with external border gateway routers and as well as R6 internal router. The BGP speaker is configured to form neighbourship with the external border routers and the internal routers and just advertise the internal network connected with R5. The routes advertised by the border gateway routers to the BGP-Speaker are processed and re-advertised to the other external networks by the BGP-Speaker with the help of SDN-IP application. The best route for each destination is decided by the SDN-IP application according to the standards of iBGP and translated into an ONOS Application Intent Request. ONOS converts the Application Intent Request into forwarding flow rules on the connected Open vSwitches in the network.

The SDN-IP application offers the integration mechanism of BGP to the ONOS controller. It behaves as a regular BGP-speaker, at the protocol level and for ONOS controller, SDN-IP is just an application that uses its services to install and update the appropriate forwarding flow rules on the Open vSwitches. The SDN network created in this use case acts as a transit AS in between the deployed legacy networks. From external border router gateway's point of view, the created SDN network is just another normal AS.

The following figure displays the topology view of the configured network in this use case, perceived from the ONOS GUI. As seen in the figure, the ONOS controller successfully identifies the configured IP addresses on each external border routers of legacy networks, two internal routers and also the configured BGP-Speaker deployed in the created SDN network.

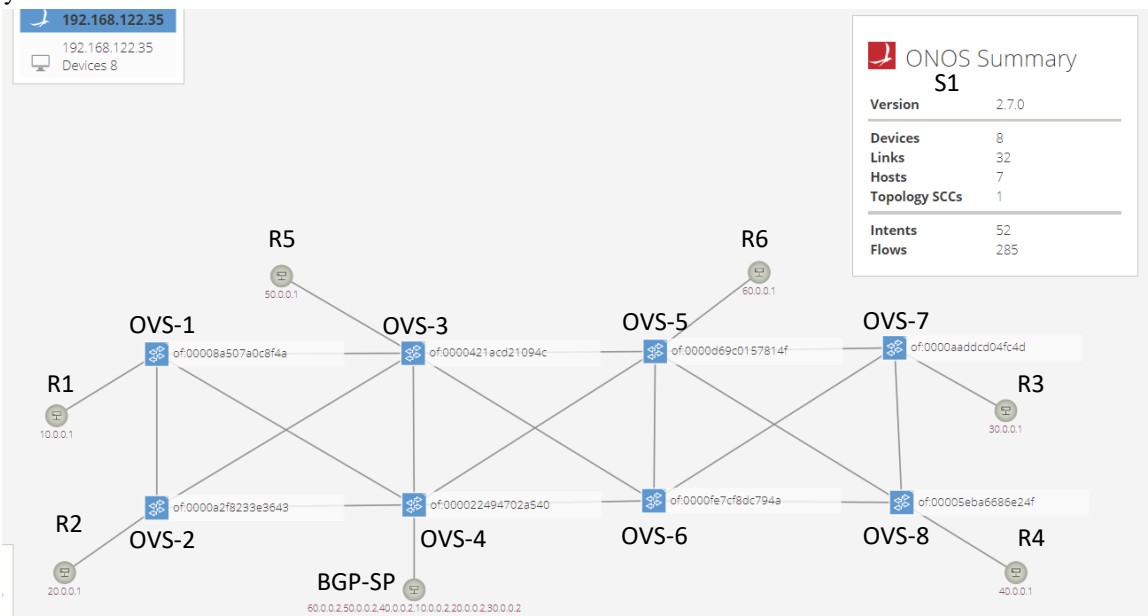


Figure 4. 79: Topology view on the ONOS GUI

4.8.2 Configuration and Working

For accurate functioning of the SDN-IP application, it requires some configuration to be set up on the ONOS controller. The configuration file, named *network-cfg-bgp.json* was created containing the exact location of external BGP border routers, internal routers and the BGP-Speaker. The appropriate BGP peering addresses are required to be configured at the switchports connecting to the routers. The complete configuration file can be found in the Appendix. BGP-Speaker also needs to be configured with the eBGP peering addresses to the border routers and with the iBGP peering addresses to the internal routers. To connect BGP-Speaker with ONOS controller, iBGP configuration at port 2000 is done so that speaker can share routes with the ONOS controller. Similarly, all the border gateway routers and internal routers are configured to peer with the BGP-Speaker and advertise their internal networks.

The BGP-Speaker was configured with FRRouting to enable IP routing services and single interface (em0) was configured with all BGP peering addresses for the routers of legacy networks and internal network. The BGP-Speaker peers in two methods:

- BGP-Speaker must have connection to the SDN data plane network, through which the BGP-Speaker can establish neighborship with external border routers using eBGP.
- BGP-Speaker must have direct connection to the ONOS controller, through which the BGP-Speaker forms iBGP neighborship with the SDN-IP application. This connectivity for establishing neighborship must be out-of-band of the SDN data plane.

For activating SDN-IP application on ONOS controller, the following applications are required to be installed and activated through ONOS CLI or ONOS GUI.

Applications:

1. Proxy ARP — APP ID: org.onosproject.proxyarp
2. Configuration Manager — APP ID: org.onosproject.config
3. SDN-IP — APP ID: org.onosproject.sdnip
4. SDN-IP Reactive Routing — APP ID: org.onosproject.reactive.routing

After successful configuration of all the devices in the network, all the legacy routers formed the eBGP peering with BGP-Speaker and the internal routers formed the iBGP peering with the BGP-Speaker in the SDN network and the routes were exchanged between them. The BGP-Speaker also formed the iBGP peering with the SDN-IP application and all the routing information was shared with the ONOS controller as seen in the figure 4.80. The border routers were configured to advertise the respective loopback addresses. The internal route 50.0.0.0/24 was configured to be advertised to the border routers, hence the next hop for that route was virtual gateway address of BGP-Speaker, discussed in detailed later in this chapter. Source node in figure 4.80, is the node IP address (192.168.122.35) for the controller's interface. This interface is connected to the NAT interface in GNS3 software as shown in figure 4.78 in order to access ONOS from outside the GNS3 software.

```
onos@root > routes
B: Best route, R: Resolved route

Table: ipv4
B R  Network           Next Hop        Source (Node)
> *  1.1.1.1/32        10.0.0.1        BGP (192.168.122.35)
> *  2.2.2.2/32        20.0.0.1        BGP (192.168.122.35)
> *  3.3.3.3/32        30.0.0.1        BGP (192.168.122.35)
> *  4.4.4.4/32        40.0.0.1        BGP (192.168.122.35)
> *  10.0.0.0/24       10.0.0.1        BGP (192.168.122.35)
> *  20.0.0.0/24       20.0.0.1        BGP (192.168.122.35)
> *  30.0.0.0/24       30.0.0.1        BGP (192.168.122.35)
> *  40.0.0.0/24       40.0.0.1        BGP (192.168.122.35)
     50.0.0.0/24       192.168.100.2   BGP (192.168.122.35)
> *  192.168.1.0/24    10.0.0.1        BGP (192.168.122.35)
> *  192.168.2.0/24    20.0.0.1        BGP (192.168.122.35)
> *  192.168.3.0/24    30.0.0.1        BGP (192.168.122.35)
> *  192.168.4.0/24    40.0.0.1        BGP (192.168.122.35)
   Total: 13
```

Figure 4. 80: Routes learnt by ONOS controller

SDN-IP Reactive Routing application is responsible for sharing the routing information of the internal network devices within the SDN network itself and also with the adjacent legacy networks. Using this application, the routers and hosts which reside inside the SDN network can communicate with other subnets within the SDN network and also with the external border routers located in other AS.

The external border routers (R1-R4) advertise the routes of their respective AS to the BGP-Speaker (BGP-SP1) in the SDN network and further these routes are re-advertised to the other external border routers of different AS, to the internal routers (R5 and R6) within the SDN network and the SDN-IP application. The SDN-IP application selects the best route for each destination according to the iBGP standards and these are translated into the ONOS Application Intent requests. ONOS controller translates these point-to-point intent requests into the flow rules, which are further implemented on the respective Open vSwitches. Moreover, multi-point-to-single-point intents are created which allows external BGP routers to peer with the BGP-Speaker. Consequently, BGP itself can also detect failures and act accordingly in case of any link failure. The SDN-IP Reactive Routing application creates the routes for internal subnets of SDN network and these are translated into the Multi-Point-to-Single-Point Intents by the ONOS Intent subsystem.

The following figure displays the part of Intents created by the ONOS Intent subsystem. It can be observed in the figure that the Intents created for the external networks advertisements are created by *org.onosproject.sdnip* and intents created for the internal networks advertisements are created by *org.onosproject.reactive.routing*.

Intents (52 total)

APPLICATION ID	KEY	TYPE	PRIORITY	STATE
6 : org.onosproject.sdnip	4.4.4.4/32	MultiPointToSinglePointIntent	260	Installed
6 : org.onosproject.sdnip	40.0.0.0/24	MultiPointToSinglePointIntent	220	Installed
6 : org.onosproject.sdnip	40.0.0.1-40.0.0.2-dst	PointToPointIntent	1000	Installed
6 : org.onosproject.sdnip	40.0.0.1-40.0.0.2-icmp	PointToPointIntent	1000	Installed
6 : org.onosproject.sdnip	40.0.0.1-40.0.0.2-src	PointToPointIntent	1000	Installed
6 : org.onosproject.sdnip	40.0.0.2-40.0.0.1-dst	PointToPointIntent	1000	Installed
6 : org.onosproject.sdnip	40.0.0.2-40.0.0.1-icmp	PointToPointIntent	1000	Installed
6 : org.onosproject.sdnip	40.0.0.2-40.0.0.1-src	PointToPointIntent	1000	Installed
181 : org.onosproject.reactive.routing	50.0.0.1/32	MultiPointToSinglePointIntent	260	Installed
181 : org.onosproject.reactive.routing	50.0.0.2/32	MultiPointToSinglePointIntent	260	Installed
181 : org.onosproject.reactive.routing	60.0.0.1/32	MultiPointToSinglePointIntent	260	Installed
181 : org.onosproject.reactive.routing	60.0.0.2/32	MultiPointToSinglePointIntent	260	Installed

Figure 4. 81: List of intents configured by SDN-IP and Reactive Routing application

Along with multiple BGP-Speakers, multiple ONOS controllers in form of cluster can be implemented to increase availability and scalability of the network. In case of implementation of multiple ONOS controllers, number of SDN-IP applications can also be increased. SDN-IP application provides the support for high availability using an active-standby model. However, among all the SDN-IP applications, only one is active and has the role of so-called Leader application and others are on standby mode. The ONOS Leader election service is utilized by the SDN-IP application to elect the Leader. Each SDN-IP application receives the routing information from the external BGP routers but only the Leader is responsible for creating the Intent requests on all the ONOS controllers. In case of failure of the Leader SDN-IP application, a new Leader is chosen from the other available SDN-IP applications on Standby mode. The elected new Leader performs the Intent synchronization to ensure all the intents are installed as per the current BGP routes learnt.

However, due to limited availability of memory resources on the host machine used, only a single instance of ONOS controller and a single instance of BGP-Speakers was implemented for testing this use case. The following figure shows the list of BGP-Speakers and BGP neighbours learnt through iBGP peering with the ONOS controller. The BGP neighbour seen here is BGP-Speaker (BGP-SP1), configured with BGP ID 10.10.10.10.

```
onos@root > bgp-speakers
speaker1: port=of:000022494702a540/10, vlan=None, peers=[20.0.0.1, 30.0.0.1, 10.0.0.1, 40.0.0.1]
onos@root > bgp-neighbors
BGP neighbor is 10.10.10.10, remote AS 500, local AS 500
  Remote router ID 10.10.10.10, IP /192.168.100.2:12910, BGP version 4, Hold time 180
  Remote AFI/SAFI IPv4 Unicast YES Multicast NO, IPv6 Unicast NO Multicast NO
  Local  router ID 192.168.100.1, IP /192.168.100.1:2000, BGP version 4, Hold time 180
  Local  AFI/SAFI IPv4 Unicast YES Multicast NO, IPv6 Unicast NO Multicast NO
  4 Octet AS Capability: Advertised Received
```

Figure 4. 82: BGP-Speaker and BGP neighbours of ONOS controller

The BGP-Speaker was able to successfully form the iBGP peering with SDN-IP application and eBGP peering with the external BGP routers as displayed in the following figure. Total of five neighbours were connected to the BGP-Speaker, four external BGP routers and one ONOS controller.

```
bgpsp1# sh ip bgp summary

IPv4 Unicast Summary:
BGP router identifier 10.10.10.10, local AS number 500 vrf-id 0
BGP table version 25
RIB entries 25, using 4600 bytes of memory
Peers 5, using 67 KiB of memory

Neighbor        V         AS MsgRcvd MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd
10.0.0.1        4        100     123     135        0    0    0 02:05:22            3
20.0.0.1        4        200     124     138        0    0    0 00:10:51            3
30.0.0.1        4        300     118     138        0    0    0 00:10:51            3
40.0.0.1        4        400      97     135        0    0    0 02:05:21            3
192.168.100.1   4        500     127     135        0    0    0 02:05:29            0

Total number of neighbors 5
```

Figure 4. 83: BGP neighbours of BGP-Speaker

After successful installation and configuration of the SDN network devices, the external routers were able to exchange and learn routes from the other Autonomous Systems. The BGP border routers of each legacy network were configured to exchange the routes for connected internal network and the same can be confirmed from one of the border routers consoles as seen in the following figure. Also, as configured the internal subnet (50.0.0.0/24) connected to the R5 router is advertised but not the subnet connected to R6.

```
R1#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     1.0.0.0/32 is subnetted, 1 subnets
C       1.1.1.1 is directly connected, Loopback0
     50.0.0.0/24 is subnetted, 1 subnets
B       50.0.0.0 [20/0] via 10.0.0.2, 01:58:52
     2.0.0.0/32 is subnetted, 1 subnets
B       2.2.2.2 [20/0] via 10.0.0.2, 01:18:08
     3.0.0.0/32 is subnetted, 1 subnets
B       3.3.3.3 [20/0] via 10.0.0.2, 01:58:52
     4.0.0.0/32 is subnetted, 1 subnets
B       4.4.4.4 [20/0] via 10.0.0.2, 01:58:52
     20.0.0.0/24 is subnetted, 1 subnets
B       20.0.0.0 [20/0] via 10.0.0.2, 01:18:08
B    192.168.4.0/24 [20/0] via 10.0.0.2, 01:58:52
     40.0.0.0/24 is subnetted, 1 subnets
B       40.0.0.0 [20/0] via 10.0.0.2, 01:58:52
     10.0.0.0/24 is subnetted, 1 subnets
C       10.0.0.0 is directly connected, GigabitEthernet0/0
C    192.168.1.0/24 is directly connected, GigabitEthernet1/0
B    192.168.2.0/24 [20/0] via 10.0.0.2, 01:18:09
B    192.168.3.0/24 [20/0] via 10.0.0.2, 01:58:52
     30.0.0.0/24 is subnetted, 1 subnets
B       30.0.0.0 [20/0] via 10.0.0.2, 01:58:52
```

Figure 4. 84: Routes learnt by the BGP border router R1

The internal PCs in each AS were configured with border router as their gateway and therefore, all the PCs were able to transfer packets to each other. The following figure shows that the PC1 from AS 100 was able to reach the PC4 located in the AS 400 via the transit SDN network.

```
PC1> trace 192.168.4.2
trace to 192.168.4.2, 8 hops max, press Ctrl+C to stop
 1   192.168.1.1   8.649 ms  9.972 ms  10.547 ms
 2   40.0.0.1   30.301 ms  29.679 ms  30.432 ms
 3   *192.168.4.2   51.541 ms (ICMP type:3, code:3, Destination port unreachable)
```

Figure 4. 85: Packet transmission from PC1 in AS 100 to PC4 in AS 400

The SDN-IP application is capable of creating the transit SDN network for the adjacent legacy networks. The received routes are processed and translated into Intent requests, which further are converted into the flow rules for Open vSwitches. All this works perfectly fine for exchanging routes between the external BGP routers, but SDN-IP application is not capable of routing any internal subnet which resides inside the SDN network. For this *SDN-IP Reactive Routing* application was implemented. This application is responsible for sharing the routing information of the internal network devices within the SDN network itself and also with the adjacent legacy networks. Using the SDN-IP Reactive Routing application, the routers and hosts which reside inside the SDN network can communicate with other subnets within the SDN network and also with the external BGP routers located in other AS.

The Virtual Gateway module of ONOS controller is implemented so that the internal network devices in SDN network know where to forward their packets to a next hop. The Virtual Gateway module maintains all the ARP information in its registry, which are learnt from the ONOS controller. When the ARP request packet of virtual gateway address arrives, this module creates the ARP reply packet and send it to the requesting network device. The MAC address of the BGP-Speaker is configured as the Virtual Gateway's MAC address. In order to exchange the routing information of the external legacy networks with the internal SDN network and vice versa, the BGP-Speaker needs to be configured to advertise the routing information of these networks. Similarly, for sharing this routing information with the ONOS controller, the ONOS needs to be configured with SDN-IP Reactive Routing application mentioning the accurate IP prefixes of each deployed network. The network subnets which are to be advertised to the external networks are stated as the public IP prefix and network subnets which are to be advertised only within the internal SDN network are stated as the private IP prefix. The BGP-Speaker needs to be configured with these public IP prefixes so that external BGP routers can reach to these subnets inside the SDN network. The detailed configuration file can be found in the Appendix.

The SDN-IP application simply receives and analyses BGP routing data from the BGP-Speakers, but it never creates or retransmits the BGP routes. This application performs as a subset of the iBGP protocol. The iBGP peering is between the BGP-Speakers and the SDN-IP application instances. Note that iBGP peering is not required between the SDN-IP application instances. There is no direct communication between the SDN-IP application instances, the Leader election service of ONOS decides the leader among SDN-IP applications and the elected application handles the complete functionality of SDN-IP application among the cluster.

In order for Internet Service Providers to have a smooth integration of SDN network with the legacy networks, SDN-IP application on ONOS provides the platform for all necessary aspects in terms of routing and interoperability. This application provides essential components the ISPs need while migrating from the legacy IP networks to the Software-defined networks and strength the case for using SDN-centric approach to migration. SDN fits as best option for migrating the networks as it opens up a number of possibilities for smart networking that is more effective at managing and running networks.

5 Summary and Perspectives

The aim of this thesis was to study the functionalities of Software-defined Networks (SDN) and practically develop the real-world resembling environment in the network emulator environment. Research about open-source SDN controllers was conducted to select the best controller from the available options. Three SDN controllers, ONOS, OpenDaylight and Ryu were found to be the well-known SDN controllers amongst the academic researchers and developers of the SDN. These three controllers were studied in detail in terms of their functionality and the services they support and provide. Correspondingly two open-source network emulator environments, GNS3 and Mininet, were selected which support the software versions of virtual as well as real-world network devices.

Research about different deployment architectures of the SDN was carried out in terms of their structures, functional differences, difficulties they promise to overcome and their deployment scenarios. From the studied architectures and selected SDN controllers for this thesis, it was found that Ryu was developed on the physically-centralised architecture where as ONOS and OpenDaylight controllers were developed on the physically-distributed and logically-centralised architecture of SDN. In other words, ONOS and OpenDaylight controllers were much easier to implement as a cluster of multiple controllers compared to the Ryu controller. All the three selected SDN controllers were installed and configured to test their functionality in respect to network services. Even though the large amount of documentation is available for all these SDN controllers, the updated versions of their software were found to be modified with several modules and these changes are yet to be well documented. During the testing of these SDN controllers those changes were identified, and also other difficulties are documented in the section of their utilization in this thesis.

After complete installation and configuration of these SDN controllers, the detailed evaluation of controller in terms of its management and operation for different network services was accomplished. These network services were converted into possible use cases and implemented to test the performance of SDN controller for different network services. However, given the course of this thesis it was not feasible to evaluate all these network services for all three selected SDN controllers. The best practice of SDN network is to deploy multiple controllers in the control plane to address the challenges such as scalability, high availability, and redundancy. To avail this practice to best use, physically-centralised architected Ryu controller was not selected for implementing the use cases for this thesis. During the installation and testing the OpenDaylight controller, it was found that some major features were not supported in the newer versions, such as GUI support since the Oxygen version release and Layer 2 switch since the Fluorine version release of the OpenDaylight software. These applications were separated from OpenDaylight or there were no longer any contributors and maintainers and thus, these features were dropped from the new versions. For these reasons, even OpenDaylight controller was not selected for implementing the use cases for this thesis. All the use cases were implemented using the ONOS controller and Open vSwitches as network devices.

To evaluate the performance of ONOS controller, four different use cases were implemented in this thesis. For accurate analysation of operation of the ONOS controller, use cases were designed to test the different network services of different domains of networking. The ONOS controller was examined with the Layer 2 VPN service VPLS to isolate the layer 2 overlay networks present in the same subnet. Different configuration methods were tested and the functioning of the VPLS application was evaluated in the first use case. The ONOS controller was able to read the configuration file via VPLS application and successfully install the forwarding flow rules on the created testbed of network devices. To create the reliable SDN network, ONOS controller was analysed with the distributed implementation of the ONOS controllers. In the second use case, three ONOS controllers were deployed to form a cluster and control the underlying network. The communication within the cluster of controllers and distribution of network devices among themselves was evaluated in the second use case. Also, proof and validation of functioning failover of an instance of controller, network device and links between network devices was carried out to examine the resilience of the ONOS controller.

Further, the ONOS controller was tested with IPv6 addressing scheme. The third use case was evaluated with both IPv6 and IPv4 addressing scheme simultaneously and working of Neighbour discovery through OpenFlow protocol was studied. In addition, to answer the challenges faced by Internet Service Providers, the IPv6 enabled SDN network was deployed to demonstrate the IPv6 tunnelling over the IPv4 network which eventually help in migration of IPv4 based networks to IPv6 networks. Finally, to illustrate the integration of SDN network with the legacy networks and evaluate routing with the ONOS controller, a fourth use case was implemented. In this

use case, assessment of how routing information is exchanged between the networks via SDN-IP application was carried out. The process of migration from legacy-based networks to SDN network was examined which provides the feasible solution of migration to service providers.

The SDN promises to provide the feasible solutions for challenges faced by service providers, network managers and network designers to deploy the ideal network infrastructure. In this thesis, the performance of SDN controller and different possibilities to manage the network was carried out. However, given the new approach of the networking, SDN itself faces some difficulties such as security of the SDN network and the best placement of SDN controller in the network. Any attack on the SDN controller would disrupt the entire performance of the underlying network. Research about possible attacks is being carried out and defence systems are prototyped to protect the SDN networks from the security threats. Another problem is excessive overhead of network controlling packets. The continuous monitoring of all the underlying network devices depending on the Southbound interface protocol utilized, may generate large number of control packets which can eventually affect the overall network performance. On the other hand, insufficient monitoring of network devices may create functioning errors of the SDN applications. The future work of this thesis could be to address these challenges faced by the SDN and evaluate the performance of SDN controllers while resolving these challenges.

6 Abbreviations

A

AS	Autonomous System

B

BGP	Border Gateway Protocol
…	

D

DHCP	Dynamic Host Configuration Protocol

E

eBGP	external Border Gateway Protocol
…	

G

GNS3	Graphical Network Simulation 3

H

HA	High Availability

I

iBGP	internal Border Gateway Protocol
ICMP	Internet Control Message Protocol
ISP	Internet Service Provider
…	

L

L2VPN	Layer 2 Virtual Private Network
LLDP	Link Layer Discovery Protocol

M

MD-SAL	Model-Driven Service Abstraction Layer
MPLS	Multi-Protocol Label Switching

N

NAT	Network Address Translation
NBI	North-bound Interface
NOS	Network Operating System

O

ONF	Open Networking Foundation
ODL	OpenDaylight
ONOS	Open Network Operating System
OSGI	Open Service Gateway Initiative

P

PW	Pseudo Wire

…

R

RIP	Routing Information Protocol

S

S3P	Security, Scalability, Stability and Performance
SBI	South-bound Interface
SDN-IP	Software-defined Network - Internet Protocol
SD-VPLS	Software-defined Virtual Private LAN Service
SNMP	Simple Network Management Protocol

T

TA	Target Address
TCP	Transmission Control Protocol
TLS	Transport Layer Security

…

V

VLAN	Virtual LAN
VPLS	Virtual Private LAN Service
VPN	Virtual Private Network

7 References

[1] T. S. ,. A. S. e. a. Manar Jammal, “Software-Defined Networking: State of the Art and Research Challenges”.

[2] “Open Networking Foundation,” [Online]. Available: https://opennetworking.org/.

[3] K. S. F. L. M. M. K. Y. W. Zohaib Latif, “A Comprehensive Survey of Interface Protocols for Software Defined Networks”.

[4] G. P. e. a. Nick McKeown, “OpenFlow: Enabling Innovation in Campus Networks”.

[5] OpenNetworkingFoundation, “OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06),” 26 March 2015. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[6] N. A. Z. Z. Z. A. Mustafa Hasan Albowarab, “Software Defined Network: Architecture and Programming Language Survey”.

[7] “SDNi: A Message Exchange Protocol for Software Defined across Multiple Domains,” 2012. [Online]. Available: https://datatracker.ietf.org/doc/id/draft-yin-sdn-sdni-00.txt.

[8] “OpenDayLight Controller,” [Online]. Available: https://www.opendaylight.org/.

[9] J. O. Diego Ongaro, “In Search of an Understandable Consensus Algorithm (Extended Version)”.

[10] “Open Network Operating System,” Open Network Foundation, [Online]. Available: https://opennetworking.org/onos/.

[11] OpenNetworkFoundation, “OpenFlow Switch Specification Version 1.0.0 (Wire Protocol 0x01),” December 2009. [Online]. Available: https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf.

[12] “NOX Controller,” [Online]. Available: https://github.com/noxrepo/nox.

[13] “VMware to Acquire Nicira,” VMware, 23 July 2012. [Online]. Available: https://news.vmware.com/releases/vmw-nicira-07-23-12.

[14] “POX controller,” [Online]. Available: https://github.com/noxrepo/pox.

[15] M. C. N. G. e. a. Teemu Koponen, “Onix: A Distributed Control Platform for Large-scale Production Networks,” [Online]. Available: https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Koponen.pdf.

[16] “Ryu SDN controller,” [Online]. Available: https://ryu-sdn.org/.

[17] D. Erickson, “The Beacon OpenFlow Controller,” 2010. [Online]. Available: https://conferences.sigcomm.org/sigcomm/2013/papers/hotsdn/p13.pdf.

[18] “Floodlight Controller,” [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview.

[19] J. B. K. G. Z. C. Yonghong FU, “Orion: A Hybrid Hierarchical Control Plane of Software-Defined Networking for Large-Scale Networks”.

[20] M. B. a. J. L. Kevin Phemius, “DISCO: Distributed Multi-domain SDN Controllers”.

[21] Y. G. Soheil Hassas Yeganeh, “Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications”.

[22] M. B. M. R. B. Othmane Blial, “An Overview on SDN Architectures with Multiple Controllers,” p. 9, 2016.

[23] B. a. L. R. Team, “Distributed SDN Control: Survey, Taxonomy and Challenges,” *IEEE Communications Surveys & Tutorials,* p. 25, December 2017.

[24] E. A. M. H. R. Esmaeil Amiri, “Controller selection in software defined networks using best-worst multi-criteria decision-making,” *Bulletin of Electrical Engineering and Informatics,* 2020.

[25] “Arista Networks Announces Acquisition of Big Switch Networks,” Arista Networks, 2020. [Online]. Available: https://www.arista.com/en/company/news/press-release/9800-pr-20200214.

[26] T. F. Y. A. H. S. Rui Kubo, “Ryu SDN Framework - Open-source SDN Platform Software,” [Online]. Available: https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201408fa4.html.

[27] F. M. V. R. P. V. e. a. Diego Kreutz, “Software-Defined Networking: A Comprehensive Survey”.

[28] “AtlanticWave SDX,” [Online]. Available: https://www.atlanticwave-sdx.net/.

[29] “Linux Foundation,” [Online]. Available: https://www.linuxfoundation.org/.

[30] “ONOS : An Overview,” Onosproject, [Online]. Available: https://wiki.onosproject.org/display/ONOS/ONOS+%3A+An+Overview.

[31] “System Components,” Onosproject, [Online]. Available: https://wiki.onosproject.org/display/ONOS/System+Components.

[32] T. Vachuska, “ONOS v2.7.0 LTS Release Adds Features and Optimization Enhancements,” [Online]. Available: https://opennetworking.org/news-and-events/blog/onos-v2-7-0-lts-release-adds-features-and-optimization-enhancements/.

[33] “Guides,” Onosproject, [Online]. Available: https://wiki.onosproject.org/display/ONOS/Guides.

[34] R. 6020, “YANG - A Data Modeling Language for the Network Configuration Protocol,” [Online]. Available: https://www.rfc-editor.org/rfc/rfc6020.

[35] “OSGi Service Gateway Specification,” 2000. [Online]. Available: https://docs.osgi.org/download/r1/r1.osgi-spec.pdf.

[36] “Karaf,” [Online]. Available: https://karaf.apache.org/.

[37] “OpenDaylight Platform Overview,” OpenDaylight, [Online]. Available: https://www.opendaylight.org/about/platform-overview.

[38] “Getting Started Guide,” OpenDaylight , [Online]. Available: https://docs.opendaylight.org/en/stable-phosphorus/getting-started-guide/index.html.

[39] F. Tomonori, “Introduction to Ryu SDN framework,” [Online]. Available: https://ryu-sdn.org/slides/ONS2013-april-ryu-intro.pdf.

[40] I. Y. Kazutaka Morita, “Ryu: Network Operating System,” [Online]. Available: https://ryu-sdn.org/slides/LinuxConJapan2012.pdf.

[41] “Ryu SDN Framework,” [Online]. Available: https://book.ryu-sdn.org/en/html/index.html.

[42] “FaucetSDN Ryu Source code,” [Online]. Available: https://github.com/faucetsdn/ryu.

[43] “Welcome to RYU the Network Operating System(NOS),” [Online]. Available: https://ryu.readthedocs.io/en/latest/getting_started.html.

[44] R. M. R. B. Arpit Gupta, “An Industrial-Scale Software Defined Internet Exchange Point”.

[45] H. Z. M. F. Naga Katta, “Ravana: Controller Fault-Tolerance in Software-Defined Networking”.

[46] “Open vSwitch,” [Online]. Available: https://www.openvswitch.org/.

[47] “Pica8,” [Online]. Available: https://www.pica8.com/.

[48] “OpenFlow software switch 1.3,” CPqD , [Online]. Available: https://cpqd.github.io/ofsoftswitch13/.

[49] “LINC - OpenFlow software switch,” [Online]. Available: https://github.com/FlowForwarding/LINC-Switch.

[50] “Indigo Virtual Switch,” [Online]. Available: https://github.com/floodlight/ivs.

[51] “What is Erlang?,” [Online]. Available: https://www.erlang.org/faq/introduction.html.

[52] “Why Open vSwitch?,” [Online]. Available: https://github.com/openvswitch/ovs/blob/master/Documentation/intro/why-ovs.rst.

[53] “Open vSwitch using OpenFlow,” [Online]. Available: https://docs.openvswitch.org/en/latest/faq/openflow/.

[54] “Mininet,” [Online]. Available: http://mininet.org/.

[55] “Download/Get Started With Mininet,” [Online]. Available: http://mininet.org/download/.

[56] D. M. S. A. Wette P, “ Maxinet: Distributed emulation of software-defined networks,” *Networking Conference.*

[57] M. F. B. M. F. Z. R. A. a. R. B. Arup Raton Roy, “Design and Management of DOT: A Distributed OpenFlow Testbed”.

[58] “Graphical Network Simulation 3,” [Online]. Available: https://www.gns3.com/.

[59] “GNS3 Marketplace,” [Online]. Available: https://www.gns3.com/marketplace/featured.

[60] “GNS3 Software,” [Online]. Available: https://www.gns3.com/software.

[61] OpenNetworkFoundation, “OpenFlow Switch Specification Version 1.1.0 Implemented (Wire Protocol 0x02),” February 2011. [Online]. Available: https://paperzz.com/doc/7086757/openflow-switch-specification.-version-1.1.0.

[62] OpenNetworkFoundation, “OpenFlow Switch Specification Version 1.2 (Wire Protocol 0x03),” December 2011. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.2.pdf.

[63] OpenNetworkFoundation, “OpenFlow Switch Specification Version 1.3.0 (Wire Protocol 0x04),” June 2012. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf.

[64] OpenNetworkFoundation, “OpenFlow Switch Specification Version 1.4.0 (Wire Protocol 0x05),” October 2013. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf.

[65] OpenNetworkFoundation, “OpenFlow Switch Specification Version 1.5.0 (Protocol version 0x06),” Decemeber 2014. [Online].

[66] “OpFlex Control Protocol,” 2014. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-smith-opflex-00.

[67] “Forwarding and Control Element Separation (ForCES) Framework,” [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3746.

[68] “Open Networking Foundation,” [Online]. Available: https://opennetworking.org/p4/.

[69] “Clarifying the differences between P4 and OpenFlow,” Open Networking Foundation, [Online]. Available: https://opennetworking.org/news-and-events/blog/clarifying-the-differences-between-p4-and-openflow/.

[70] “The ONOS Web GUI,” Onosproject.org, [Online]. Available: https://wiki.onosproject.org/display/ONOS/The+ONOS+Web+GUI.

[71] “Introducing JSON,” [Online]. Available: https://www.json.org/json-en.html.

[72] “Intent Framework,” Onosproject, [Online]. Available: https://wiki.onosproject.org/display/ONOS/Intent+Framework.

[73] “Announcing Mininet 2.3.0,” Mininet.org, [Online]. Available: http://mininet.org/blog/2021/02/28/announcing-mininet-2-3-0/.

[74] B. H. N. M. Bob Lantz, “A Network in a Laptop: Rapid Prototyping for Software-Defined Networks,” *9th ACM Workshop on Hot Topics in Networks,* October 2010.

[75] B. Lantz, “Onosproject.org,” 27 January 2017. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Mininet+and+onos.py+workflow.

[76] D. P. K. I. P. R. Charles Chan, “How to Enable IPv6,” Onosproject.org, January 2017. [Online]. Available: https://wiki.onosproject.org/display/ONOS/IPv6#IPv6-HowtoEnableIPv6.

[77] “Virtual Private LAN Service (VPLS) RFC 4761,” [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4761.

[78] “ Pseudowire Setup and Maintenance RFC 4447,” no. https://datatracker.ietf.org/doc/html/rfc4447.

[79] M. Y. A. G. e. a. Madhusanka Liyanage, “Enhancing Security, Scalability and Flexibility of Virtual Private LAN Services,” 2017.

[80] K. G. A. K. J. G. e. a. K Gaur, “A survey of Virtual Private LAN Services (VPLS): Past, present and future,” *Computer Networks,* June 2021.

[81] “Virtual Private LAN Service (VPLS) RFC 4762,” [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4762.

[82] M. Liyanage, “Enhancing security and scalability of Virtual Private LAN Services,” University of Oulu, Finland, 2016.

[83] C.-M. O. e. a. Carolina Fernández, “Virtual Private LAN Service - VPLS,” Onosproject.org, 30 May 2017. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Virtual+Private+LAN+Service+-+VPLS.

[84] W. M. L. F. K. Wenjuan Li, “A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures,” *Journal of Network and Computer Applications,* p. 14, 2016.

[85] J. Halterman, “Atomix,” 2013. [Online]. Available: https://github.com/atomix/atomix.

[86] J. H. e. a. Luca Prete, “Forming a cluster,” Onosproject.org, [Online]. Available: https://wiki.onosproject.org/display/ONOS/Forming+a+cluster.

[87] Y.-F. L. Eric Tang, “Notes on cluster formation for Docker instances,” Onosproject.org, 2022. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Notes+on+cluster+formation+for+Docker+instances.

[88] J. Halterman, “ONOS Cluster Configuration,” Onosproject.org, 2018. [Online]. Available: https://wiki.onosproject.org/pages/viewpage.action?pageId=28836788.

[89] S. Wu, “Network Discovery,” Onosproject.org, 2016. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Network+Discovery.

[90] R. Eddy, “Multicast Use Case,” Onosproject.org, 2015. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Multicast+Use+Case.

[91] G. M. S. A. C. R. R. H. E. M. Dimas A. Marenda, “Intent-Based Path Selection for VM Migration Application with Open Network Operating System,” Institut Teknologi Bandung.

[92] T. T. D. L. e. a. Wenfeng Xia, “A Software Defined Approach to Unified IPv6 Transition”.

[93] D. B. R. P. M. Babu R. Dawadi. Shashidhar R. Joshi, “Towards Smart Networking with SDN Enabled IPv6 Network,” 2022.

[94] “Guidelines for Using IPv6 Transition Mechanisms during IPv6 Deployment,” [Online]. Available: https://datatracker.ietf.org/doc/html/draft-arkko-ipv6-transition-guidelines-14#section-4.

[95] Y. F. H. H. e. a. Mahmoud Mazhar Samad, “Deploying Internet Protocol version 6 (IPv6) over Internet Protocol version 4 (IPv4) tunnel”.

[96] “Advisory Guidelines for 6to4 Deployment RFC 6343,” [Online]. Available: https://www.rfc-editor.org/rfc/rfc6343.html.

[97] “Connection of IPv6 Domains via IPv4 Clouds RFC 3056,” [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3056.

[98] “RIPng for IPv6 RFC 2080,” [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2080.

[99] D. B. R. S. R. J. a. P. M. Babu R. Dawadi, “Legacy Network Integration with SDN-IP Implementation towards a Multi-Domain SoDIP6 Network Environment,” *electronics,* 2020.

[100] “A Border Gateway Protocol 4 (BGP-4) RFC 4271,” [Online]. Available: https://www.rfc-editor.org/rfc/rfc4271 .

[101] “Quagga Routing Software Suite,” [Online]. Available: https://www.nongnu.org/quagga/index.html.

[102] “BSD Router Project: Open Source Router Distribution,” [Online]. Available: https://bsdrp.net/BSDRP.

[103] “FRRouting Project,” [Online]. Available: https://frrouting.org/.

8 Appendix

Configuration file flows.json for flow rule creation as discussed in chapter 4.2.2.

```
  GNU nano 4.8                  flows.json
{
  "priority": 100,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:00004e6d3cf34349",
  "treatment": {
    "instructions": [
      {
        "type": "OUTPUT",
        "port": "3"
      }
    ]
  },
  "selector": {
    "criteria": [
      {
        "type": "ETH_DST",
        "mac": "CA:03:17:6D:00:08"
      },
      {
        "type": "ETH_SRC",
        "mac": "CA:01:0F:AE:00:08"
      },
      {
        "type": "IN_PORT",
        "port": "1"
      },
      {
        "type": "IPV4_SRC",
        "ip": "10.0.0.1/24"
      },
      {
        "type": "IPV4_DST",
        "ip": "10.0.0.3/24"
      }
    ]
  }
}
```

Configuration file intents.json for intent creation as discussed in chapter 4.2.3.

```
  GNU nano 4.8                  intents.json
{
 "type": "HostToHostIntent",
 "appID": "org.onosproject.ovsdb",
 "priority": 1000,
 "one": "CA:02:0F:BE:00:08/None",
 "two": "CA:03:17:6D:00:08/None",
}
```

Configuration file for VPLS configuration through REST API as discussed in chapter 4.5.2.

GNU nano 4.8 network-cfg-vpls.json

```
{
  "ports": {
    "of:00008a507a0c8f4a/14": {
      "interfaces": [ {
          "name": "h1",
          "mac" : "CA:01:11:1D:00:08"
        } ]
    },
    "of:0000a2f8233e3643/14": {
      "interfaces": [ {
          "name": "h2",
          "mac" : "CA:02:11:2D:00:08"
        } ]
    },
    "of:000022494702a540/14": {
      "interfaces": [ {
          "name": "h3",
          "mac" : "CA:05:11:5D:00:08"
        } ]
    },
    "of:0000b210c06d6046/14": {
      "interfaces": [ {
          "name": "s1",
          "mac" : "CA:03:11:3D:00:08"
        } ]
    },
    "of:0000fa9767881347/14": {
      "interfaces": [ {
          "name": "s2",
          "mac" : "CA:04:11:4D:00:08"
        } ]
    },
    "of:00005eba6686e24f/14": {
      "interfaces": [ {
          "name": "s3",
          "mac" : "CA:06:11:6D:00:08"
        } ]
    }
  },

  "apps" : {
    "org.onosproject.vpls" : {
      "vpls" : {
        "vplsList" : [
          {
            "name" : "VPLS1",
            "interfaces" : ["h1", "s2", "s3"],
          },
          {
            "name" : "VPLS2",
            "interfaces" : ["h2", "h3", "s1"],
          }
        ]
      }
    }
  }
}
```

Configuration file for Atomix cluster formation as discussed in chapter 4.6.1.

Atomix-1 configuration file as seen below, similarly configuration file for Atomix-2, Atomix-3 were created.

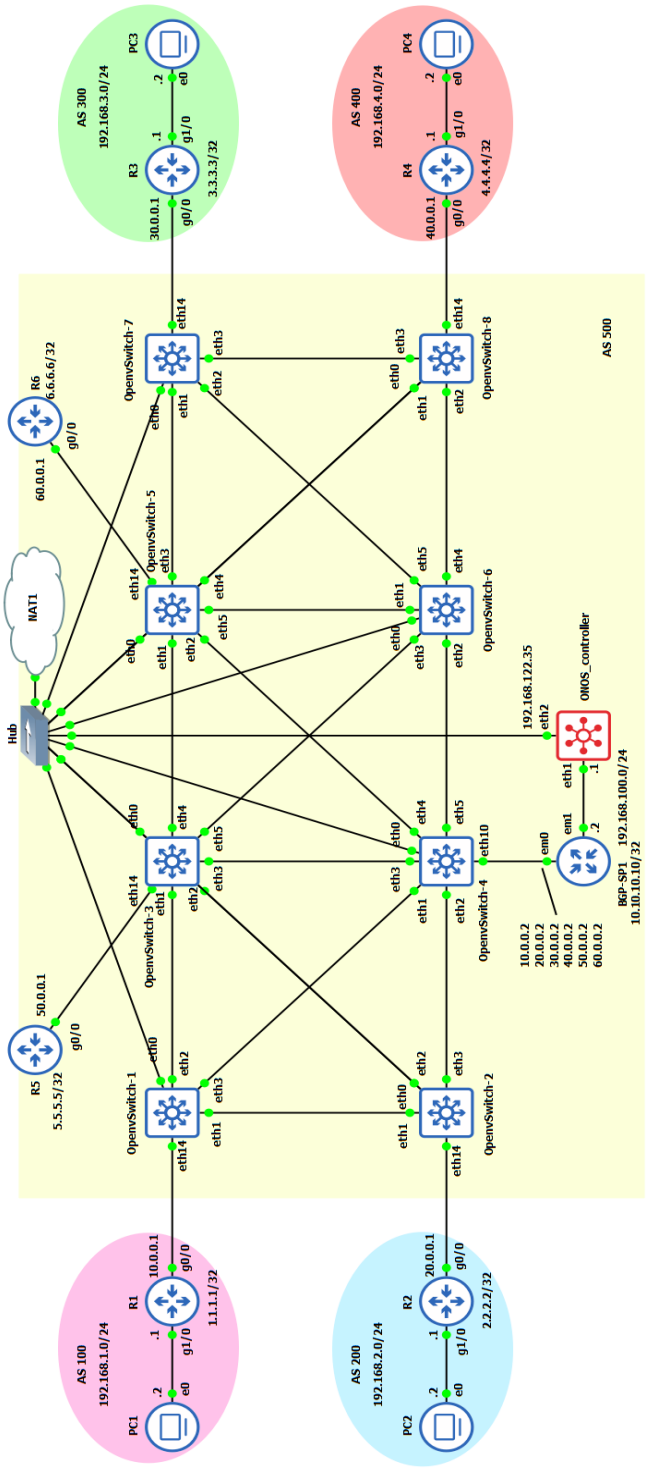
```
 GNU nano 4.8            atomix-1.conf
cluster {
  cluster-id: onos
  node {
    id: atomix-1
    address: "172.17.0.2:5679"
  }
  discovery {
    type: bootstrap
    nodes.1 {
      id: atomix-1
      address: "172.17.0.2:5679"
    }
    nodes.2 {
      id: atomix-2
      address: "172.17.0.3:5679"
    }
    nodes.3 {
      id: atomix-3
      address: "172.17.0.4:5679"
    }
  }
}

management-group {
  type: raft
  partitions: 1
  storage.level: disk
  members: [atomix-1, atomix-2, atomix-3]
}

partition-groups.raft {
  type: raft
  partitions: 7
  storage.level: disk
  members: [atomix-1, atomix-2, atomix-3]
}
```

Configuration file for ONOS cluster formation as discussed in chapter 4.6.1.

ONOS-1 configuration file as seen below, similarly configuration file for ONOS -2, ONOS -3 were created.

```
 GNU nano 4.8            cluster-1.json
{
  "name": "onos",
  "node": {
    "id": "172.17.0.5",
    "ip": "172.17.0.5",
    "port": 9876
  },
  "storage": [
    {
      "id": "atomix-1",
      "ip": "172.17.0.2",
      "port": 5679
    },
    {
      "id": "atomix-2",
      "ip": "172.17.0.3",
      "port": 5679
    },
    {
      "id": "atomix-3",
      "ip": "172.17.0.4",
      "port": 5679
    }
  ]
}
```

Topology created in the GNS3 with different legacy networks and SDN network as seen in Figure 4. 78.



Configuration file for SDN-IP configuration as discussed in chapter 4.8.2.

The interfaces configuration. (1/2)

```
 GNU nano 4.8                network-cfg-bgp.json
{
  "ports": {
      "of:00008a507a0c8f4a/14": {
      "interfaces": [
        {
          "name": "r1",
          "ips" : [ "10.0.0.2/24" ],
          "mac" : "0c:8a:49:34:00:00"
        }
      ]
    },
    "of:0000a2f8233e3643/14": {
      "interfaces": [
        {
          "name": "r2",
          "ips" : [ "20.0.0.2/24" ],
          "mac" : "0c:8a:49:34:00:00"
        }
      ]
    },
    "of:0000aaddcd04fc4d/14": {
      "interfaces": [
        {
          "name": "r3",
          "ips" : [ "30.0.0.2/24" ],
          "mac" : "0c:8a:49:34:00:00"
        }
      ]
    },
    "of:00005eba6686e24f/14": {
      "interfaces": [
        {
          "name": "r4",
          "ips" : [ "40.0.0.2/24" ],
          "mac" : "0c:8a:49:34:00:00"
        }
      ]
    },
    "of:0000421acd21094c/14": {
      "interfaces": [
        {
          "name": "r5",
          "ips" : [ "50.0.0.2/24" ],
          "mac" : "0c:8a:49:34:00:00"
        }
      ]
    },

    "of:0000d69c0157814f/14": {
      "interfaces": [
        {
          "name": "r6",
          "ips" : [ "60.0.0.2/24" ],
          "mac" : "0c:8a:49:34:00:00"
        }
      ]
    }
  },
```

The BGP speaker and Reactive Routing application configuration. (2/2)

```
"apps" : {
   "org.onosproject.router" : {
      "bgp" : {
          "bgpSpeakers" : [
              {
                  "name" : "speaker1",
                  "connectPoint" : "of:000022494702a540/10",
                  "peers" : [
                      "10.0.0.1",
                      "20.0.0.1",
                      "30.0.0.1",
                      "40.0.0.1"
                   ]
               }
           ]
       }
    },
     "org.onosproject.reactive.routing" : {
          "reactiveRouting" : {
              "ip4LocalPrefixes" : [
                  {
                      "ipPrefix" : "50.0.0.0/24",
                      "type" : "PUBLIC",
                      "gatewayIp" : "50.0.0.254"
                  },
                  {
                      "ipPrefix" : "60.0.0.0/24",
                      "type" : "PRIVATE",
                      "gatewayIp" : "60.0.0.254"
                  }
               ],
              "ip6LocalPrefixes" : [
               ],
               "virtualGatewayMacAddress" : "0c:8a:49:34:00:00"
          }
      }
   }
}
```

Configuration of BGP-Speaker as discussed in chapter 4.8.2.

```
Current configuration:
!
frr version 7.2
frr defaults traditional
hostname bgpsp1
!
interface em0
 ip address 10.0.0.2/24
 ip address 20.0.0.2/24
 ip address 30.0.0.2/24
 ip address 40.0.0.2/24
 ip address 50.0.0.2/24
 ip address 60.0.0.2/24
!
interface em1
 ip address 192.168.100.2/24
!
interface em3
 shutdown
!
interface lo0
 ip address 10.10.10.10/32
!
router bgp 500
 bgp router-id 10.10.10.10
 neighbor 10.0.0.1 remote-as 100
 neighbor 10.0.0.1 ebgp-multihop 255
 neighbor 20.0.0.1 remote-as 200
 neighbor 20.0.0.1 ebgp-multihop 255
 neighbor 30.0.0.1 remote-as 300
 neighbor 30.0.0.1 ebgp-multihop 255
 neighbor 40.0.0.1 remote-as 400
 neighbor 40.0.0.1 ebgp-multihop 255
 neighbor 192.168.100.1 remote-as 500
 neighbor 192.168.100.1 port 2000
 neighbor 192.168.100.1 timers connect 5
 !
 address-family ipv4 unicast
  network 50.0.0.0/24
 exit-address-family
!
line vty
!
end
```