# EXPLORING SDN AND ITS USE CASE IN OPENSTACK

## Major Project Report

*Submitted in partial fulfilment of the requirements*

*for the degree of*

**Master of Technology**

**in**

**Electronics & Communication Engineering**

**(Embedded Systems)**

By


## Janki H. Chhatbar

**(13MECE06)**




**Electronics & Communication Engineering Branch**

**Electrical Engineering Department**

**Institute of Technology**

**Nirma University**

**Ahmedabad-382 481**


**May 2015**

# EXPLORING SDN AND ITS USE CASE IN OPENSTACK

**Major Project Report**

*Submitted in partial fulfilment of the requirements*
*for the degree of*

**Master of Technology**
**in**
**Electronics & Communication Engineering**
**(Embedded Systems)**

By

**Janki H. Chhatbar**
**(13MECE06)**

Under the guidance of

**External Project Guide:**
**Mr. Sunil Singh**
Senior Engineer (Level 1),
e-Infochips Pvt. Ltd.,
Ahmedabad.

**Internal Project Guide:**
**Prof. Sachin H. Gajjar**
Assistant Professor (EC Dept.),
Institute of Technology,
Nirma University, Ahmedabad.


**Electronics & Communication Engineering Branch**
**Electrical Engineering Department**
**Institute of Technology**
**Nirma University**
**Ahmedabad-382 481**

**May 2015**

# Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

2. Due acknowledgement has been made in the text to all other materials used.

**- Janki H. Chhatbar**

# Disclaimer

"The content of this thesis does not represent the technology, opinions, beliefs, or positions of e-Infochips Pvt. Ltd., its employees, vendors, customers, or associates."


# Certificate

This is to certify that the Major Project entitled **"Exploring SDN and its use case in OpenStack"** submitted by **Janki H. Chhatbar (13MECE06)**, towards the partial fulfilment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad, is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

**Prof. S. H. Gajjar**
Guide

**Dr. N. P. Gajjar**
Program Coordinator

**Dr. D. K. Kothari**
Section Head, EC

**Dr. P. N. Tekwani**
Head of EE Dept.

**Dr. Ketan Kotecha**
Director, IT

# Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr. P. N. Tekwani**, Head of Electrical Engineering Department, and **Dr. N. P. Gajjar**, PG Coordinator of M. Tech. Embedded Systems program for allowing me to undertake this thesis work and for his guidelines during the review process.

I take this opportunity to express my profound gratitude and deep regards to **Prof. Sachin H. Gajjar**, guide of my major project for his exemplary guidance, monitoring and constant encouragement throughout the course of this thesis. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I would take this opportunity to express a deep sense of gratitude to **Mr. Bhaskar Trivedi**, Delivery Manager (Level 2), for giving me an opportunity to work on this project. I thank my Project Mentor **Mr. Rajeshkumar Vivekanandan**, Delivery Manager (Level 1), for his cordial support, constant supervision as well as for providing valuable information regarding the project and guidance, which helped me in completing this task through various stages. I would also thank **Mr. Prajose John**, Technical Lead (Level 2), for always helping me, giving good suggestions and solving my doubts. I thank **Mr. Sunilkumar Singh**, Senior Engineer (Level 1), for his support and assistance in completing my project.

I am obliged to all the members of OpenStack team, e-Infochips Pvt. Ltd. for the valuable information provided by them in their respective fields.

Lastly, I thank almighty, my parents, and friends for their constant encouragement without which this assignment would not be possible.

**- Janki H. Chhatbar**

**13MECE06**

# Abstract

Software Defined Networking (SDN) is a new emerging approach towards networking which makes managing a network easy and efficient. It has advantages like rapid innovation, network-wide view, more flexibility and easy debug. SDN is an emerging networking concept which decouples control plane and data plane. The control plane is called a controller. One such controller is OpenDayLight (ODL).

OpenDayLight is a project aiming to make SDN a success. There are various projects under ODL like ODL controller, Dlux, SNMP4SDN, ovsdb-northbound, ovs-openstack. ODL controller acts as a control plane in the SDN architecture and controls the switches lying underneath. OpenDayLight controller can be used to control an emulated network as well as a real ethernet switches. ODL controller described is Helium SR1.1 version. It is benchmarked with a tool called WCBench.

To get acquainted with the controller, a network emulator called Mininet is used. Various ways to create custom network in Mininet are also described. Virtual switches like OVS and CPqD are managed using ODL. The procedure to be followed to make the controller interact with real Ethernet switches is discussed.

SDN and traditional networks are compared for performance in NS3 simulator. The result could not be validated. The reasons are discussed as well.

Cloud computing is gaining momentum. Recent buzz word, OpenStack is in focus. A good understanding on cloud computing and its advantages, particularly to software developers are obtained. Three node architecture Icehouse installation on Ubuntu 14.04 Desktop Operating System is set up. OpenStack uses Open vSwitch switches in its connection. These switches could be managed by ODL too. This brings out the possibility of integrating ODL with OpenStack. The step by step procedure and advantages of integrating them both are highlighted.

# Contents

# List of Tables

# List of Figures

# Abbreviation Notation and Nomenclature

API ..... Application Programming Interface
CLI ..... Command Line Interface
Dlux ..... openDayLight User Interface
GPS ..... Global Positining System
IP ..... Internet Protocol
LLDP ..... Link Layer Discovery Protocol
NETCONF ..... NETwork CONFiguration
ODL ..... OpenDayLight
OF ..... OpenFlow
OFP ..... OpenFlow Protocol
OID ..... Object Identifier
OSGi ..... Open Service Gateway initiative
OVS ..... Open vSwitch
OVSDB ..... Open vSwitch DataBase
PDU ..... Protocol Data Unit
REST ..... Representational State Transfer
SDN ..... Software Defined Network
SNMP ..... Simple Network Management Protocol
TCP ..... Transport Control Protocol
TLV ..... Type-Length-value
UDP ..... User Datagram Protocol
UI ..... User Interface
VLAN ..... Very Large Area Network
VM ..... Virtual Machine
VNF ..... Virtual Network Function

# Chapter 1

# Introduction

## 1.1 Background

Today everything is in a network. Almost all devices have an Internet Protocol (IP) address and can communicate with each other. For proper communication, ensuring that the routes are installed correctly in a router is a must. But routers come with limitations. Their memory is limited. Vendor-specific operating systems (OS) also at times restrict a network administrator from implementing many things. A change in network, like adding or deleting a router, would mean changing the routes in all concerned surrounding routers. This becomes cumbersome when there are hundreds of routers.

A break-through is an introduction of concept named Software Defined Networking (SDN). An era of programmable computer networks is on its way. SDN architecture makes networking devices OS independent.

This new concept is to be explored to find ways to incorporate it into the existing network.

## 1.2 Motivation

Social media, mobile devices and cloud computing are pushing traditional networks to their limits. Compute and storage have benefited from incredible innovations in virtualization and automation, but those benefits are constrained by limitations in the network. Administrators may spin up new compute and storage instances in minutes, only to be held up for weeks by rigid and often-times manual network operations.

SDN has the potential to revolutionize legacy data centres by providing a flexible way to control the network so it can function more like the virtualized versions of compute and storage today. Its advantages to cloud computing platform are the ones not to be overlooked. OpenStack is cloud computing platform providing Infrastructure-as-a-Service (IaaS). There is a huge chance that SDN can be incorporated in the cloud network.

SDN will drive significant changes in how networks are built and operated. SDN places more control of network configuration and state in the hands of logically centralized software - putting greater control of network-based innovation and differentiation in the hands of network operators. It leverages global views of network resource and service demand information, promoting operations, automation, and resource optimization, while liberating and accelerating new service creation.

## 1.3 Problem Statement

The task is to get acquainted with OpenDayLight (ODL) controller. The approach taken is to first start with Mininet emulator and then move to real ethernet switches. The controller is then used to control OpenStack overlay network.

1. To create a network in Mininet and make a switch act as a firewall using ODL controller.

2. To create a network of two hosts connected to a real Ethernet switch and use

SNMP4SDN plugin of OpenDayLight Helium controller to dump the flow into the switch such that the hosts are not able to ping each other.

3. To implement bandwidth throttling using ODL.

4. To integrate ODL with OpenStack network.

## 1.4 Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2 and 3 covers the literature about SDN, ODL and OpenStack. Chapter 2 gives overview of SDN, Mininet and ODL. It also describes ways in which a Mininet could be installed. It mentions working of OpenFlow (OF) protocol. It explains Open vSwitch (OVS) and CPqD switches. Chapter 3 explains cloud computing concept, its advantages, particularly to software developers, and working of OpenStack.

Chapter 4 and 5 implements the learnt concept. Chapter 4 guides through installing ODL controller, openDayLight User eXperience (Dlux) and SNMP4SDN features of the controller. It describes how to inter-operate Mininet with ODL controller. Two ways to create a custom topology in Mininet are also included. It discusses ways of programming the controller to achieve desired network behavior. Bandwidth throttling achieved using ODL is also a part of this chapter. Chapter 5 explains setting up and OpenStack Icehouse three node architecture installation. These chapters also highlight the issues faced in achieving the problem statement and its solution adopted.

Chapter 6 describes the procedure of using ODL with OpenStack.

Chapter 7 states conclusion and lists out future scope of the project.

**Note:** Through out the thesis, issue and its solution would be shown in a grey box as below:

Issue: This is the issued faced.

Solution: This is the solution adopted.

The code snippet is included in a box as shown below:

```
Code goes here.
```

## 1.5 Supporting Technologies

Many available technologies were used in achieving the problem statement. The thesis also mentions use of protocols used. There is a minimum system requirements for running the ODL controller. These along with technologies used are summarised as below.

| | |
|---|---|
| System Requirements: | Java 7, 64-bit OS, 4 GB RAM |
| Operating system: | Linux, Windows |
| Emulator: | Mininet, NS3 |
| Networking Protocols: | SNMPv2, LLDP, OpenFlow, OVSDB |
| Languages: | C, Java, Python |
| Software: | Net-SNMP agent |
| Tools: | Wireshark, mib2c |
| Virtualiser: | VirtualBox |
| API: | REST |
| Framework: | Django, Karaf |

# Chapter 2

# Software Defined Networking

## 2.1 Introduction to Software Defined Networking

Software Defined Networking is an emerging concept which aims to centralise the network routing mechanism. To understand it better, lets start with a simple analogy.

Suppose a package of data is to be sent across town. It is given to a courier service. The courier service person starts off on the road to deliver it. He now is in the network.

He has an idea of the route which stops at places to ask for directions and find fastest route. At times he encounters a end road and has to return searching for another route. Traffic might be high on certain roads and he has to take a road with lighter traffic. The packet might be too large to be taken through narrow raods. Those people he is asking are analogous to routers. They have the knowledge and he needs to consult them at each point in the journey.

The same analogy applies over SDN. The courier guy now has a smartphone with Global Positioning System (GPS). He just starts off with the packet and route gets updated in the GPS. GPS is the central control here. The central control is aware of the shape of the package and can route him accordingly. He can also see

traffic conditions on GPS. GPS dynamically updates the route. If his phone dies, he can ask the way.

The central point, or control plane is software as opposed to the hardware routers in the first scenario. In case of unavailability of the software, the hardware provides a route.

So, analysing technically, traditional network hardware has the rules and logic for controlling the flow and modification of data in proprietary firmware, which is partitioned into data planes and control planes[1].

To understand control and data planes better, lets have another analogy. Consider public transportation of a city. Before sending bus drivers out, a plan is needed.

**Control Plane = Learning what is to be done.**

The planning stage includes deciding which paths will be taken by bus. This is similar to the control plane in the network. People haven't been picked up yet, nor have been dropped off, but the paths and stops are known due to the plan. The control plane is primarily about the learning of routes.

In a routed network, this planning and learning can be done through static routes, where the router are trained about remote networks, and how to get there. Dynamic routing protocols allow the routers to train each other regarding how to reach remote networks, can also be used. This is all the control plane.

**Data Plane = Actually moving the packets based on what is learned.**

Now, after the routers know how to route for remote networks, along comes a customers packet. This is were the data plane begins. The data plane symbolises the movement of the customer's data packets over the transit path. (The path to be used was learned in the control plane stage earlier)[2].

The tasks of control and data planes are summarised in table 2.1 on page 7.

### 2.1.1 Need of separating these planes

Separating data and control planes have various benefits as listed below.

Table 2.1: Control and Data plane summary

| Control-Plane tasks | Data-Plane tasks |
|---|---|
| Less time-critical | Operations occurring real-time on packet path |
| Control and management of device operation | Core device operations |
| Table maintenance, port states, etc | Receive, process and transmit packets |

1. **Rapid innovation:** Control logic is not tied to the hardware. This leads to more rapid innovation.

2. **Network-wide view:** Centralised logic makes it easier to infer and reason about network behaviour.

3. **More flexibility:** Flexibility can introduce more services easily.

4. **Evolution and development:** It is not limited to the software that comes with the hardware.

5. **Easy debug:** Controlling the network from a high-level software program makes debugging or checking of network behaviour easy.

### 2.1.2 SDN Architecture

From a high level view, SDN is commonly described in layers as explained below.

1. **Network Apps & Orchestration:** SDN applications run on top of SDN controller and communicate to it through NorthBound APIs (NBI).

2. **Controller Platform:** SDN controller is he middle layer in SDN architecture. It is the intelligent part which takes routing decisions. It manages the underlying data plane through SouthBound APIs (SBI). The ODL controller that is discussed in this thesis falls into this second layer.

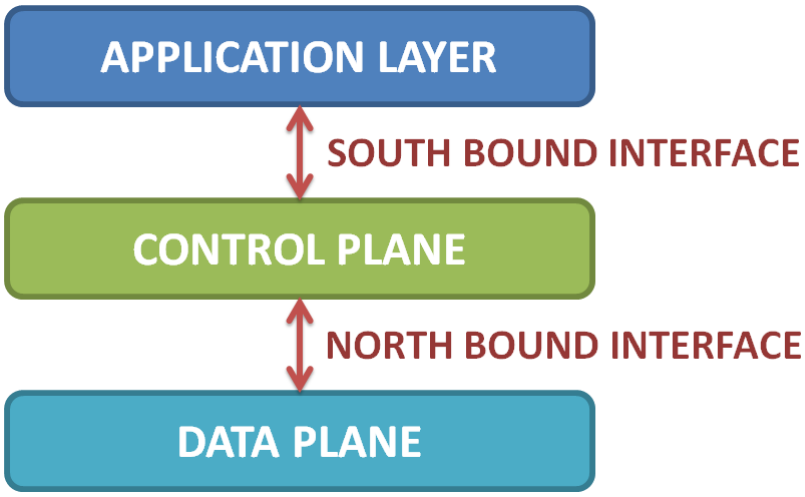


Figure 2.1: SDN Architecture

3. **Physical & Virtual Network Devices:** The lowest layer in SDN architecture, it contains physical and virtual networking devices like routers and switches.

The SDN architecture is pictorially depicted in figure 2.1 on page 8.

### 2.1.3 Difference between traditional and SDN based network

Traditional network has routers to connect to different subnets. Routers contain intelligence (Central Processing Unit (CPU) and OS), memory (RAM, ROM) which stores flows and data plane which forwards the packets. Each router will have its own OS and Command Line Interface (CLI). Routers from different vendors generally do not inter-operate and if they do, its not without a cumbersome process. In SDN, the intelligence is taken from routers and shifted to a central place. In SDN terms, this central place is called a controller. Controller is a software intelligent enough to take routing decisions. Basically router is split into 2 parts: Intelligence is shifted to a common place and forwarding hardware at its place. This difference is shown in figure 2.2 on page 9.

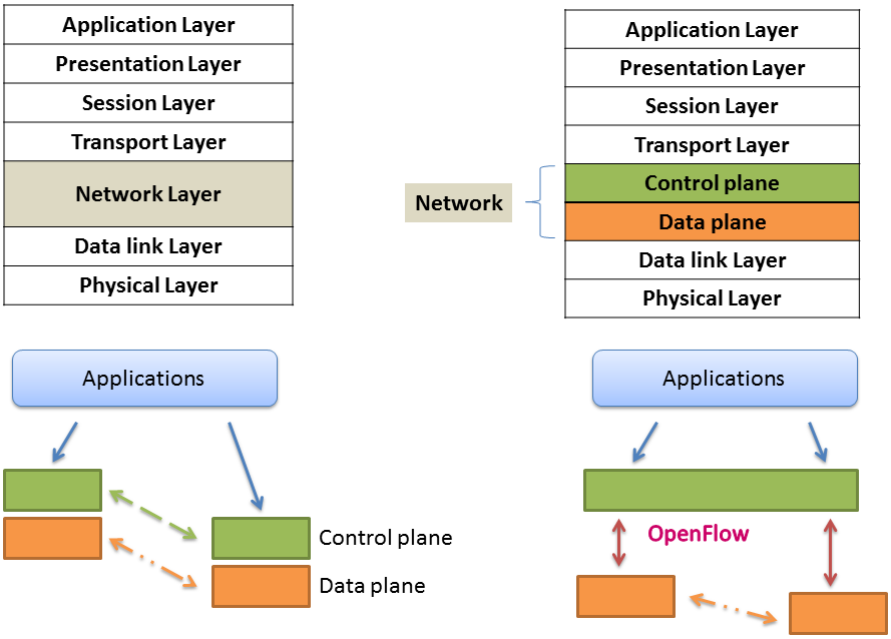


Figure 2.2: Difference between traditional and SDN network

## 2.2 OpenDayLight Controller

OpenDayLight (ODL) Controller is one among many SDN controllers. It is an open source project under Linux Foundation. Many corporations like Cisco, Red Hat, Brocade, Juniper are involved in developing it. It celebrated its second birthday in April 2015 [3]. Its first release was called Hydrogen, later Helium and next Formal Lithium Release is expected during mid-2015 [4]. Lithium release is expected to have support for Internet of Things (IoT) as well. The controller is more than 2,230,390 lines of code [5], by March 2015, majorly written in Java, followed by C++ and relatively less in C. It can be installed on a hardware with 64 bit architecture, 4 GB RAM and supporting Java 7. The code is built using maven and has Open Services Gateway Initiative (OSGi) framework. OSGi allows dynamic loading of bundles, thus various features of ODL can work independent of each other.

Three aspects of the project are listed below.

1. **Code:** To create a robust, extensible, open source code. The code should cover

major common components required to build a SDN solution.

2. **Acceptance:** To get broad industrial acceptance amongst vendors and users.

3. **Community:** To have a thriving and growing technical community contributing to the code base, using the code in commercial products.

It is a collection of various projects working in collaboration of each other. The whole list of projects is available at [6]. ODL release used in this thesis is Helium SR1.1. Pictorial representation of Helium is shown in figure 2.3 on page 10. ODL has many

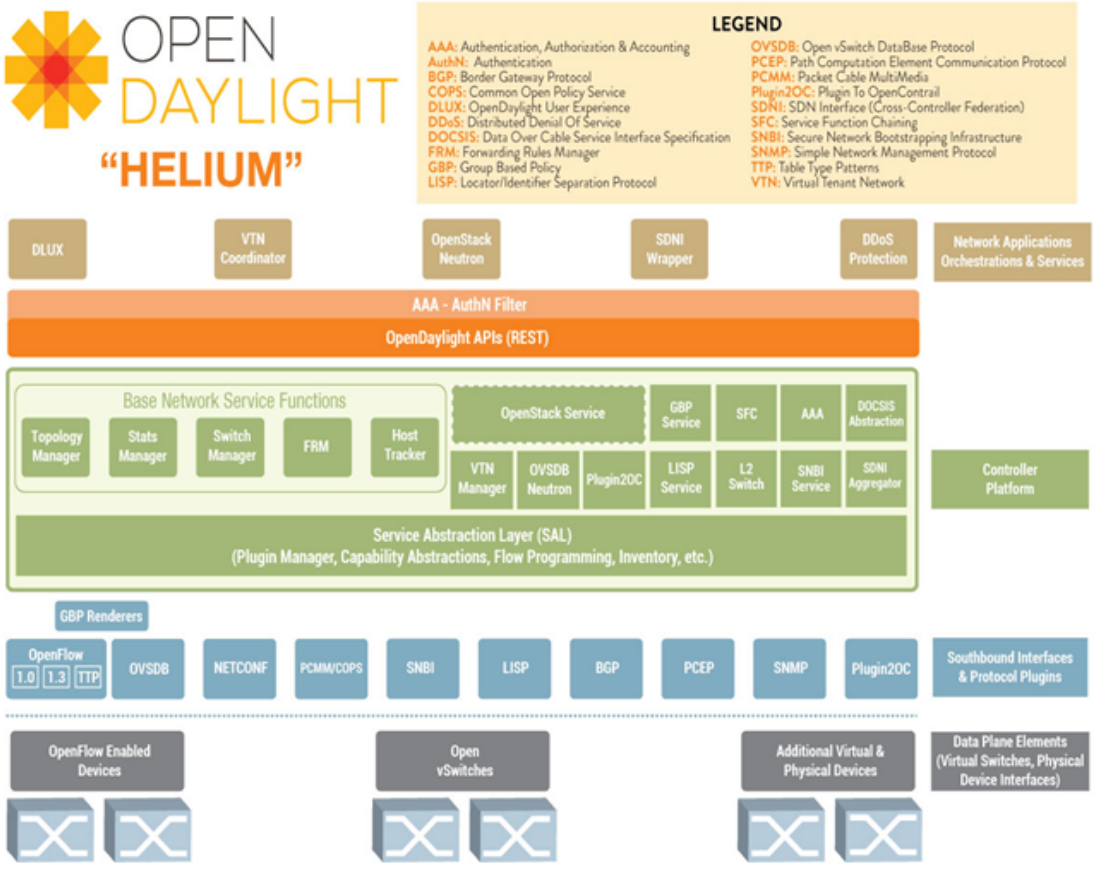


Figure 2.3: ODL Helium architecture [7]

SouthBound Interfaces (SBI). These let ODL communicate to variety of underlying devices like ethernet switches, Network Configuration (NETCONF) protocol

enabled devices, OpenFlow (OF) enabled devices, Open vswitch (OVS). To communicate with ethernet switches, Simple Network Management Protocol (SNMP) is used. ODL communicates with OVS through OF and Open vSwitch DataBase management (OVSDB) protocol.

### 2.2.1 OpenDayLight User Interface

OpenDayLight User Interface called Dlux is a pure JavaScript based ODL user interface that uses REST/RESTConf APIs of controller. Helium is the first release of Dlux. Dlux can be deployed as karaf feature along with controller or it can work as a standalone application. It is shown in snapshot 2.4 on page 11.

The following are the main features of Dlux.

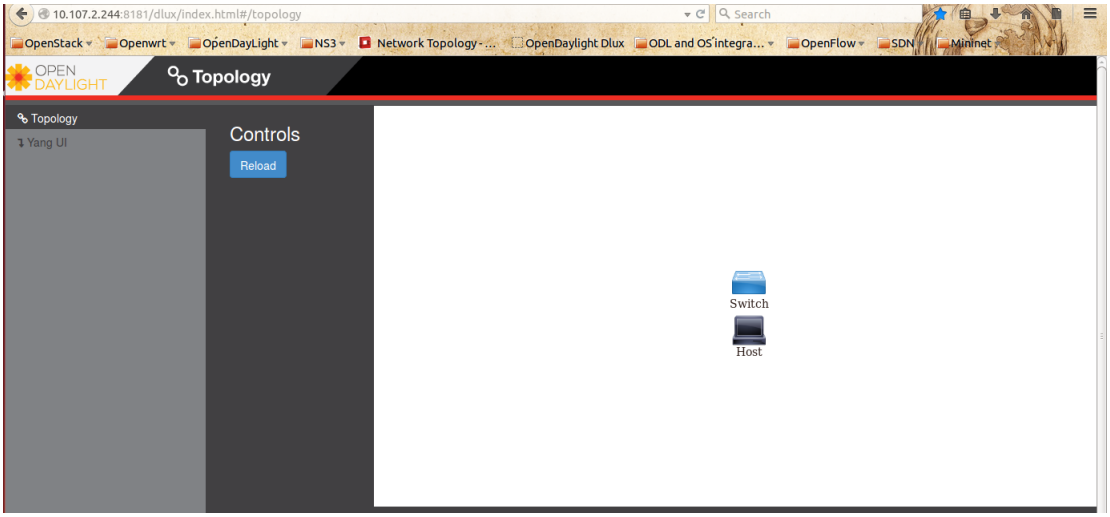


Figure 2.4: ODL User Interface (Dlux)

1. Details of nodes, node connectors and statistics information present in Model Driven Service Abstraction Layer (MD SAL) inventory RestConf API.

2. Visual representation of nodes and hosts in network topology of MD-SAL. Hosts information is tracked by L2-switch project.

3. Yang UI - Web interface to see the yang models present in controller. This is a very powerful feature that allows user to perform operations like adding or deleting flows on the yang model.

4. Ported features such as Connection manager, flows, static routes.

### 2.2.2 SNMP4SDN

To make ODL controller communicate with the ethernet switches, a new plugin called SNMP4SDN is developed. These switches are cheap and programmable to some extend through SNMP and CLI. They can also report their status to their administrative computer i.e ODL controller.

For the controller to be able to configure the switches, it first needs to know which switches are under it. It also needs to know the topology of the switches. Simple Network Management Protocol (SNMP) and Link Layer Discovery Protocol (LLDP) play a major role in this process.

A code is written in Java which helps in discovering a switch and knowing its

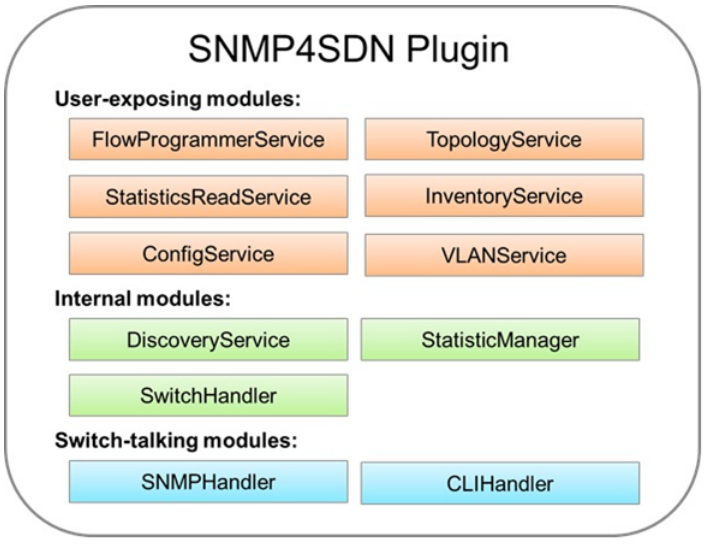


Figure 2.5: SNMP4SDN modules [9]

topology. The code is divided into parts called modules, each one dedicated to a

single purpose. These modules can be summarised as is in figure 2.5 on page 12 and SNMP4SDN as a whole is a plugin to ODL project. Its internal architecture is shown in figure 2.6 on page 13.

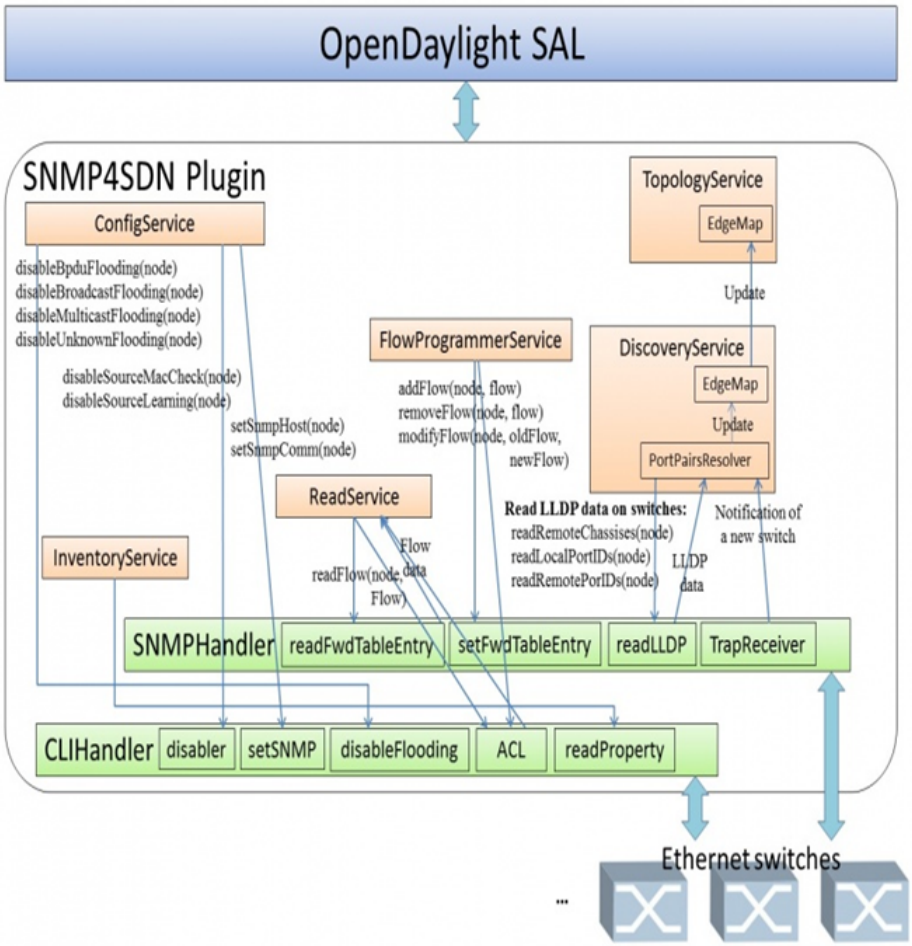


Figure 2.6: SNMP4SDN Internal Architecture [8]

#### 2.2.2.1 Switch Discovery

Its a mechanism through which switches tell the controller that I am under you. This is achieved by configuring every switch with its SNMP host i.e controller so that the switch can send trap to the controller at boot-up.

This way the controller will know of the switch when it boots up. One thing to be taken care is the community in the trap sent by the switch must match that

of the controller provided controller has one otherwise the controller will reject the switch.

This is depicted pictorially in figure 2.7 on page 14.

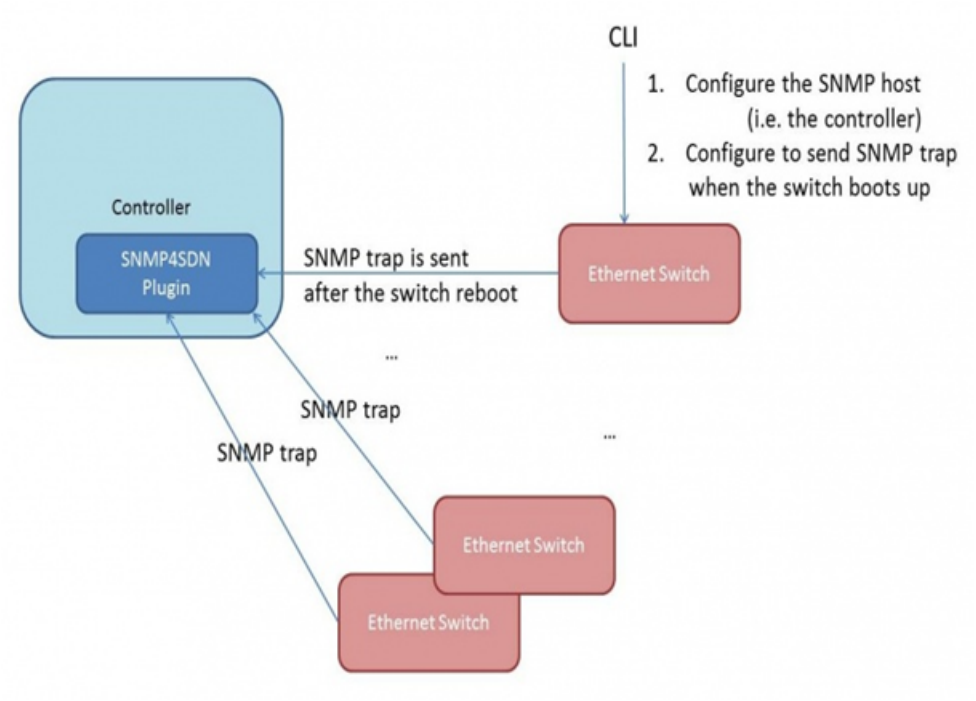


Figure 2.7: Switch Discovery by Traps [10]

#### 2.2.2.2 Topology Discovery

Discovering the topology among the switches is also crucial. This is done by reading LLDP data from each switch. The plugin queries each switch, switch replies with the LLDP data and the topology among them is resolved. LLDP data contains each switchs port id, the port id and chassis id of the neighbour switch. This process is pictorially represented in figure 2.8 on page 15.

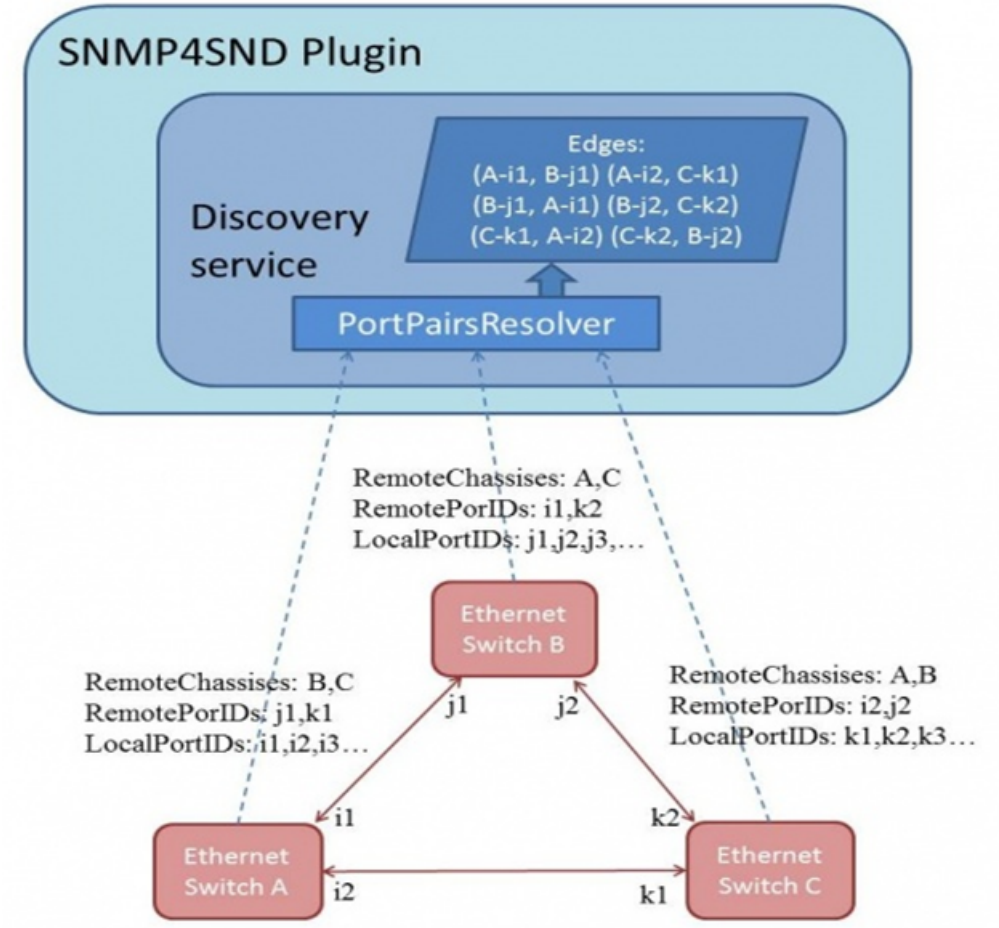


Figure 2.8: Topology Discovery [11]

### 2.2.3 Benchmarking ODL

#### 2.2.3.1 CBench

Performance is an important issue, be it a software or a hardware. Performance is used for benchmarking and for continuously improving the code after studying the collected statistics. One such tool is CBench. It is very simple and written in C by Rob Sherwood. It is a program used for testing OpenFlow controllers by generating packet-in events for new flows. It emulates a bunch of switches. These switches connect to a controller and send packet-in messages and watch for flow-mods to get pushed down. CBench measures various performance issues related to flow setup time. It emulates a configurable number of OpenFlow switches that all communi-

cate with a single OpenFlow controller. Each emulated switch sends a configurable number of new flow (Open-Flow packet in) messages to the OpenFlow controller and waits for the appropriate flow setup (OpenFlow flow mod or packet out) responses. It records the difference in time between request and response. Cbench supports two modes of operation: latency and throughput mode. In latency mode, each emulated switch maintains exactly one outstanding new flow request, waiting for a response before soliciting the next request. Latency mode measures the Open-Flow controllers request processing time under low-load conditions. By contrast, in throughput mode, each switch maintains as many outstanding requests as buffering will allow, that is, until the local TCP send buffer blocks. Hence, throughput mode measures the maximum flow setup rate that a controller can maintain. Its algorithm is [13]:

```
Algorithm:


pretend to be n switches (n=16 is default)
create n openflow sessions to the controller
if latency mode (default):
    for each session:
        1) send up a packet in
        2) wait for a matching flow mod to come back
        3) repeat
        4) count how many times #1-3 happen per sec
else in throughtput mode (i.e., with '-t'):
    for each session:
        while buffer not full:
            queue packet_in's
            count flow_mod's as they come back
<snip>
```

#### 2.2.3.2 Issues with CBench

1. It is not actively maintained.
2. It is poorly documented.
3. It doesnot contain nay unit tests.
4. It outputs unstructed results to standard output.
5. It is single threaded.
6. It has limited code path.
7. The code is not readable and hard to maintain. Much of the code could be replaced by Python stdlib calls.

#### 2.2.3.3 WCBench

ODL is also tested for its performance through a wrapper around CBench called *wcbench* which is developed and maintained by Daniel Farrell at Red Hat. Its whole code is available at Git Hub [14]. It initially supported only Fedora OS. Support for Ubuntu 14.04 was added by me. WCBench is a Python wrapper around CBench. It is a collection of scripts which runs CBench on ODL, stores output in *results.csv* file and plots graphs of the results. *.csv* format was chosen because its the same format that is consumed by the Jenkins Plot Plugin. ODL uses this plugin to automatically run a subset of the functionality provided by WCBench against ODL builds.

It contains many three scripts:

1. *./wcbench.sh [options]* is a shell script which
   - Checks for the OS and installs OS specific CBench.
   - Installs ODL controller.
   - Configures and starts ODL.

- Optionally pins ODL to given number of processors.
- Optionally runs CBench for the given number of minutes. depending on the *[options]* given and stores output in *.csv* file. This also contains a field for specifying ODL controller host IP and port in case the controller is on another machine and tests are being run from another machine.

2. *./loop_wcbecnh.sh [options]* is a shell script which runs WCBench in a loop. Depending on *[options]*, it will

   - Loop WCBench runs without restarting ODL for given number of times
   - Loop WCBench runs, restarting ODL between runs for given number of times
   - Run WCBench for a given number of minutes
   - Pin ODL to given number of processors

   and stores output in *.csv* file.

3. *./stats.py [options]* is a Python script which gives out statics of the above two scripts ran. It takes *result.csv* file, compute stats about the collected data and outputs on the console or a graph depending on the options given.

   The parameters in the *results.csv* file for each run includes:

   - The name of the controller under test
   - The IP address of the ODL controller
   - A run number for each run, starting at 0 and counting up
   - The number of CPUs on the system running ODL
   - The flows/sec average from the CBench run
   - Human-readable time that the run finished

- Unix time (in seconds) at the beginning of the run
- Unix time (in seconds) at the end of the run
- The TESTS_PER_SWITCH value passed to CBench
- The number of switches simulated by CBench
- The number of MAC addresses used by CBench
- The duration of each test in milliseconds
- The total RAM on the system running ODL
- The used RAM on the system running ODL at the end of a test run
- The free RAM on the system running ODL at the end of a test run
- The steal time on the system running ODL at the start of the test
- The steal time on the system running ODL at the end of the test
- The one minute load of the system running ODL
- The five minute load of the system running ODL
- The fifteen minute load of the system running ODL
- The iowait value at the start of the test on the system running ODL
- The iowait value at the end of the test on the system running ODL

## 2.3 OpenFlow

OpenFlow (OF) is a programmable network protocol. It is designed to manage and direct traffic among routers and switches from various vendors. It allows a server to instruct network switches as to where to send packets. Its a standardised protocol for

interacting with the forwarding behaviour of switches from multiple vendors. It is a controller to switch protocol that runs over Transport Layer Security or unprotected Transport Control Protocol (TCP) connection[15]. There are basically two types of messages namely

1. Controller to switch
    - Specify how packets are to be forwarded.
    - Configures parameters such as VLAN priorities.
2. Switch to controller
    - Inform the controller when links go down or when a packet arrives with no specified forwarding instruction.

### 2.3.1 Message Format

The OF message format is as shown in the figure 2.9 on page 20. The various

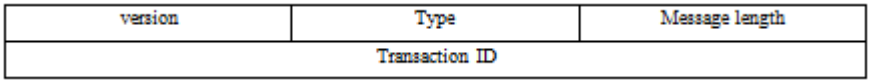


Figure 2.9: OpenFlow Message Format [15]

files of the message format are as explained below.

- **Version:** This filed indicates version of OFP.
- **Type:** It describes types of messages. As of OF specifications 3.4.1, there are 29 types of messages.
- **Message length:** It is the length of header and payload in octets.
- **Transaction ID:** This id is used to control transaction of packets and matching requests and responses.

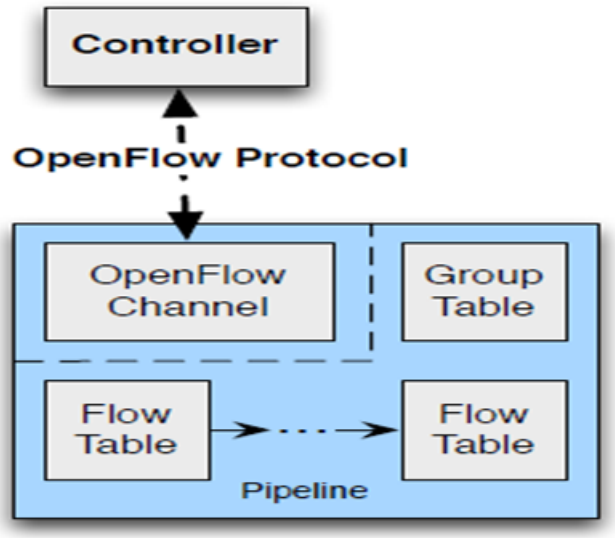


Figure 2.10: OpenFlow Switch [15]

### 2.3.2 OF Switch Components

An OpenFlow Logical Switch consists of one or more flow tables and a group table, which perform packet lookups and forwarding, and one or more OpenFlow channel to an external controller. The switch communicates with the controller and the controller manages the switch via the OpenFlow switch protocol.

OpenFlow-compliant switches come in two types namely

1. OpenFlow-only: They support only OpenFlow operation, in those switches all packets are processed by the OpenFlow pipeline, and can not be processed otherwise.

2. OpenFlow-hybrid switches: They support both OpenFlow operation and normal Ethernet switching operation, i.e. traditional Layer2 Ethernet switching, VLAN isolation, Layer3 routing (IPv4 routing, IPv6 routing...).

### 2.3.3 OF Pipeline

The OpenFlow pipeline of every OpenFlow Logical Switch contains one or more flow tables, each flow table containing multiple flow entries. The OpenFlow pipeline

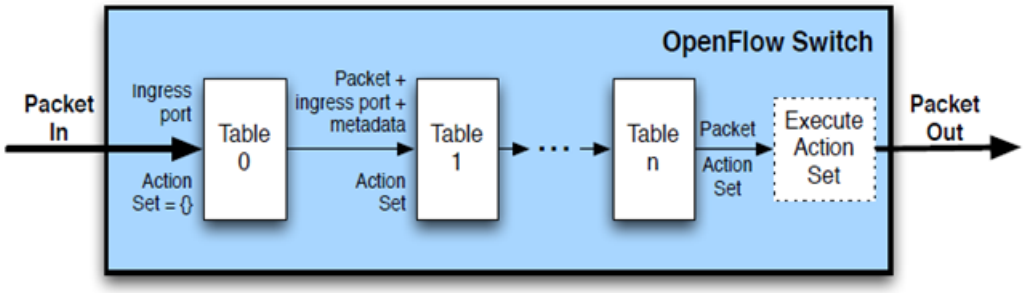


Figure 2.11: OpenFlow Pipeline Processing [15]

processing defines how packets interact with those flow tables as depicted in figure 2.11 on page 22. An OpenFlow switch is required to have at least one flow table, and can optionally have more flow tables. An OpenFlow switch with only a single flow table is valid, in this case pipeline processing is greatly simplified. The details of flow table matching is explained in figure 2.12 on page 22.

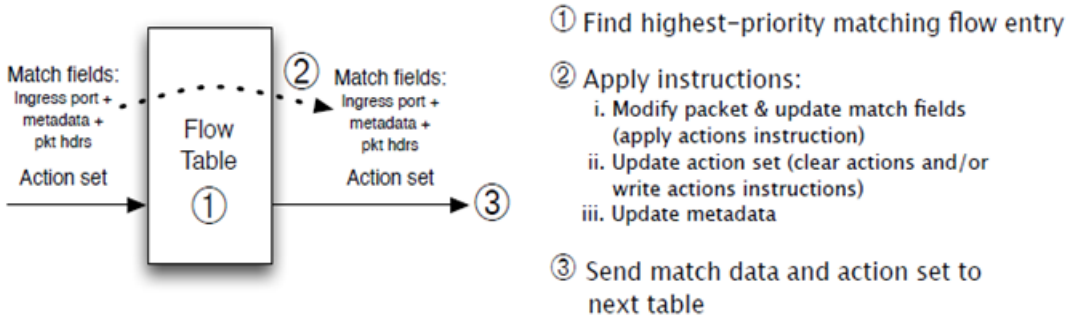


Figure 2.12: OpenFlow Flow Table [15]

The flow tables of an OpenFlow switch are sequentially numbered, starting at 0. Pipeline processing always starts at the first flow table. The packet is first matched against flow entries of flow table 0. Other flow tables may be used depending on the outcome of the match in the first table.

When processed by a flow table, the packet is matched against the flow entries of the flow table to select a flow entry. If a flow entry is found, the instruction set included in that flow entry is executed. These instructions may explicitly direct

the packet to another flow table, where the same process is repeated again. A flow entry can only direct a packet to a flow table number which is greater than its own flow table number, in other words pipeline processing can only go forward and not backward. Obviously, the flow entries of the last table of the pipeline can not include the Goto-Table instruction. If the matching flow entry does not direct packets to another flow table, pipeline processing stops at this table, the packet is processed with its associated action set and usually forwarded.

If a packet does not match a flow entry in a flow table, this is a table miss. The behaviour on a table miss depends on the table configuration. The instructions included in the table-miss flow entry in the flow table can flexibly specify how to process unmatched packets, useful options include dropping them, passing them to another table or sending them to the controllers over the control channel via packet-in messages. This whole process id depicted in figure 2.13 on page 23

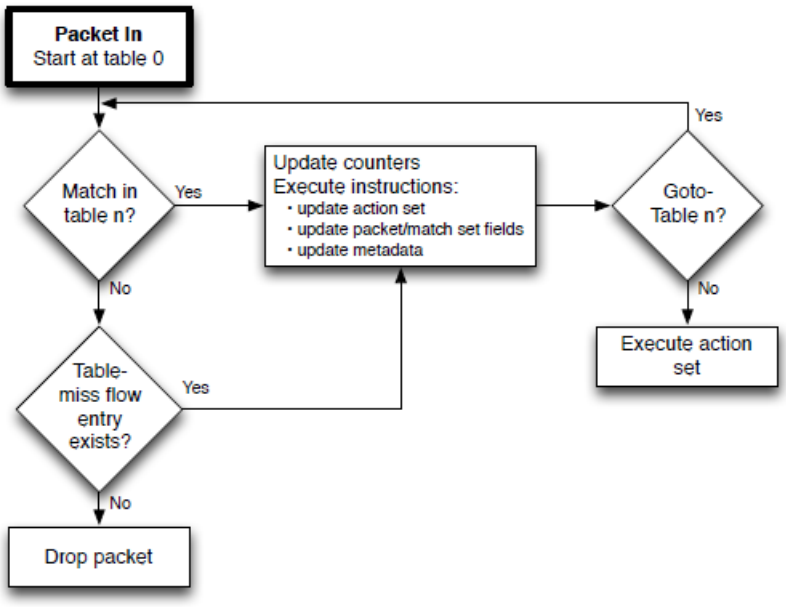


Figure 2.13: Flow Table entry Matching [15]

### 2.3.4 Flow Table

A flow table contains of flow entries. Each flow entry will have various fields. These fields are as explained below.

1. **match fields:** to match against packets. These consist of the ingress port and packet headers, and optionally other pipeline fields such as metadata specified by a previous table.
2. **priority:** matching precedence of the flow entry.
3. **counters:** updated when packets are matched.
4. **instructions:** to modify the action set or pipeline processing.
5. **timeouts:** maximum amount of time or idle time before flow is expired by the switch.
6. **cookie:** opaque data value chosen by the controller. May be used by the controller to filter flow entries affected by flow statistics, flow modification and flow deletion requests. Not used when processing packets.
7. **flags:** flags alter the way flow entries are managed, for example the flag OFPFF_SEND_FLOW_REM triggers flow removed messages for that flow entry.

These fields are also depicted in figure 2.14 on page 24 A flow table entry


Figure 2.14: Flow Table Fields [15]

is identified by its match fields and priority: the match fields and priority taken together identify a unique flow entry in a specific flow table. The flow entry that wildcards all fields (all fields omitted) and has priority equal to 0 is called the table-miss flow entry.

## 2.4 Open vSwitch

Open vSwitch (OVS) is an OF capable virtual switch. They are typically used with hypervisors to interconnect virtual machines (VMs) within a host and between different hosts across networks. They are suitable for multi-server deployment. Few of its features are:

- VLAN tagging
- standard spanning tree protocol
- Quality of Service control

### 2.4.1 OVS architecture

OVS switches contains three main components:

1. **vswitchd:** it is an OVS daemon.
2. **ovsdb-server:** it is a database and stores ovs configurations.
3. **kernel module:** it currently supports Linux.

OVS switches can be compiled from source available at [20]. Its latest release is 2.3.1.

## 2.5 CPqD

CPqD is an independent institute in Brazil focusing on Information and Communcation technology. CPqD switch was develop at this institute with Ericsson TrafficLab 1.1 softswitch implementation as a base and having support for OF 1.3. The source could be found at GitHub repository [21]. It has following components:

1. **ofdatapath:** implementation of the siwtch.

2. **ofprotocol:** secure channel which connects the controller to the switch.

3. **oflib:** a library for converting to/from 1.3 wire format

4. **dpctl:** a tool for configuring the switch from the console.

Its installation steps could be found at [21].

## 2.6 Mininet overview

Mininet is an OpenFlow enabled network emulator. It supports variety of networking devices like switches, end-hosts, routers, and links on a single Linux kernel. A Mininet host acts like a real machine. It can be ssh'ed into. Just like real networks, link speed and bandwidth can also be spcified in Mininet. Applications like iperf also work in Mininet. It has a provision of making a host a HTTP server through a python script.

In short, Mininet's virtual hosts, switches, links, and controllers are the real thing they are just created using software rather than hardware and for the most part their behavior is similar to discrete hardware elements. It is usually possible to create a Mininet network that resembles a hardware network, or a hardware network that resembles a Mininet network, and to run the same binary code and applications on either platform. It does all this just with a simple command.

### 2.6.1 Positives of Mininet

Mininet combines many of the best features of emulators, hardware testbeds, and simulators. These are summarised as below [18].

1. **Compared to full system virtualization based approaches,** Mininet:

   - **Boots faster:** It takes seconds to boost up instead of minutes.
   - **Scales larger:** A network of hundreds of hosts and switches can be created.

- **Provides more bandwidth:** Typically 2 Gbps of total bandwidth is provided on modest hardware.
- **Installs easily:** A prepackaged VM is available that runs on VMware or VirtualBox for Mac/Win/Linux with OpenFlow v1.0 tools already installed.

2. **Compared to hardware testbeds, Mininet**

   - Is inexpensive and always available.
   - Is quickly reconfigurable and restartable.

3. **Compared to simulators, Mininet**

   - Runs real, unmodified code including application code, OS kernel code, and control plane code (both OpenFlow controller code and Open vSwitch code).
   - Easily connects to real networks.
   - Offers interactive performance - it can be typed at.

It creates a network wth 1 switch and 2 hosts attached. The switch is connection to the internal OpenFlow controller.

```
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>
```

### 2.6.2 Installing Mininet

There are three ways to install Mininet as discussed below. One of the ways used is described below. Details for other ways of installation could be found at [19].

1. **Mininet VM installation** This is the easiest and most foolproof approach. Follow the steps below
   - Download Mininet VM image from here.
   - Download and install a virtualization system.
   - In virtualbox, go to file ->import and import the ovf file that was downloaded.
2. **Native installation from source**
3. **Installation from Packages**

Once downloaded login with username and password as ***mininet***.

### 2.6.3 Limitations

Mininet, though being a great help in exploring SDN concept, has limitations as listed below [18]

1. Mininet-based networks cannot (currently) exceed the CPU or bandwidth available on a single server.

2. Mininet cannot (currently) run non-Linux-compatible OpenFlow switches or applications; this has not been a major issue in practice.

3. Running on a single system is convenient, but it imposes resource limits: if the server has 3 GigaHertz (GHz) of Central Processing Unit (CPU) and can switch about 3 Gigabits per second (Gbps) of simulated traffic, those resources will need to be balanced and shared among virtual hosts and switches.

4. Mininet uses a single Linux kernel for all virtual hosts; this means that softwares depending on BSD, Windows, or other operating system kernels cannot be run.

5. Mininet won't write an OpenFlow controller; a controller is to be developed or found if custom routing or switching behavior is needed.

6. Currently Mininet doesn't do Network Address Translation (NAT) out of the box. This means that virtual hosts will be isolated from Local Area Network (LAN) by default; this is usually a good thing, but means hosts can't talk directly to the Internet unless a means to do so is provided for them.

7. Currently all Mininet hosts share the host file system and PID space; this means that care must be taken while running daemons requiring configuration in /etc, and in killing the processes.

8. Unlike a simulator, Mininet doesn't have a strong notion of virtual time; this means that timing measurements will be based on real time, and that faster-than-real-time results (e.g. 100 Gbps networks) cannot easily be emulated.

9. Since the Mininet hosts share the filesystem, there might be a need to create different configuration files for each Mininet host and specify them as startup options if a specific configuration for a program is needed.

10. Sharing filesystem also leads to file collision if same file is created in the same directory.

# Chapter 3

# Dwelling into OpenStack

## 3.1 Understanding on Cloud Computing

Cloud Computing !!!!!!!! Its a big name companies are moving into now a days. Splitting the word and defining it:

Cloud is a collection of servers - all linked to each another. Computing means working with these servers.

In simple terms, Cloud computing is accessing, developing, editing, providing and storing all data and applications from any place in the world. The only minimum requirement the cloud ask for is The Internet with a computer.

Now that it is clear what cloud is, the question arises of setting up and using the cloud. Based on that, clouds are mainly of 3 types:

1. **Private:** Big Companies setup their own cloud with all the required hardware and IT team for its maintenance.

2. **Public:** If a company don't want to spend money on the setup and maintenance, they will rent a cloud for their use and pay the rent to the cloud provider according to its usage. All the headache of the IT team, up time of

cloud and its maintenance will be taken care of by the cloud provider.

3. **Hybrid:** Suppose, some company knows that the product it is selling on its website has high traffic during 2 particular months in a year and the website is rarely used in other months during the year. At this point of time, the concept of hybrid cloud comes into picture. Hybrid Cloud is a combination of both public and private cloud in which some services are rented by using public cloud and other services reside in private cloud. The company rent the cloud for two months to satisfy their customers and handle the traffic from their private cloud during the rest of the year.

But what are these clouds used for? What do we get from these clouds?
Basically, there are three main services that are provided by vendors in the cloud :

1. **Infrastructure-as-a-service:** Abbreviated as IaaS, this is the service that resides in the most bottom layer of all the other services. The name itself suggests that in this service, the vendor provides the whole infrastructure (Storage, networking & access) for usage or to upload to web for customers. They maintain all the hardware and you just need to pay them the rent according to your usage.
   Examples of IaaS vendors are Amazon EC2, S3, OpenStack etc... This thesis describes various projects under OpenStack, setting up of OpenStack infrastructure and its usage.

2. **Platform-as-a-Service:** Coming exactly on top of IaaS is Platform-as-a-Service, abbreviated as PaaS. PaaS is specifically for Software Developers. What cloud computing companies like Google do is, they provide the platform of different languages like Java, Python etc. to develop an app and charge for these in return. Benefit of this service is that the deployment, upgradation and testing of the app becomes a lot easier.
   Example of PaaS companies are Google AppEngine, Salesforce etc...

3. **Software-as-a-Service:** A lot more of people now a days might be using Dropbox, Google Drive etc for storing their photos and their other personal stuff. That is nothing but SaaS known as Software As A Service. So this term will not need any more explanation.
   Even checking email is also an example of using cloud.

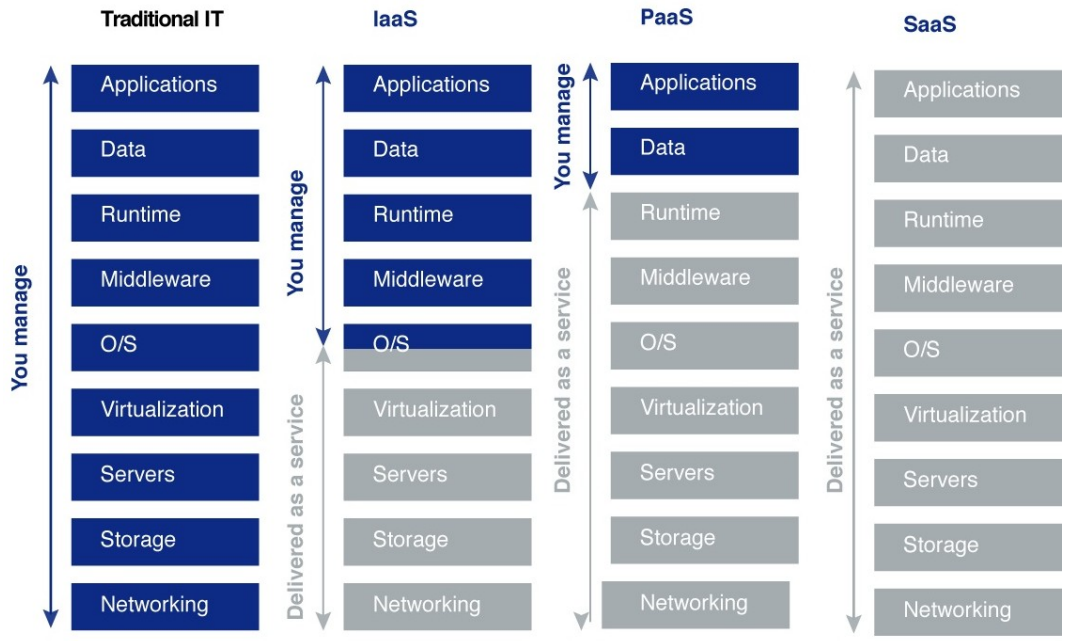


Figure 3.1: Understanding Cloud Computing services and difference with Traditional Computing [24]

Understanding on these services are very clear from the figure 3.1 on page 32 Control over the cloud is maximum for a private cloud, ofcourse and mininum for public cloud because it is controlled by a cloud vendor. Level of abstraction for more for SaaS and less for IaaS. Usage flexibility is more in IaaS as with a VM anything can be done as opposed to SaaS, wherein only that could be achieved that the service provides. These concepts are pictorially depicted in figure 3.2 on page 33.

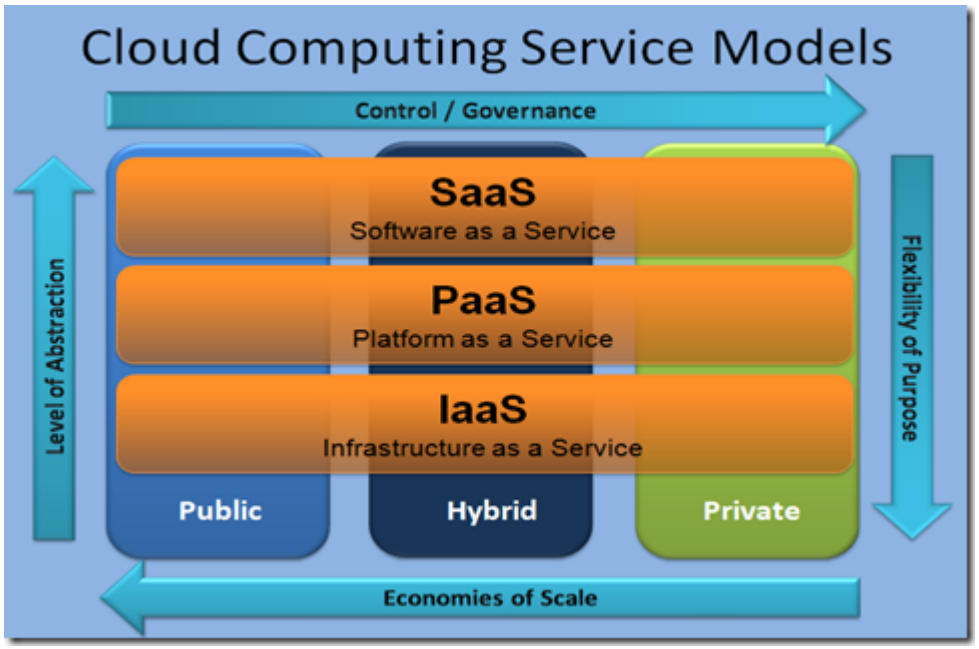


Figure 3.2: Cloud Computing models and services [23]

### 3.1.1 Cloud for Developers

A Cloud for Developers? More and more companies now-a-days are trying to provide their product,service or support on the cloud. In other sense, it could be said that they are becoming more developer friendly by taking their point of view into considersation. For a developer moving to cloud saves lots of time, hardware and efforts when compared to traditional software development methods.

Lets start with a step by step analysis.

To write a code, an editor is needed. On-cloud editors like Google App Engine, Cloud 9 are available with command predictions and highlighting to assist writing. Since the code is on cloud, another person will also be able to view/edit the code remotely. A group of people can contribute to the same code increasing its efficiency. Since the code is on cloud, it coulde be sdited any time at any place having just a computer connected to Internet. Editors are accompanied by compilers too. Say a windows user and want to interpret a python code. Python interpreter will have to be installed. This is not a problem. Problem arises when unfortunately all the drives

of the computer are full. Python interpreter can't be installed. What could be the possbile solutions now? Buy an external hard disk or increase RAM of computer. Both these options are hardware dependent and incure cost. Another option is - Move to cloud, simple. This reduces OS or hardware dependency for compiling the code plus the code becomes omnipresent.

On-cloud project management tools like SalesForce are also available that ease tracking the progress of leads and the team. The progress can be communicated with the client as well.

Clouds also give the facility of storing the code and the results. Say results were stored on external hard disk and it got corrupted or got lost. These results need to be shown to the client and laptop's battery dies. Because the data is safely stored on cloud, one can immediately log into his/her cloud account from the client's computer and continue with the presentation or retrieve the lost data.

Moving further from this, after the coding is done, the next stage the developer enters into is the testing of the program or app. Again there are alternatives as to whether a developer wants to test it locally or on the cloud. Some companies like Bugzilla also provides facility to keep a report of bug tracking at various points of time during app development which helps the developer or a group of developers in an organization to reduce the downtime or error in their application.

Also, as an app developer if app is running on cloud, the cloud will take care of the scalability as the demand for the app increases.Some companies have even gone one step further than this. They provide the console to the developers so that they can keep a continuous tracking of the traffic on their app and are also provided with some amount of control on their app during their run-time.

In a nutshell, there is a cloud solution for each and everything a developer needs to in writing a software, deploy it easily and make it available for their clients in the fastest and easiest possible manner. It promotes agile software development methodology.

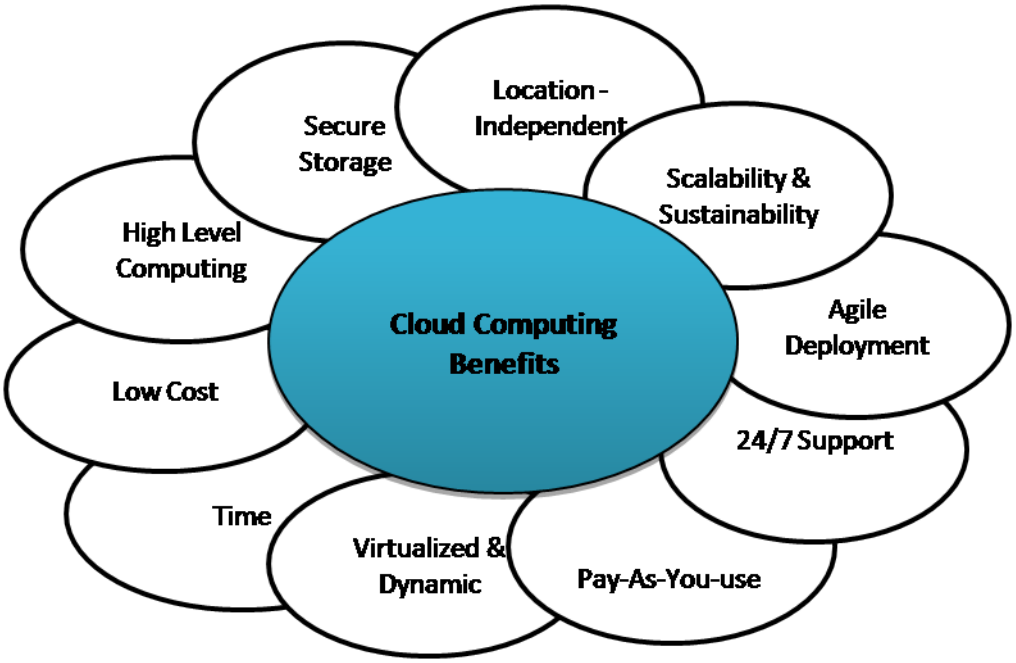


Figure 3.3: Advantages of Cloud Computing [25]

#### 3.1.1.1 Advantages of Cloud Computing

From the above description, it is very evident that cloud could ease the process of developing and managing an app. There are numerous other advantages of cloud computing depending on the industry of interest. Few common advantages common to all can be viewed in figure 3.3 on page 35.

## 3.2 OpenStack

OpenStack is a cloud computing software platform. It is also free and open source. It is primarily deployed as an Infrastructure as a Service (IaaS) solution. It is a group of various interrelated projects that control creating VMs, storing data and images, and networking resources managed through a web-based dashboard, a RESTful API or command-line tools. It is released under the terms of the Apache License by openstack.org.

OpenStack project began in 2010 as a joint venture of Rackspace Hosting and

NASA. It is currently managed by the OpenStack Foundation. It is a non-profit corporate entity established in September 2012 to promote OpenStack software and its community. It is a community of 200+ companies, including Cisco, Huawei, Arista Networks, Qosmos, AT&T, Ericsson, AMD, SolidFire, Avaya, Juniper Networks, Citrix, Intel, Dell, Mirantis, Dreamhost, Red Hat, EMC, Go Daddy, Hewlett-Packard, IBM, Pure Storage, Internap, Oracle, Mellanox, Acelio, NEC, NetApp, VMware, Nexenta, Canonical, PLUMgrid, SUSE Linux, VMTurbo and Yahoo!.

Till date (April 2015) 11 releases of OpenStack are released named alphabetically as Austin, Bexar, Cactus, Diablo, Essex,Folsom, Grizzly, Havana, Icehouse, Juno and Kilo released on 30 April 2015. This thesis describes OpenStack Icehouse release. Icehouse was released on 17 April 2014 and includes following components

1. **Nova:** Called Compute. Hosts VMs and manages their life cycle.

2. **Glance:** Called Image service, it stores as well as retrieves virtual machine disk images. Compute uses these images while creating an instance.

3. **Swift:** Called Object storage, it stores and retrieves unstructured data objects via a RESTful, HTTP based API.

4. **Horizon:** It is on OpenStack dashboard written in Python using Django framework. It facilitates creation and deletion of VMs, Networks, subnets, routers. It shows information about the running instances.

5. **Keystone:** Called Identity service, it is an authorization and authentication service of OpenStack. All API calls are authorised by it.

6. **Neutron:** Called Networking service, it enables network connectivity as a service for other OpenStack services, such as OpenStack Compute. Provides an API for users to define networks and the attachments into them. Has a pluggable architecture that supports many popular networking vendors and technologies.

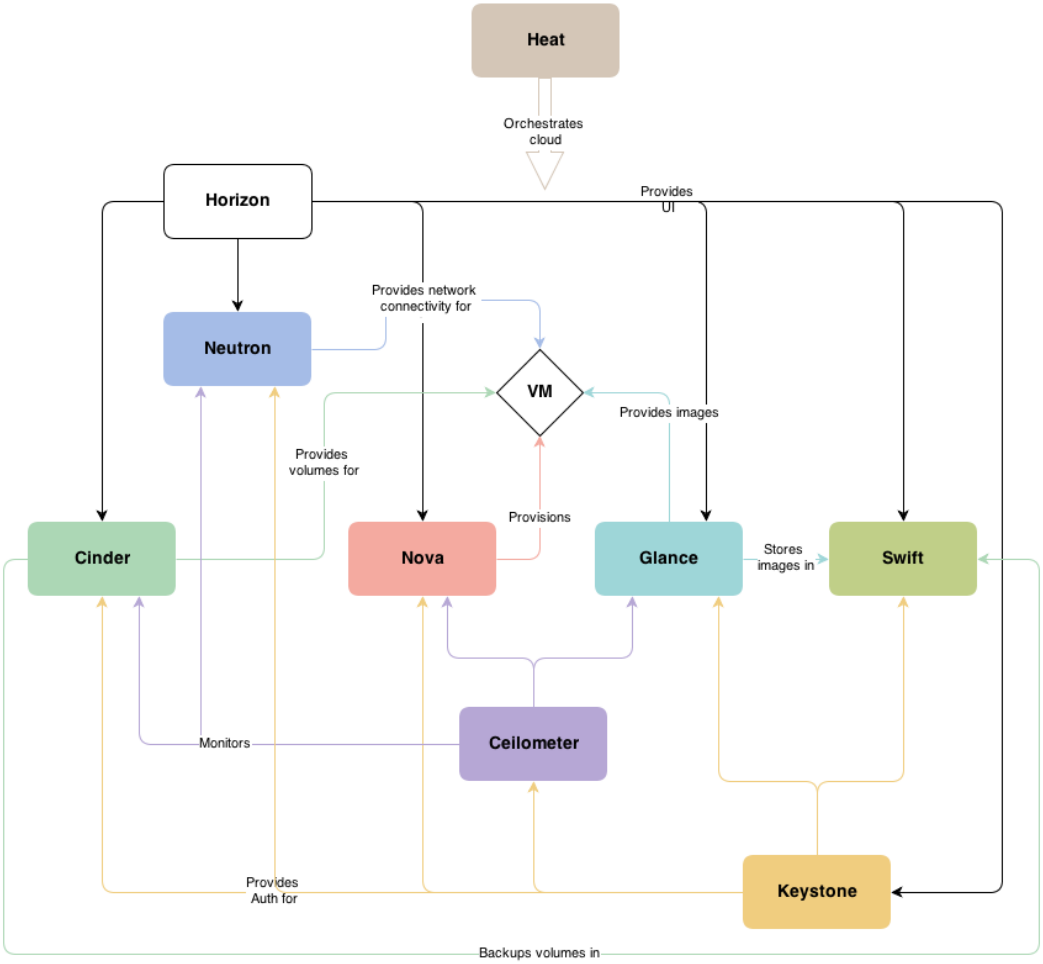


Figure 3.4: Conceptual architecture of OpenStack [26]

7. **Cinder:** Called Block storage, it provides persistent block storage to instances in running mode.

8. **Heat:** Called orchestration, it manages the complex system of OpenStack.

9. **Ceilometer:** Called Telemetry, it is used for billing scalability and benchmarking of OpenStack clouds.

10. **Trove:** It is a Database-as-a-Service of OpenStack.

These components and their inter-connectivity is shown in the figure 3.4 on page 37.

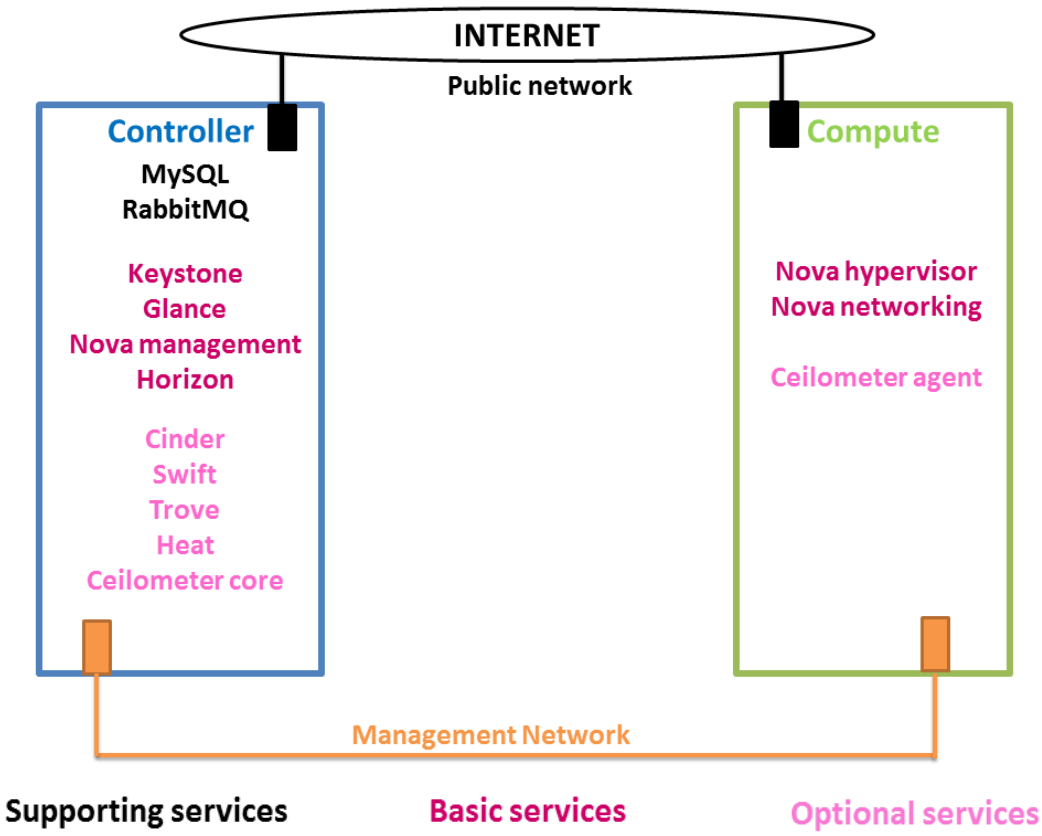


Figure 3.5: Nova network architecture

### 3.2.1 Architecture of OpenStack

Two types of architecture are very popular in OpenStack, 2 node with Nova network and 3 node with Neutron network. These nodes need a minimal physical requirements.

Services needed and installed in the nodes for these architectures are depicted in figure 3.5 on page 38 for Nova network and in figure 3.6 on page 39 for Neutron network. The figures also highlight the connections between the nodes. Nova network didnot handle complex network topologies. It supports three simple kinds of network topologies namely Flat, Flat DHCP and VLAN. From the Folsom release of OpenStack, Neutron network was introduced. It was earlier named Quantum but renamed Neutron due to clash with other company's name. It facilitates creation of

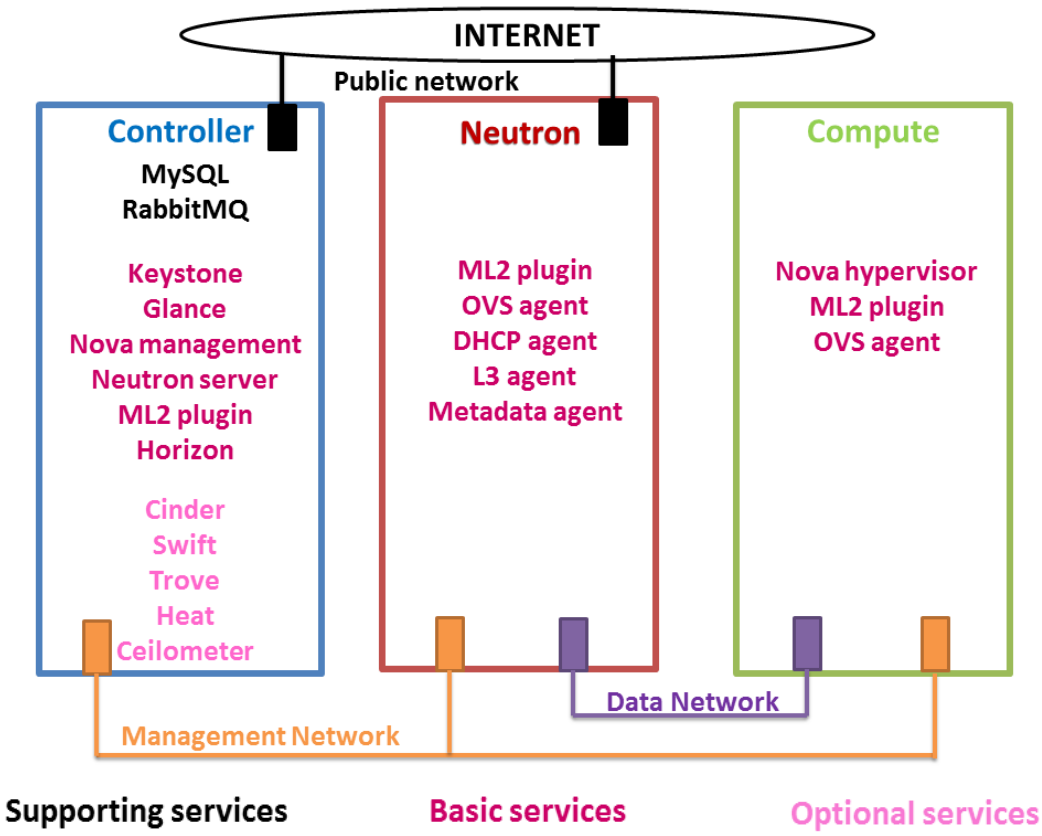


Figure 3.6: Neutron network architecture

network, subnets and ports.

### 3.2.2 Types of network

OpenStack uses 3 distinct networks:

1. **Management network:** It is used for inter process communication. Messaging service like Rabbitmq, databases like MySql uses this network for communication. It should be ensured and isolated from other networks.

2. **Data network:** this network is used by instances to talk to each other. L3 services and DHCP services are too accessed through this network. This should be isolated and secured. It is to be mapped to underlying physical network. Physical network could be of Flat type or VLAN type. Variable *bridge_mapping* in

file *ml2_conf.ini*.

3. **External network:** it exposes API services of Nova and Glance to consumers outside OpenStack. It also allows instances to communicate with Internet and public network using floating IP.

Layer 2 isolation could be achieved through 3 techniques:

1. VLAN
2. VxLAN
3. Global Routing Encapsulation (GRE) tunnel

### 3.2.3 Floating IP

OpenStack setup is in a private network. Instances are given IP address from DHCP agent of Neutron and are accessed within OpenStack using this private IP. When instances need to communicate through public network, they use new IP called *floating IP*. Each instance will have 2 IPs, private and floating. Neutron assigns the IP. It neither uses any DHCP service nor being assigned statically. It is Neutron L3 agent's responsibility to route the packets having floating IP to correct destination.

# Chapter 4

# Working with ODL

This chapter discusses a step by step procedure followed to implement the problem statement. The first section describes interacting with Mininet. It shows how to create a network attached to an ODL controller in Mininet. It shows how by a single command a switch could be converted into a firewall. The second section describes the usage of SNMP4SDN plugin of ODL project. Third section implements bandwidth throttling through ODL. Section 4 runs benchmark on ODL and its result is displayed. Section 5 compares SDN with traditional network in NS3 simulator.

## 4.1 Working with Mininet

ODL controller acts as a control plane and can control underlying network devices remotely. Mininet talks to controller on port 6633 for OF 1.1 and on port 6653 for OF 1.3 version. The following was the procedure followed in using controller to control a network in mininet.

1. The controller was downloaded in a zip file from [28]. The file was unzipped and the karaf batch file from the *bin* directory was run on windows machine. The controller was prepared to communicate with Mininet by installing the features

```
opendaylight-user@root>feature:install odl-mdsal-apidocs
odl-restconf odl-dlux-core odl-l2switch-switch odl-l2switch-ui
odl-openflowplugin-flow-services
```

These features will install dlux and L2 switch capabilities. The switches on mininet and hosts connected to them will be seen on dlux. **l2switch-switch** feature is needed to make controller communicate with OF switches created in Mininet.

2. Mininet was installed as a VM as discussed in Mininet VM installation section in Installing Mininet. Mininet talks to ODL controller with OF protocol. The hosts are labelled h1,....hN and switches as s1,....sN. An IP address assigned to Mininet VM was found by *ifconfig* command.

Important thing to be remembered is creating a Host-only adapter named VirtualBox Host-Only Ethernet Adapter. Putty was used to remotely to log into Mininet with username ***mininet*** and password ***mininet***. X11 forwarding was enabled and Xming was installed to allow X-forwarding. This allowed to run Wireshark in Mininet so as message flow between controller and Mininet can be viewed. Once the set up was completed, a network with linear topology was created. The network had 3 switches with a host connected to each switch. The switches are OVS switches. The command for creating this network was given as shown in figure 4.1 on page 43. The connection can be pictorially shown in figure 4.2 on page 43.

–mac option will assign the mac address similar to IP address of the switch. The switches configuration can be seen from the command prompt like shown in figure 4.3 on page 44. The links between hosts and switches can be viewed by command ***dump***. Many more commands can be used. Three of them are shown in figure 4.4 on page 44. ODL will install few default flows into the switches. These flows are shown in figure 4.5 on page 45. Since the ODL controller has a simple forwarding application, the hosts were able to ping each other. This is depicted in figure 4.5

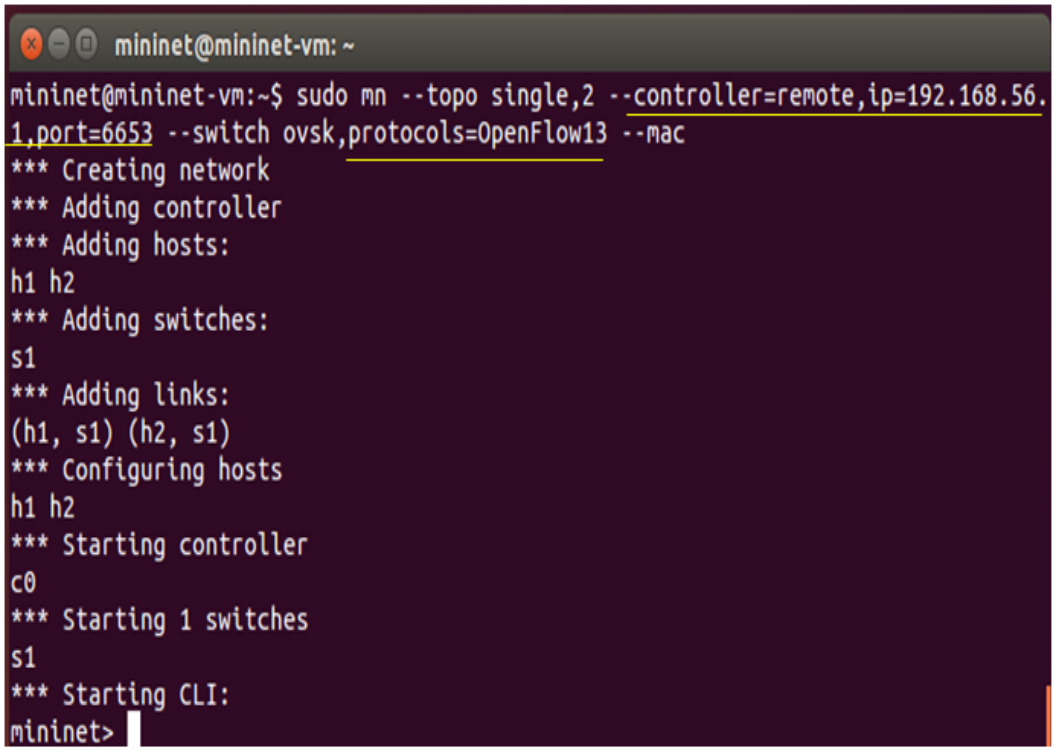


Figure 4.1: Creating a network in Mininet

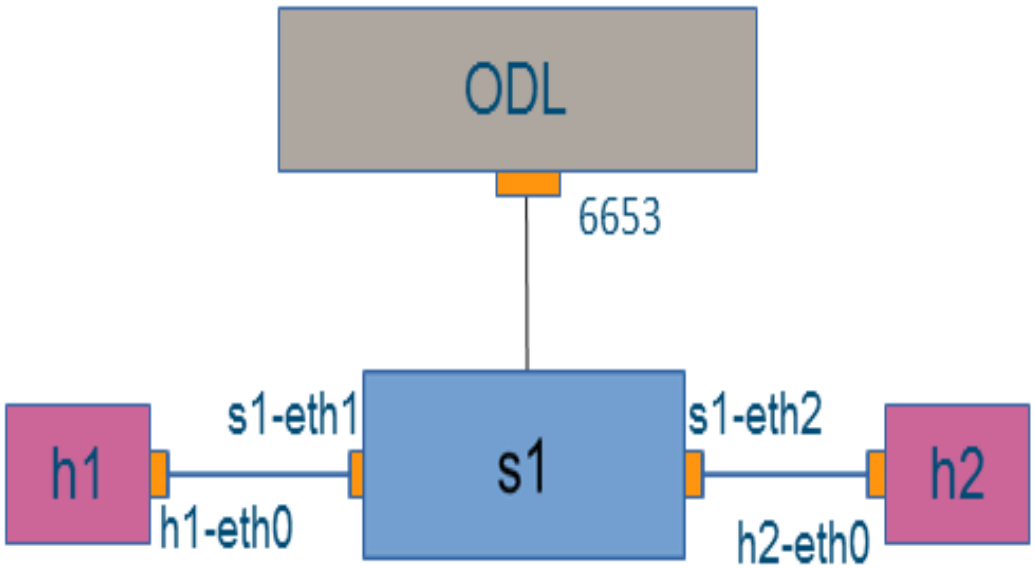


Figure 4.2: Linear topology in Mininet

```
mininet> sh ovs-vsctl show
aad071f2-ee72-47c0-a963-8cb5c033ad1b
    Bridge "s1"
        Controller "ptcp:6634"
        Controller "tcp:192.168.56.1:6653"
            is_connected: true
        fail_mode: secure
        Port "s1"
            Interface "s1"
                type: internal
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1-eth2"
            Interface "s1-eth2"
    ovs_version: "2.0.2"
mininet>
```

Figure 4.3: Switch configuration in Mininet

```
mininet@mininet-vm: ~
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth2
s2 lo:  s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s2-eth3
c0
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=1696>
<Host h2: h2-eth0:10.0.0.2 pid=1699>
<Host h3: h3-eth0:10.0.0.3 pid=1701>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1706>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=1709>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=1712>
<RemoteController c0: 192.168.56.1:6633 pid=1689>
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s2-eth1 (OK OK)
h3-eth0<->s3-eth1 (OK OK)
s2-eth2<->s1-eth2 (OK OK)
s3-eth2<->s2-eth3 (OK OK)
mininet>
```

Figure 4.4: Links among hosts

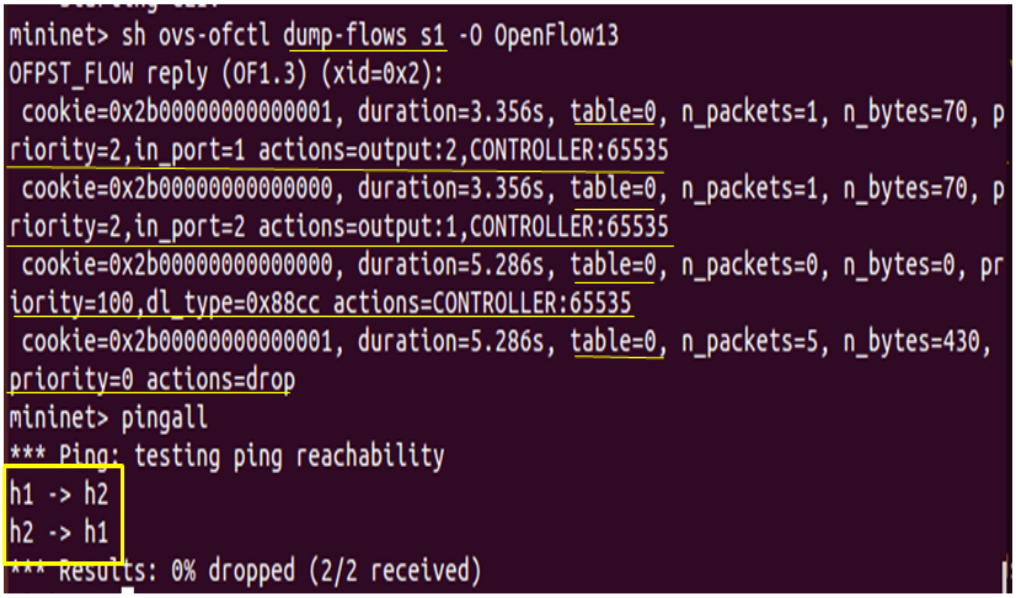


Figure 4.5: Default flows in the switches

on page 45. When the hosts ping each other, the switch is not aware of the hosts. It goes to the controller to ask fo the course of action. This is *packet-in* message. The controller is aware of the network topology and thus directs the switch s1 to output all packets for host h2 on port 2. These flows are shown in figure 4.6 on page 46. The aim is to stop in-flow or out-flow of packets from host h3. So the switch s3 is to be instructed to drop the messages it receives ob port 1. This is done by commanding the switch as

```
mininet> sh ovs-ofctl add-flow s3 in_port=1,actions=drop
```

ofctl is a command line tool to add/delete/modify flows/VLAN information in OF switches. This flow can be added in the switch through ODL too. For this, the flow is first added into ODL either through POSTMAN REST API client or through Yang UI. Yang UI can be accessed from dlux. POSTMAN plugin needs to be added in Chrome web browser and works offline. A code in xml is to be written and *PUT* into the switch. The switch's address with the flow table is given.

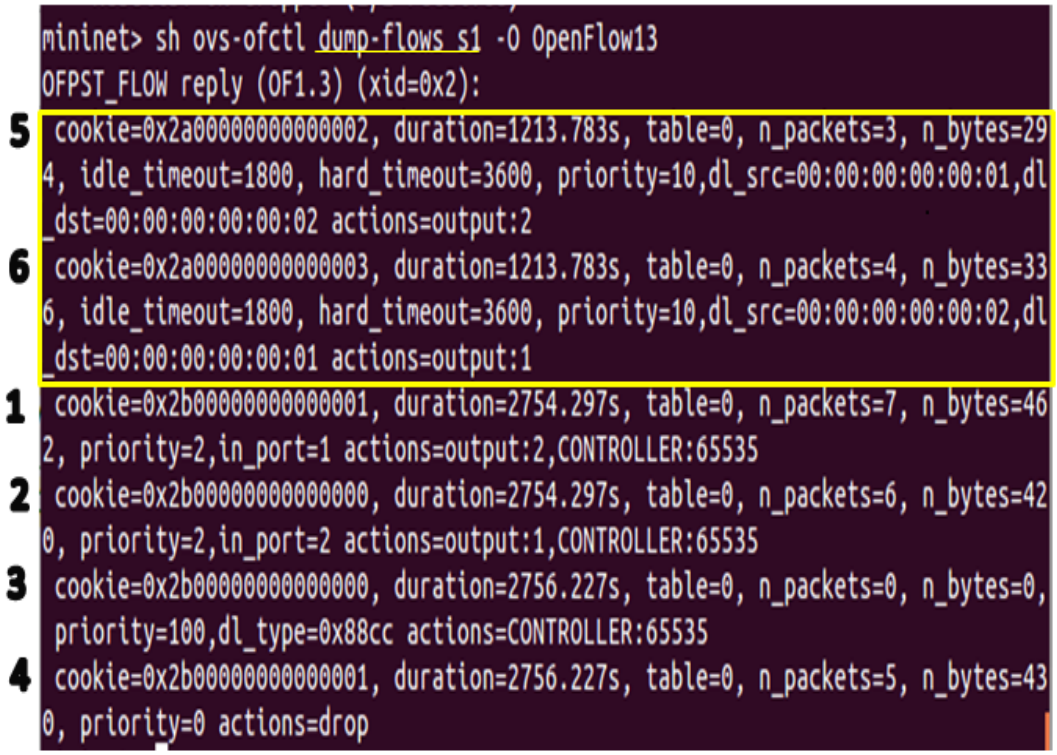


Figure 4.6: Flows after ping

```
localhost:8181/restconf/config/opendaylight-inventory/nodes
/openflow/1/table/0/flow/4
```

Headers saying that the content type and acceptable content is application type in xml are specified.

```
Content-Type: application/xml
Accept: application/xml
```

The xml code to be written in POSTMAN is:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <priority>1000</priority>
    <flow-name>h2drop</flow-name>
    <id>4</id>
    <table_id>0</table_id>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                   <order>0</order>
                   <drop-action/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
</flow>
```

The flow is stored in *config database* of ODL. It is then transmitted to the switch. The switch adds it to its flow table numbered 0 with flow id = 4 and priority = 1000. The flow tells switch 1 to drop all the packets coming on any port.

### 4.1.1 Custom topologies in Mininet

Two topologies namely mesh and ring were also created using python. The connectivity among the hosts was tested by pinging all hosts. The steps to create a topology is summarised below.

1. Using mn command
    - Goto /home/mininet/mininet/custom
    - Write a python file for the required topology eg:mesh.py
    - In the same directory, run the command sudo mn –custom mesh.py –topo mytopo –controller=remote,ip=192.168.56.1
2. without using mn command
    - Goto /home/mininet/mininet/examples
    - Write a python code eg. ring.py
    - In the same directory, run the script sudo python ring.py

## 4.2 Working with Ethernet switch

A flow for dropping a packet was successfully added into a virtual OpenFlow switch. The same thing was needed to be done with real Ethernet switches. ODL plugin SNMP4SDN was used for this purpose. Since there were no unused switches able, Linux computer was used to act as a switch. The plugin discovered the switches by identifying the traps and LLDP data sent by switches.

| Issue:Ethernet switch was unavailable |
| --- |
| Solution:<br>Used Linux computer as a switch. Net-SNMP agent was installed to extend the SNMP agent in the computer and code was written to make it return dummy values. |

### 4.2.1 Configuring SNMP in Linux Computer

The switch were configured to send LLDP data and SNMP data by installing and starting their respective daemons using below commands.

```
controller1@janki:~$ sudo apt-get install snmp snmpd lldpd
```

Since computer needed to reply to SNMP queries sent by controller, an agent called Net-SNMP agent was needed. The agent was downloaded as pointed out below.

1. Downloaded the source file from http://www.net-snmp.org/download.html and unzipped it.

2. From the folder, ran configure file with embedded perl.
   This enabled use of mib2c tool to extend the agent.

   ```
   ./configure --enable-embedded-perl
   make
   sudo make install
   ```

3. The agent was configured to send traps to controller at boot-up and to reply with values of OIDs asked by controller by writing ***snmp.conf*** and ***snmpd.conf***. These can be generated using config utility of Net-SNMP agent by a command as shown below.

   ```
   snmpconf –g basic_setup
   ```

### 4.2.2 Using SNMP4SDN plugin

The SNMP4SDN feature was installed into Helium controller by typing following commands in karaf console.

```
opendaylight-user@root>feature: install feature:install
odl-adsal-northbound odl-snmp4sdn-all
```

Now the controller is needed to be informed of the switches under it along with their CLI username, passwords, Medium Access Control (MAC) address and IP address. This information was written in a .csv file and placed on Desktop. An example of this file is shown in figure 4.7 on page 50 The file is read into controller by the below

```
MAC,IP,SNMP_Community,CLI_Username,CLI_Password
90:94:e4:23:13:e0,192.168.0.32,private,admin,password
90:94:e4:23:0b:00,192.168.0.33,private,admin,password
90:94:e4:23:0b:20,192.168.0.34,private,admin,password
```

Figure 4.7: SwitchDB.csv file [9]

command.

```
opendaylight-user@root>snmp4sdn:ReadDB
\home\controller1\Desktop\SwitchDB.csv
```

The topology is discovered by following command.

```
opendaylight-user@root>snmp4sdn:TopoDiscover
```

This command asked for chassis Id and Port Id of the computer by using get requests. These information is present in LLDP-MIB as *lldpLocChassisId* and *lldpLocPortId* respectively. Their OID is *.1.0.8802.1.1.2.1.3.2.0* and *.1.0.8802.1.1.2.1.3.7.1.3* respectively. Since the computer did not have these value in-built, it did not reply with any data.

The solution to this is to write a C program to extend the agent. For this the LLDP-MIB is to be added into the agent. The steps taken for this are listed below.

1. The default directory where MIBs are searched was found by the following command

   ```
   net-snmp-config --default-mibdirs
   ```

   One such directory was ***usr local share snmp mibs***

2. LLDP-MIB was downloaded and copied into the above folder.

3. Agent was told to include the above MIB by writing the below line in the snmp.conf file situated at /usr/local/etc/snmp/snmp.conf.

***mibs +ALL***

4. A c and h file to make agent return a dummy lldpLocChassisId was written using mib2c tool by the command below.

```
mib2c-c mib2c.scalar.conf lldLocChassisId
```

   Since lldpLocChassisId has only one instance, it is a scalar. Thus scalar configuration file was used in the above command.

5. A dummy value was added into the lldLocChassisId.c generated.

6. c and h files were moved into the folder. /net-snmp-5.7.2/agent/mibgroup

7. The agent was reconfigured to include the lldpLocChassisId files generated by following commands.

```
./configure --with-mib-modules="lldpLocChasssisId"
make
sudo make install
```

8. It was tallied that the files were configured into agent correctly by

   - Checking that the file /agent/mibgroup/mib_modules_inits.h mentioned lldpLocChassisId.
   - Adding a line to print ***lldpLocChassisId*** in the initialization function in lldpLocChassisId.c file.

9. The agent was restarted by command ***snmpd***.

The agent still did reply with the dummy chassis id value. The flow of messages between the ODL controller and Linux computer is shown in figure 4.8 on page 52.

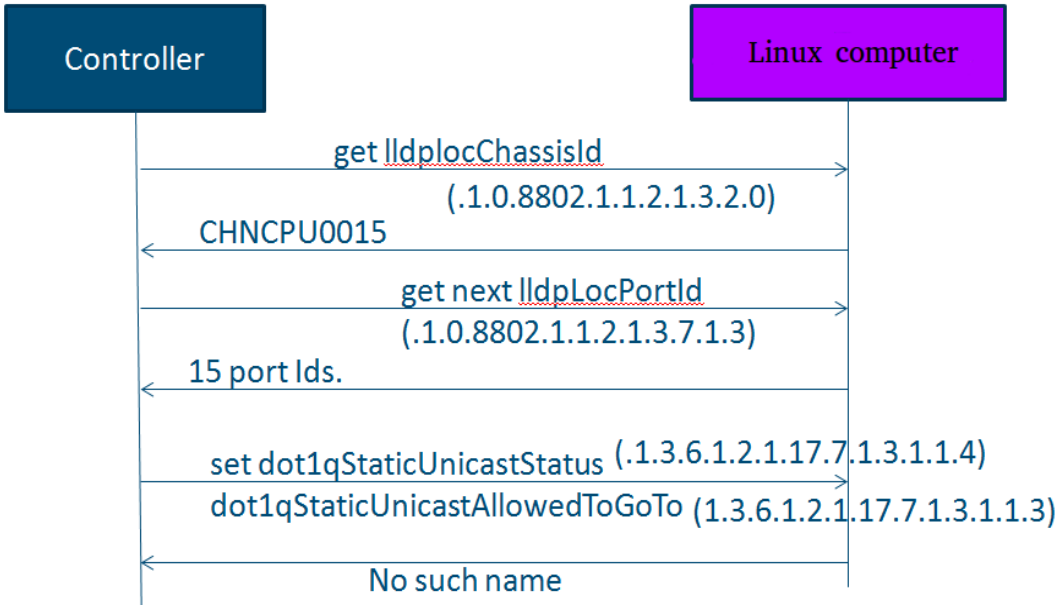


Figure 4.8: Exchange of messages between ODL and Computer

## 4.3 Bandwidth throttling through ODL

Bandwidth throttling is a concept of limiting bandwidth to desired value. It is a Quality of Service (QoS) factor. These are configured into the switch through ODL by defining meters. OVS switch is installed through its package.

```
controller1@janki:~$ sudo apt-get install openvswitch-switch
```

Course of action taken to connect OVS to ODL are:

1. Create a bridge named *mybridge*.

```
controller1@janki:~$ sudo ifconfig h1 up
```

2. OVS switches are managed by ODL controller by the command.

```
controller1@janki:~$ sudo ovs-vsctl set-manager \
tcp:ip_of_controller:6640
```

3. To make OVS connect to internet, attach one of its port to computer's internet stack and the physical interface of computer to OVS.

4. Attach 2 tap interfaces to OVS switch.

```
controller1@janki:~$ sudo ip tuntap add mode tap h1
controller1@janki:~$ sudo ip tuntap add mode tap h2
controller1@janki:~$ sudo ifconfig h1 up
controller1@janki:~$ sudo ifconfig h2 up
controller1@janki:~$ sudo ovs-vsctl add-port mybridge h1
controller1@janki:~$ sudo ovs-vsctl add-port mybridge h2
```

5. Check the bridge configurations.

```
controller1@janki:~$ sudo ovs-vsctl show
9243e0d7-44fb-42f7-b206-a51687c9978f
    Bridge mybridge
        Port "h2"
            Interface "h2"
        Port mybridge
            Interface mybridge
                type: internal
        Port "h1"
            Interface "h1"
    ovs_version: "2.0.2"
```

Start two Ubuntu VMs and attach them to these tap interfaces *h1* and *h2* in bridged mode. These connections are pictorially shown in figure 4.9 on page 55. Now that the physical setup is ready, lets get into its programming. Define a meter of desire and write its xml code.

```
<meter
    xmlns="urn:opendaylight:flow:inventory">
    <meter-id>1</meter-id>
    <container-name>mymeter</container-name>
    <meter-name>mymeter</meter-name>
    <flags>meter-kbps</flags>
    <meter-band-headers>
        <meter-band-header>
            <band-id>0</band-id>
            <band-rate>50000</band-rate>
            <meter-band-types>
                <flags>ofpmbt-drop</flags>
            </meter-band-types>
            <band-burst-size>0</band-burst-size>
            <drop-rate>50000</drop-rate>
            <drop-burst-size>0</drop-burst-size>
        </meter-band-header>
    </meter-band-headers>
</meter>
```

This meter limits the bandwidth to 50000 kbps. Put this to address of the switch.

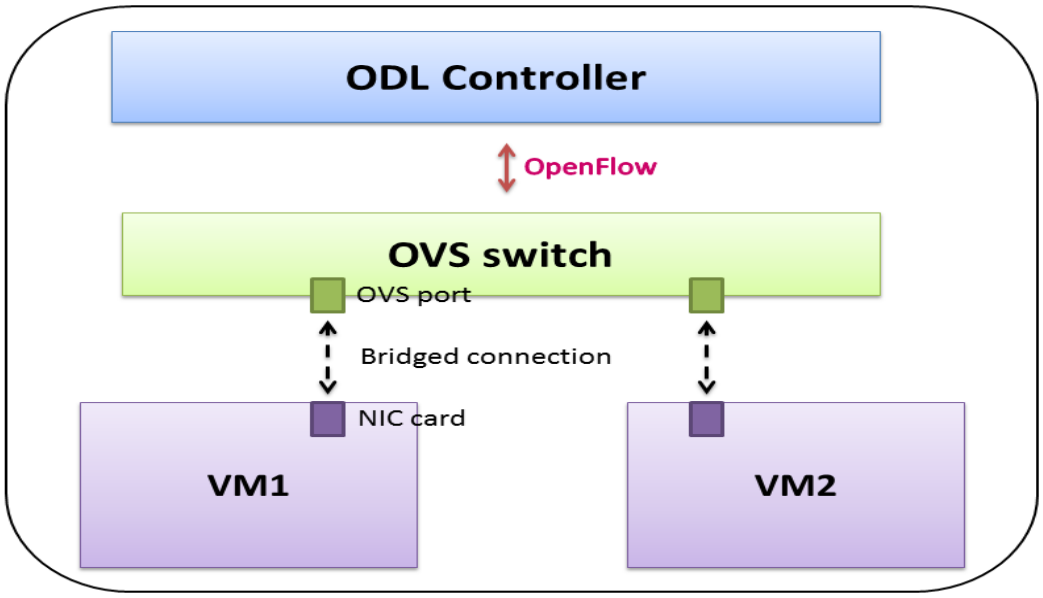


Figure 4.9: OVS connected to ODL and VMs attached

```
PUT http://ip:8181/restconf/config/opendaylight-inventory:nodes
/node/openflow:1/meter/1
```

Give another REST call to put a flow with defined meter.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
  <priority>2</priority>
    <hard-timeout>0</hard-timeout>
    <idle-timeout>0</idle-timeout>
    <flow-name>dl_dst-tb7-p2p4</flow-name>
```

```
<cookie_mask>255</cookie_mask>
<cookie>7</cookie>
 <match>
    <ethernet-match>
      <ethernet-destination>
            <address>6a:e9:2d:37:39:29</address>
      </ethernet-destination>
    </ethernet-match>
 </match>
<id>10</id>
<table_id>0</table_id>
<instructions>
   <instruction>
     <order>0</order>
        <meter>
            <meter-id>1</meter-id>
        </meter>
    </instruction>
    <instruction>
        <order>1</order>
        <apply-actions>
            <action>
               <order>0</order>
                <output-action>
                    <output-node-connector>openflow:1:1
                    </output-node-connector>
                    <max-length>60</max-length>
                </output-action>
```

```
                </action>
            </apply-actions>
        </instruction>
    </instructions>
</flow>
```

*ethernet-destination* is the MAC address of the VM. This flow can be seen in OVS and defined meter is also configured. But the bandwidth was not limited. After searching for the issue, it was found that OVS switch, at the moment doesnot support meter configurations [29] [30].

| Issue: OVS switches donot support meter configurations through ODL |
|---|
| Solution: Use another soft switch called CPqD |

### 4.3.1 Using CPqD switch

Its installation steps are summarised below:

1. Install dependencies.

```
controller1@janki:~$ sudo apt-get install git-core autoconf \
automake autotools-dev pkg-config make gcc g++ libtool \
libc6-dev cmake libpcap-dev libxerces-c2-dev unzip \
libpcre3-dev flex bison libboost-dev
```

2. Downgrade bison to get NetBee to compile correctly.

```
controller1@janki:~$ wget -nc http://de.archive.ubuntu.com \
/ubuntu/pool/main/b/bison/bison_2.5.dfsg-2.1_amd64.deb
```

```
controller1@janki:~$ http://de.archive.ubuntu.com/ubuntu/ \
pool/main/b/bison/libbison-dev_2.5.dfsg-2.1_amd64.deb
controller1@janki:~$ sudo dpkg -i bison_2.5.dfsg-2.1_amd64.deb
libbison-dev_2.5.dfsg-2.1_amd64.deb
controller1@janki:~$ rm bison_2.5.dfsg-2.1_amd64.deb \
libbison-dev_2.5.dfsg-2.1_amd64.deb
```

3. Install and compile NetBee.

```
controller1@janki:~$ wget -nc http://www.nbee.org/download
/nbeesrc-jan-10-2013.zip
controller1@janki:~$ unzip nbeesrc-jan-10-2013.zip
controller1@janki:~$ cd nbeesrc-jan-10-2013/src
controller1@janki:~$ cmake .
controller1@janki:~$ make
controller1@janki:~$ sudo cp ../bin/libn*.so /usr/local/lib
controller1@janki:~$ sudo ldconfig
controller1@janki:~$ sudo cp -R ../include/* /usr/include/
controller1@janki:~$ cd ../..
```

4. Clone CPqD switch from its Git repository.

```
controller1@janki:~$ git clone https://github.com/CPqD/ \
ofsoftswitch13.git
controller1@janki:~$ cd ofsoftswitch13
controller1@janki:~$ git checkout \
d174464dcc414510990e38426e2e274a25330902
```

```
controller1@janki:~$ ./boot.sh
controller1@janki:~$ ./configure
controller1@janki:~$ make
controller1@janki:~$ sudo make install
controller1@janki:~$ cd ..
```

Create tap interfaces connect them to the switch and bridge them to VMs. Connect the switch to ODL. The instructions are summarised below.

```
controller1@janki:~$ sudo udatapath/ofdatapath \
--datapath-id=000000000001 --interfaces=h1,h2 ptcp:6680 &
controller1@janki:~$ secchan/ofprotocol tcp:127.0.0.1:6680 \
tcp:ip_of_controller:6653
```

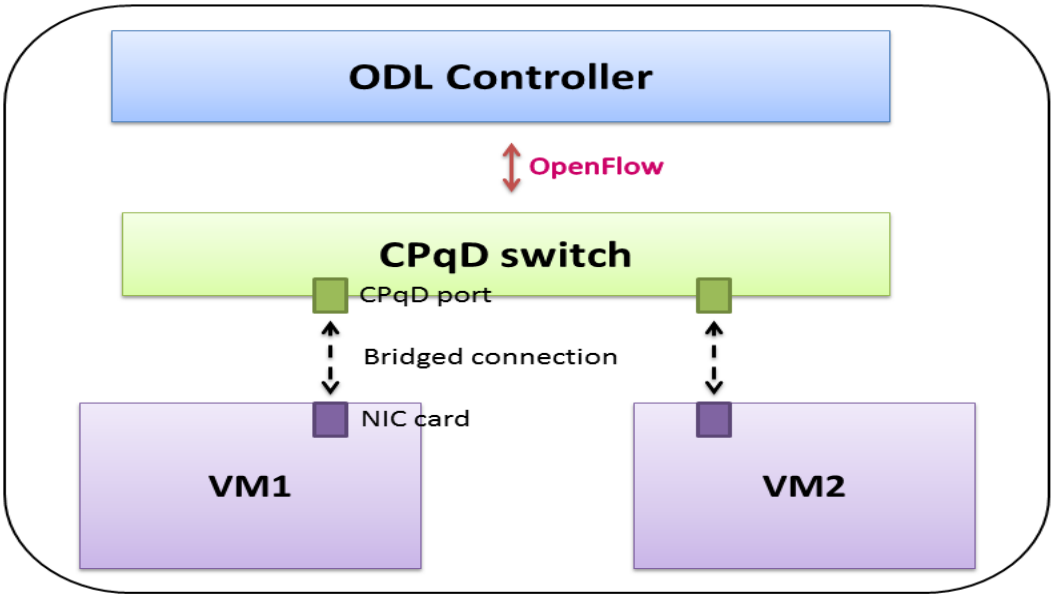


Figure 4.10: CPqD switch attached to ODL and VMs

The connection is shown in figure 4.10 on page 59. Apply the same REST calls for meter and its associated flow as described above here. Ping the hosts and notice

the difference in RTT.

```
Before meter:   average RTT = 1.030 ms
After meter:    average RTT = 1.089 ms
```

## 4.4 WCBench

WCBench is to be cloned form its Git repository. The steps followed are summarised below:

```
controller1@janki:~$ git clone https://www.github.com/dfarrell07 \
/wcbench
Cloning into 'wcbench'...
remote: Counting objects: 565, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 565 (delta 0), reused 0 (delta 0), pack-reused 561
Receiving objects: 100% (565/565), 157.94 KiB | 223.00 KiB/s, done.
Resolving deltas: 100% (299/299), done.
Checking connectivity... done.
controller1@janki:~$ cd wcbench/
controller1@janki:~/wcbench$ ./wcbench.sh -c
./wcbench.sh: 47: ./wcbench.sh: declare: not found
./wcbench.sh: 50: ./wcbench.sh: Syntax error: "(" unexpected
```

| Issue: Declare not found and Syntax error: |
| --- |
| Solution:<br>Changed shebang line from "#!/usr/bin/env sh" to "#! /bin/bash" |

```
controller1@janki:~/wcbench$ ./wcbench.sh -c
CBench is installed
controller1@janki:~/wcbench$ ./wcbench.sh -i
Installing OpenDaylight dependencies
Downloading OpenDaylight Helium 0.2.3
Unzipping OpenDaylight Helium 0.2.3
odl-openflowplugin-flow-services added to features installed
at boot
odl-openflowplugin-drop-test added to features installed at
boot
controller1@janki:~/wcbench$ ./wcbench.sh -o
Starting OpenDaylight
Will repeatedly attempt connecting to Karaf shell until it's ready
Issued 'dropAllPacketsRpc on' command via Karaf shell to
localhost:8101
Issued 'log:set ERROR' command via Karaf shell to localhost:8101
controller1@janki:~/wcbench$ ./wcbench.sh -r
Collecting pre-test stats
Running CBench against ODL on localhost:6633
Collecting post-test stats
Collecting time-irrelevant stats
Average responses/second: 29787.57
/home/controller1/results.csv not found or empty, building fresh
one
controller1@janki:~/wcbench$ ./stats.py -g ram flows
```

This generates the graph with *number of runs* on x-axis and *RAM used (in MB)* and *Flows per second* on y-axis. Total RAM used in the above run is around 3600 MB. Flows added to the switch by the controller per second are around 30,000 as

depicted in graph 4.11 on page 62.

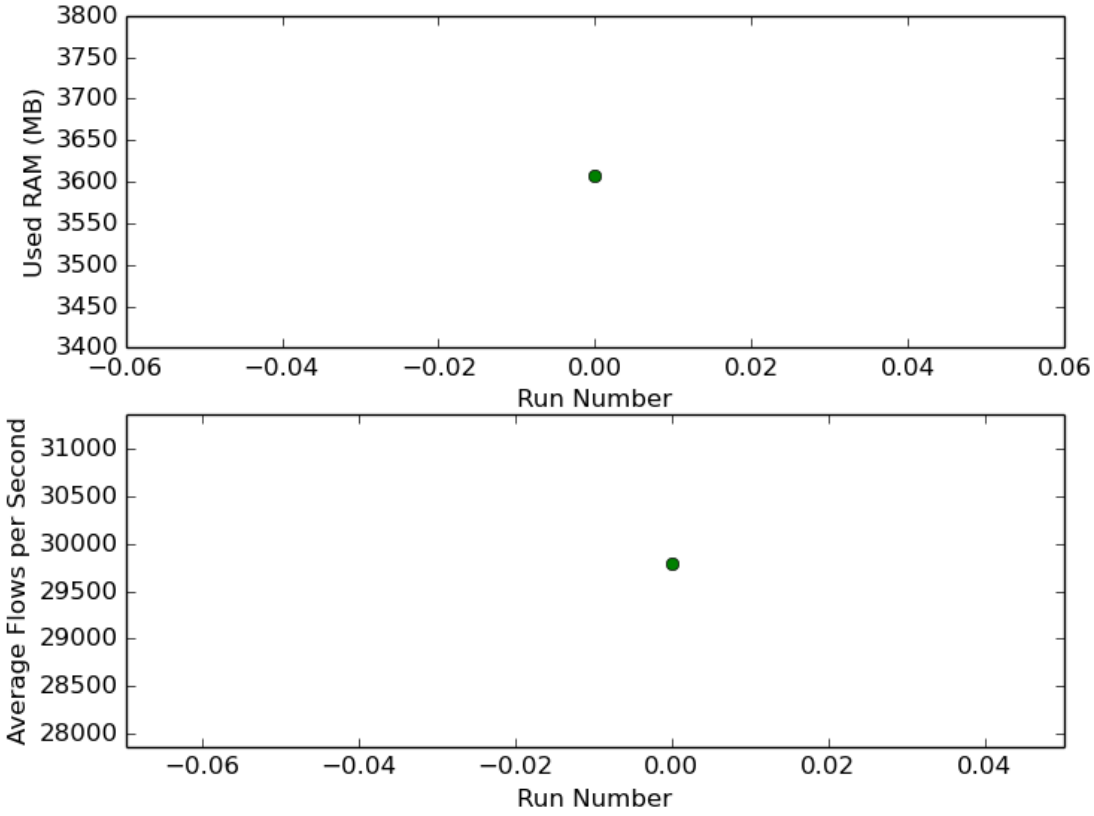


Figure 4.11: Stats from running wcbench once

To run wcbench repeatedly, *loop_wcbench.sh* script can be used.

| Issue: No support for Ubuntu: |
| --- |
| Solution:<br>Changed ./wcbench.sh to support Ubuntu 14.04.<br>Code snippet: |

```
# To find the OS
python -mplatform | grep Ubuntu
return=$?
if [[ $return == 0 ]]; then
    OS=ubuntu
else
    python -mplatform | grep fedora
    return=$?
    if [[ $return == 0 ]]; then
            OS=fedora
    else
        echo "OS is not supported"
    fi
fi
# Install required packages
    echo "Installing OpenDaylight dependencies"
    if [[ $OS == "ubuntu" ]]; then
        sudo apt-get install java-1.7.0-openjdk unzip wget
    elif [[ $OS == "fedora" ]]; then
        sudo yum install -y java-1.7.0-openjdk unzip wget
    fi
```

The full code can be accessed from [14]

Issue: Ctrl+c to kill loop_wcbench.sh

Solution:

*loop_wcbench.sh* script didnot have option to run the script for a given number of time. Once started, it had to be stopped through **ctrl+c**. The script was changed to accept an argument for the number of times it needs to be run. This change is now merged into its master branch at GitHub [14]. Code snippet:

```
loop_no_restart()
{
    num_runs=$1
    echo "Looping WCBench without restarting ODL"
    for (( runs_done = 0; runs_done < $num_runs; runs_done++ ));
    do
        echo "Starting run $(expr $runs_done + 1) of $num_runs"
        start_odl

        # Do this last so fn returns same exit code
        run_wcbench
    done
}
```

## 4.5 Comparing SDN with traditional network

Few simulators support OpenFlow protocol. NS3 is one of them. NS3 simulator source code and its installation steps can be found at [31] and [32] respectively. NS3 was installed through Bake. A network was created in simulator with a switch attached to 4 hosts. Leftmost host is to ping rightmost host and RTT to be calculated. The network connection is shown in figure 4.12 on page 65. RTT for both

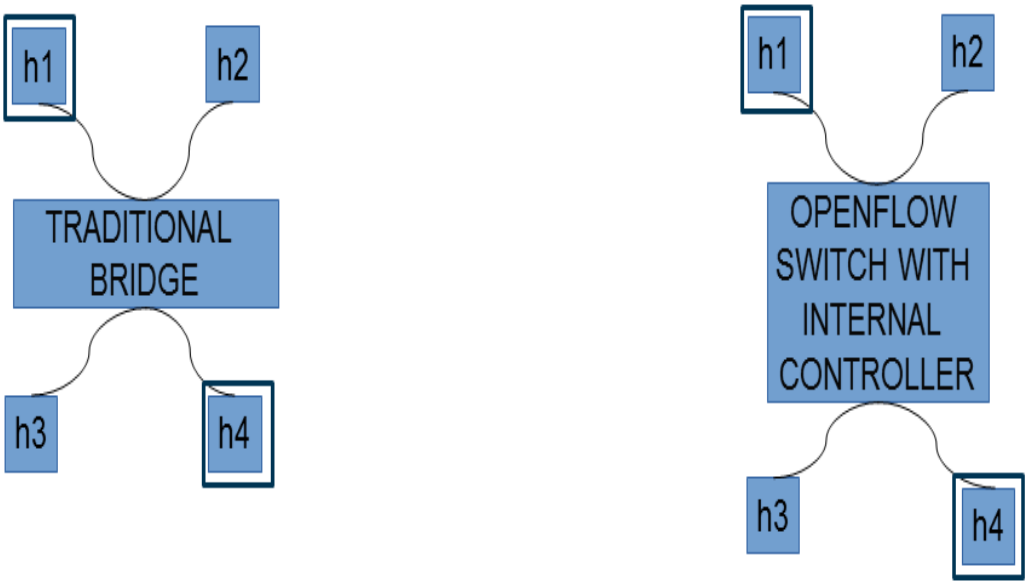


Figure 4.12: SDN and traditional network simulated in NS3

the networks was found to be same.

Issue: NS3 simlautor to compare the networks

1. bridge in traditional network cannot be compared with OVS in SDN.
2. no traditional switch module in NS3
3. NS3 uses an internal SDN controller and hence no account of time taken between OVS and controller.

This makes the comparison unfair though the RTT is same for both networks as discussed at NS3 Google Group [34].

# Chapter 5

# Installing OpenStack setup

3 node architecture OpenStack Icehouse setup was installed. OS used is Ubuntu Desktop 14.04. Physical configurations of 3 nodes is summarised in table 5.1 on page 66. RabbitMq is used as a messaging queue and MySQL as database. Initially

| | Controller | Neutron | Compute |
|---|---|---|---|
| RAM (in GB) | 4 | 4 | 4 |
| Storage (in GB) | 200 | 80 | 200 |
| CPU | Core i3 | Pentium | Core i3 |

Table 5.1: Physical configurations of nodes

nova network was setup and then Neutron. Pictorial representation of installation is depicted in figure 5.1 on page 67. Installation guides can be found at [] and []. Following the guides strictly doesnot give a working setup. Few changes were made to have a fully functional setup. Few issues are discussed here.

Issue: VM s not getting IP address

```
Solution:
dnsmasq service neds to be installed.
neutron$ sudo apt-get install dnsmasq
```

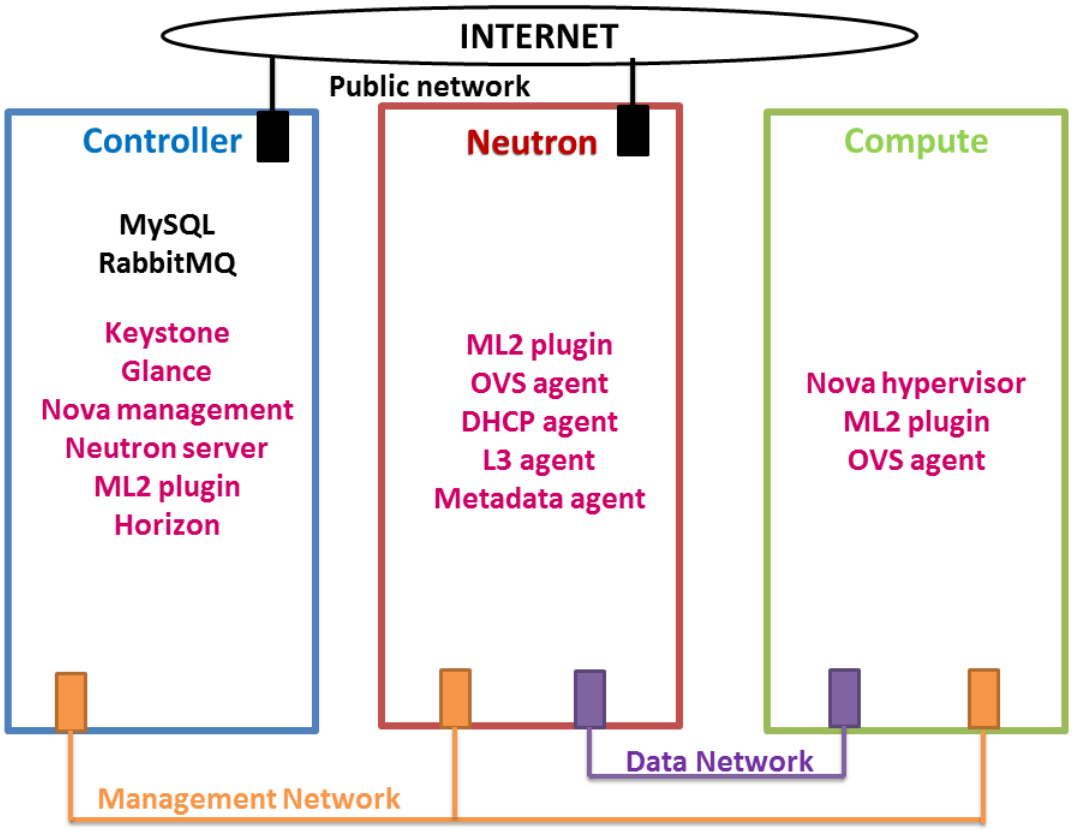


Figure 5.1: OpenStack installation setup

Installation is a cumbersome process which takes around 5 hours with continuous human attention required.

| Issue: Installation takes 5 hours |
| --- |
| Solution:<br>Automated the installation through Python script.<br>Code snippet: |

```
#Writing to /etc/hosts
def write():
    with open("/etc/hosts", "a") as hostfile:
        hostfile.write(neutron_internal_ip + "\t" + neutron_host
            + "\n")
        hostfile.write(compute_internal_ip + "\t" + compute_host
            + "\n")
        hostfile.write(controller_internal_ip + "\t" +
            controller_host + "\n")
```

## 5.1 Within the nodes

### 5.1.1 Creating a network

Whenever a network and a subnet are created, its namespace is created in Neutron. It could be validated with the command

```
compute$ sudo ip netns
compute$ qdhcp-network_id
```

The namespace 2 interfaces:

1. one loopback
2. DHCP interface having an IP address in the created network range. This interface serves DHCP requests. Let it be called *tap-xyz*.

Interface *tap-xyz* is attached to one of the ports of *bt-int*. *br-int* is an OVS integration bridge. This bridge is then connected to another bridge called tunnel bridge representated as *br-tun* through patch ports and veth pairs. *br-tun* is then attached to physical interface to data network. At this point of time, *br-int* has 2 flows with

drop and normal actions making it act like a simple L2 switch broadcasting all the messages as they come.

Code snippet:

```
neutron:~$ sudo ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=385.819s, table=0, n_packets=679,
    n_bytes=96106, idle_age=8, priority=1 actions=NORMAL
 cookie=0x0, duration=385.708s, table=22, n_packets=0,
    n_bytes=0, idle_age=385, priority=0 actions=drop
```

### 5.1.2 Launching an instance

This is done on Compute node. After creating a network, a VM is attached to it. VM gets attached to linux bridge. LInux bridge implements IP tables for security. Linux bridge is then connected to *br-int* which in turn to *br-tun* and then to physical interface to data network. GRE tunnel is created between Neutron and Compute node for data network packets. GRE tunnel attaches to *br-tun* on both nodes. It is the job *ovs agent* running on both nodes to ensure that the bridges and tunnel are created. VM's configuration details could be found in file located at $\backslash var \backslash lib \backslash nova \backslash instances \backslash instance\text{-}id$

#### 5.1.2.1 Creating a router

A router when created has its own namespace in Neutron node. Router is used to connect two networks. The router's namespace will have 3 interfaces:

1. loopback

2. attaching to 1 network

3. attaching to another network

Interfaces attaching to the networks have IPs. These IPs act as the gateway for the networks. These interfaces are connected to the *br-int* bridge on Neutron. The whole connection between Compute and Neutron is shown in figure 5.2 on page 70. The bridge configurations at the end is shown below.

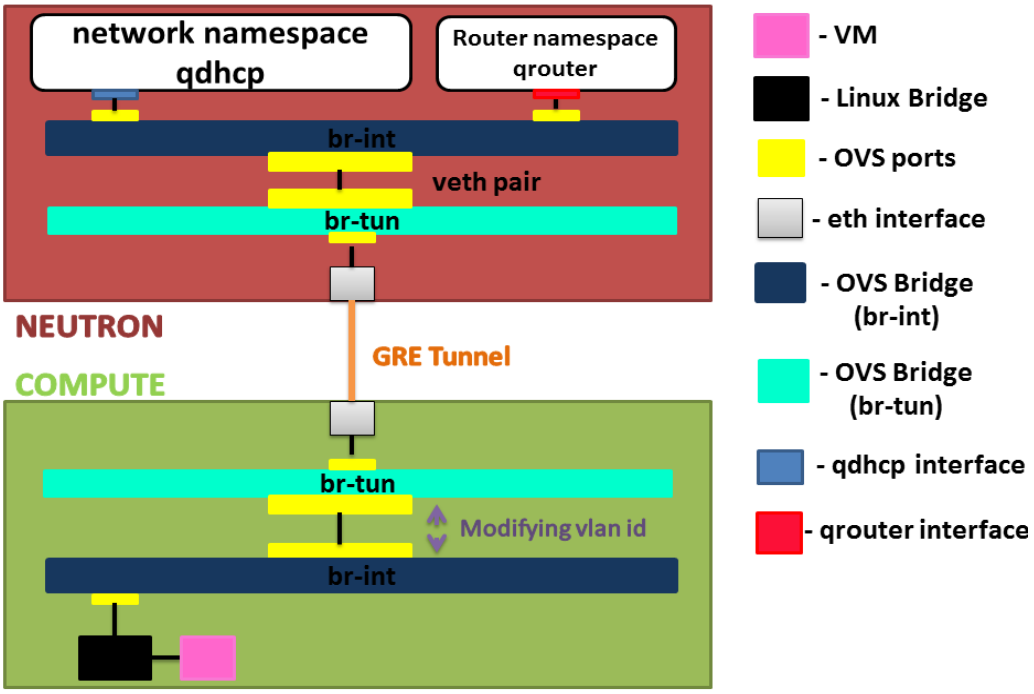


Figure 5.2: Within Compute and Neutron nodes

```
neutron$ sudo ovs-vsctl show
f9f9a961-bd7f-45e9-81e1-f7fbdecc48ca
    Bridge br-tun
        Port "gre-0a6b0086"
            Interface "gre-0a6b0086"
                type: gre
                options: {in_key=flow, local_ip="ip_of_neutron",
                         out_key=flow, remote_ip="ip_of_compute"}
        Port br-tun
            Interface br-tun
                type: internal
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
    Bridge br-int
        fail_mode: secure
        Port br-int
            Interface br-int
                type: internal
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
    ovs_version: "2.0.2"
```

```
compute$ sudo ovs-vsctl show
819c7929-e1b3-47e8-a979-d2e803ad0e8c
    Bridge br-tun
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
        Port "gre-0a6b0087"
            Interface "gre-0a6b0087"
                type: gre
                options: {in_key=flow, local_ip="ip_of_compute",
                         out_key=flow, remote_ip="ip_of_neutron"}
        Port br-tun
            Interface br-tun
                type: internal
    Bridge br-int
        fail_mode: secure
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
        Port br-int
            Interface br-int
                type: internal
    ovs_version: "2.0.2"
```

# Chapter 6

# Integrating ODL with OpenStack

OpenStack uses OVS for internode communciations. OVS are managed by ons agent running on both Neutron and Compute node. It does Layer 2 jobs. ODL has southbound plugin to manage OVS switches. This means that the OVS bridges of OpenStack can be managed by ODL as well. ODL can be used as SDN controller for OpenStack. ODL has *ovsdb northbound plugin* which takes REST calls from Neutron as shown by (1) in figure 6.1 on page 74. This plugin then directs *ovsdb southbound plugin* and *OpenFlow plugin* to configure the OVS switches in Neutron and Compute node as shown by (2) in same figure. When VM is created, a GRE tunnel is formed between both nodes as shown by (3) in same figure.

## 6.1 Integrating ODL with OpenStack

Steps to integrate them are as shown below: [34]

1. Install required features in ODL

```
opendaylight-user@root>feature:install odl-base-all\
odl-aaa-authn odl-restconf odl-nsf-all odl-adsal-northbound\
odl-mdsal-apidocs odl-ovsdb-openstack odl-ovsdb-northboun
odl-dlux-core
```

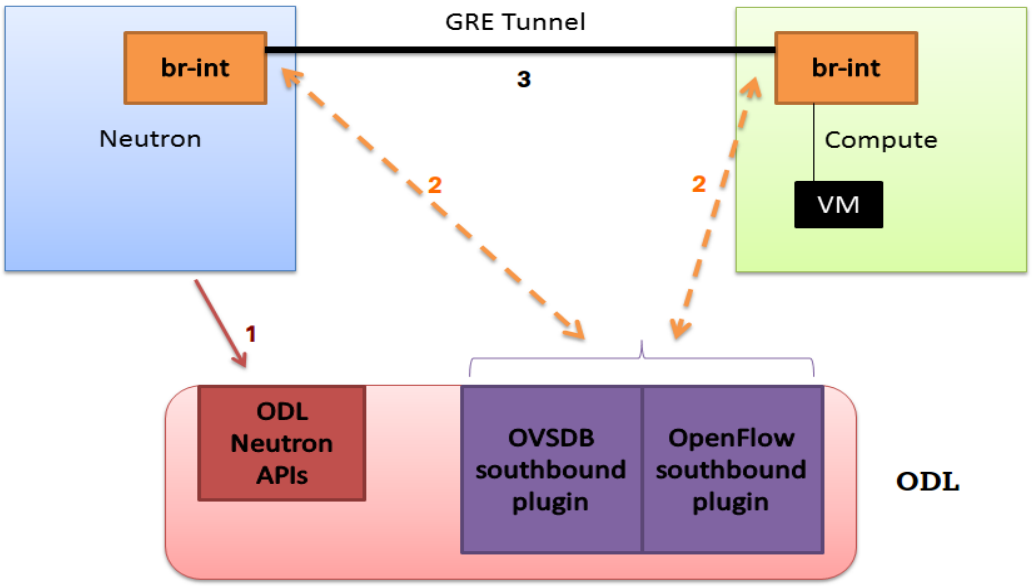


Figure 6.1: ODL integrated with OpenStack

2. Start with clean OpenStack slate with no networks or routers created.

3. Stop neutron server running on Controller node.

```
controller$ sudo service neutron-server stop
```

4. Stop OVS agent on both Neutron and Compute nodes.

```
sudo service neutron-plugin-ovs-agent stop
```

5. On both Neutron and Compute node, stop OVS switches and clear its configurations and then start it.

```
sudo service openvswitch-switch stop
sudo rm -rf /var/log/openvswitch/*
sudo rm -rf /etc/openvswitch.conf.db
sudo service openvswitch-switch start
```

OVS switch configurations on both Neutron and Compute node should look like this

```
sudo ovs-vsctl show
9f3b38cb-eefc-4bc7-828b-084b6yjolgffd
ovs_version: "2.0.2"
```

6. Set ODL to be OVS switch's manager.

```
sudo ovs-vsctl set-manager tcp:ip_of_controller:6640
```

OVS switch configurations on both Neutron and Compute node should look like this

```
sudo ovs-vsctl show
9f3b38cb-eefc-4bc7-828b-084b6yjolgffd
Manager "tcp:172.16.21.56:6640"
    is_connected: true
Bridge br-int
    Controller "ip_of_controller:6633"
    fail_mode: secure
    Port br-int
        Interface br-int
ovs_version: "2.0.2"
```

Note that ODL has created integration bridge *br-int*.

7. On all three nodes, Controller, Compute and Neutron, direct ML2 plugin of Neutron to ODL.

```
In /etc/neutron/plugins/ml2/ml2_conf.ini ensure following
fields are set as shown:
```

```
mechanism_drivers = openvswitch
[ml2_odl]
password = admin
username = admin
url = http://ip_of_controller:8080/controller/nb/v2/neutron
```

8. Reset Neutron database.

```
controller$ mysql -u root -p
mysql> DROP DATABASE neutron;
mysql> CREATE DATABASE neutron;
mysql> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'localhost'
       IDENTIFIED BY 'NEUTRON_DBPASS';
mysql> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'%'
       IDENTIFIED BY 'NEUTRON_DBPASS';
```

9. Verify that the integration is successful. If it is successful, curl to ODL's Neutron northbound plugin will return networks running on OpenStack.

```
curl -u admin:admin http://ip_of_controller:8080/controller/nb/
v2/neutron/networks


{
   "networks" : [ ]
}
Since there are no networks, it returns NULL.
```

The final bridge connections are as shown.

```
neutron$ sudo ovs-vsctl show
192a6b47-6993-4a70-9952-c38fd609d10c
    Manager "tcp:ip_of_controller:6640"
        is_connected: true
    Bridge br-int
        Controller "tcp:ip_of_controller:6633"
            is_connected: true
        fail_mode: secure
        Port "gre-ip_of_compute"
            Interface "gre-ip_of_compute"
                type: gre
                options: {key=flow, local_ip="ip_of_neutron",
                         remote_ip="ip_of_compute"}
        Port "tap-network1"
            Interface "tap-network1"
                type: internal
        Port br-int
            Interface br-int
    ovs_version: "2.0.2"

compute$ sudo ovs-vsctl show
7442b848-af5d-44ba-b28d-3e07a8e54326
    Manager "tcp:ip_of_controller:6640"
        is_connected: true
    Bridge br-int
        Controller "tcp:ip_of_controller:6633"
            is_connected: true
        fail_mode: secure
        Port "tap-VM1"
            Interface "tap-VM1"
```

```
        Port br-int
            Interface br-int
        Port "gre-ip_of_neutron"
            Interface "gre-ip_of_neutron"
                type: gre
                options: {key=flow, local_ip="ip_of_compute",
                         remote_ip="ip_of_neutron"}
    ovs_version: "2.0.2"
```

## 6.2 Advantages of using ODL with OpenStack

Few of the advantages of using ODL with OpenStack are

1. Only 1 bridge *br-int* is created. This makes debugging easy.
2. No need to run *neutron-plugin-openvswitch-agent* on Neutron and Compute nodes. Job of this plugin is done by ODL.
3. Whenever a device is to be added in Neutron, its driver is to be written in ML2 plugin of Neutron. ODL already has various southbound plugins so with ODL + OpenStack setup, no new drivers need to be written.
4. Underlay network can also be managed with ODL. This leads to better utilization of physical underlay network.
5. All ODL apps can be run with OpenStack.

## 6.3 Negative point of using ODL with OpenStack

*neutron-plugin-ovs-agent* is responsible for creating bridges and GRE tunnel. With ODL, ODL creates the bridges. GRE tunnel end points are to be explicitly given to ODL to create the tunnel.

Issue: VMs not getting IP:

```
Solution:
Define the IP to attach GRE tunnel end point to. On both Neutron
and Compute, find the ip of OVS and assign IP to it.
compute$ ovs-vsctl get Open_vSwitch . _uuid
    #This will return id of br-int
compute$ ovs-vsctl set Open_vSwitch <uuid-returned> \
         other_config:local_ip=ip1
```

# Chapter 7

# Conclusion

It can be concluded that SDN is a new hope into easing a network administrators work. Since there is a single controlling entity i.e the controller, evolution can be made faster. SDN also makes adding a new application in a network faster and easier. It addresses the limitations of the physical routers, switches and various network devices with increase in network traffic.

New versions of ODL controller are released and are to be released which signifies that it is going to be a next age of networking. Since they have a SNMP4SDN plugin, Ethernet switches are also going to be in market for quite a long time. With SDN, a dumb switch can be converted into a firewall. This tells that almost anything can be done in a network based on SDN concept. Support for IoT in Lithium release of ODL guarantees a long future for ODL and existence of SDN in every aspect of networking.

In days to come, traditional networks are expected to be replaced by SDNs. As SDN decouples control and data plane, adding new applications into a network becomes easier and hence new applications can also be expected to come up. Since SDN makes routing devices independent of vendors, any application that can be thought of could most probably be deployed. Vendors like Cisco and Brocade have come up with hteir own ODL based propertiary controller named *Cisco Extensible Network Controller (XNC)* and *Vyatta Controller* respectively. These are commer-

cial grade controllers.

ODL controller promises SDN to be a reality. It could be taken to a next level and combined with cloud. OpenStack is gaining popularity. Companies like PayPal are usng it. Using ODL in cloud platform like OpenStack brings all advantages of SDN to cloud. It encourages to make efficient use of physical underlay network as well as OpenStack overlay network. ODL takes over job of layer 2 agent (OVS agent) in OpenStack. ODL can also be configured to replace L3 agent of neutron and do all routing tasks. Only task Neutron will be left with is DHCP service.

# Bibliography

[1] G. Thomas, "Software Defined Networking (SDN) for the non-technical CXO" 27 October 2013 [Online]. Available: $http://cxounplugged.com/2013/10/software-defined-networking-sdn$ [Accessed 9 December 2014].

[2] K. Barker, "What is control plane and data plane," 14 August 2011. [Online]. Available: $https://learningnetwork.cisco.com/thread/33735$ [Accessed 9 December 2014].

[3] OpenDayLight Blog post by Neela Jacques submitted on April 1, 2015. [Online]. Available: $http://www.opendaylight.org/blogs/neela-jacques$ [Accessed 25 April 2015].

[4] OpenDayLight wiki page for "Simultaneous Release: Lithium Release Plan" last modified on 21 April 2015 and accessed on 23 April 2015. [Online]. Available: $https://wiki.opendaylight.org/view/Simultaneous\_Release:Lithium\_Release\_Plan$ [Accessed 25 April 2015].

[5] OpenHub web page accessed on 23 April 2015. [Online] Available: $https://www.openhub.net/p/opendaylight$ [Accessed 25 April 2015].

[6] OpenDayLight wiki page on "Project list" last modified on 12 March 2015. [Online]. Available: $https://wiki.opendaylight.org/view/Project\_list$ [Accessed 25 April 2015].

[7] OpenDayLight web page "OpenDaylight Helium – The Rise of Open SDN," 2014. [Online]. Available: $http://www.opendaylight.org/software$ [Accessed 25 April 2015].

[8] "SNMP Object Navigator," [Online]. Available: $http://tools.cisco.com/Support/SNMP/do/BrowseOID.do?local=en\&substep=2\&translate=Translate\&tree=NO$ [Accessed 25 April 2015].

[9] "SwitchDB.txt," [Online]. Available: $https://wiki.opendaylight.org/images/3/34/SwitchDB.txt$ [Accessed 25 April 2015].

[10] "File:SNMP4SDN TopologyDiscovery.jpg," 14 September 2013. [Online]. Available: $https://wiki.opendaylight.org/view/File:SNMP4SDN\_TopologyDiscovery.jpg$ [Accessed 25 April 2015].

[11] "SNMP4SDN:Installation Guide," 5 December 2014. [Online]. Available: $https://wiki.opendaylight.org/view/SNMP4SDN:Installation\_Guide$ [Accessed 25 April 2015].

[12] Conference paper on "On Controller Performance in Software-Defined Networks". Available:$https://www.usenix.org/system/files/conference/hot-ice12/hotice12-final33\_0.pdf$ [Accessed 25 April 2015].

[13] oflops GitHub repository of Andreas Wundsam Andreas Wundsam and last modified 5 months ago. Available: $https://github.com/andi-bigswitch/oflops/tree/master/cbench$ [Accessed 25 April 2015].

[14] WCBench GitHub repository of Daniel Farrell. Available: $https://github.com/dfarrell07/wcbench$ [Accessed 25 April 2015].

[15] OpenFlow Switch Specification Version 1.3.4 ( Protocol version 0x04 ), 27 March 2014. [White Paper] Available: $https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf$ [Accessed 9 December 2014].

[16] OVSDB manaual. Available: $http : //openvswitch.org/ovs - vswitchd.conf.db.5.pdf$ [Accessed 25 April 2015].

[17] "Mininet website," [Online]. Available: $http : //mininet.org/$ [Accessed 9 December 2014].

[18] "Introduction to Mininet," [Online]. Available: $https : //github.com/mininet/mininet/wiki/Introduction - to - Mininet$ [Accessed 9 December 2014].

[19] "Download/Get Started With Mininet," [Online]. Available: $http : //mininet.org/download/$ [Accessed 9 December 2014].

[20] Source code of Open vSwitch. Available: $http : //openvswitch.org/download/$ [Accessed 25 April 2015].

[21] CPqD GitHub repository authored by Eder Leo Fernandes 23 days ago. Available: $https : //github.com/CPqD/ofsoftswitch13$ [Accessed 25 April 2015].

[22] Blog on "Difference between private and floating IP. Available: $https : //www.rdoproject.org/Difference\_between\_Floating\_IP\_and\_private\_IP$ [Accessed 25 April 2015].

[23] Blog on "Effective Data Protection for Cloud Computing and its Relevance in the Nigeria Economy" posted on June 16, 2011. Available: $https : //toyinogunmefun.wordpress.com/2011/06/16/effective - data - protection - for - cloud - computing - and - its - relevance - in - the - nigeria - economy/$ [Accessed on 22nd April 2015].

[24] Blog on "An economic view on Cloud Computing." by Peter Hanselman, Posted on May 22, 2011. Available: $https : //peterhanselman.wordpress.com/2011/05/22/cloudnomics/$ [Accessed on 22nd April 2015].

[25] Blog on "How cloud based software solutions drive innovative performance management" by Srinivasan Sankar and Beenu Nishani Dewasurendra. Avail-

able: $http://www.cimaglobal.com/Our-locations/SriLanka/Thought-leadership-update/2013-managing-innovation/How-cloud-based-software-solutions-drive-innovative-performance-management/$ [Accessed on 22nd April, 2015].

[26] OpenStack Installation guide on Ubuntu. Avaliable: $http://docs.openstack.org/icehouse/install-guide/install/apt/content/ch_overview.html$ [Accessed on 22nd April 2015].

[27] Git Hub repository by Chaima Ghribi last updated 2 months ago. Available: $https://github.com/ChaimaGhribi/OpenStack-Icehouse-Installation/blob/master/OpenStack-Icehouse-Installation.rst$ [Accessed on 23 April 2015].

[28] OpenDayLight download page. Available: $http://www.opendaylight.org/software/downloads$. [Accessed 25 April 2015].

[29] Question about adding meter to OVS at ODL ask-answer forum. Available: $https://ask.opendaylight.org/question/2461/problem-adding-meter-postman/$. [Accessed 25 April 2015].

[30] Question about adding meter to OVS at ODL ask-answer forum. Available: $https://ask.opendaylight.org/question/2473/ovs-231-openflow-13-meter-feature/$. [Accessed 25 April 2015].

[31] Source code of NS3. Available: $https://www.nsnam.org/release/ns-allinone-3.22.tar.bz2$. [Accessed 25 April 2015].

[32] Installtion steps of NS3. Available: $https://www.nsnam.org/wiki/Installation\#Building_ns-3_with_build.py$. [Accessed 25 April 2015].

[33] NS3 Google Group discussion dated February 2 2015. Available: $https://groups.google.com/forum/\#!topic/ns-3-users/4PEPXvZ32dI$

[34] OpenDayLight wiki on "OpenStack and OpenDayLight" last modified on 22 January 2015. Available: *https* : *//wiki.opendaylight.org /view/OpenStack_and_OpenDaylight*. [Accessed on 23 April 2015].