

“CAPSTONE PROJECT REPORT”

MIGRATION OF LEGACY NETWORK
INFRASTRUCTURE
INTO
SDN ENABLED NETWORK

Submitted by: Madhurima Kumar

Submission: April 2015

Project Supervisor: Mr. Muhammad Durrani

Master of Science in Internetworking (MINT)



ACKNOWLEDGEMENT

I would like to express my profound gratitude and deep regards to my supervisor Mr. Muhammad Durrani for his exemplary guidance, monitoring and constant encouragement throughout the project. I would also like to sincerely thank Prof. Mike MacGregor, MINT Director and Mr. Shahnawaz Mir, MINT Coordinator for their immense support and for providing me with the unique opportunity to do this industry related project.

ABSTRACT

This project demonstrates an emerging next-generation alternative for cloud service providers and enterprise data centers by allowing the network to be manipulated by software (OpenFlow, OVSD, SDN), making it flexible to customize the network to local needs, eliminating undesirable features, cutting down the costs significantly and opening it up for future innovation. The aim of this project is to showcase transparent migration of legacy service provider network infrastructure into SDN enabled network.

The project report includes the contrast between legacy and SDN networks depicting the need for SDN in today's world. It shows the lab setup designed for this project, describing every SDN component and protocol required for migration, followed by the actual implementation and results along with concluding with the beneficial changes brought about with SDN. Additionally, a different way of bringing in new features and changes in the implementation of this scenario, making it even better have been discussed in the future scope section.

TABLE OF CONTENT

Abstract.....	3
List of Figures.....	5
CHAPTER-1 Introduction.....	6
1.1. Legacy Networks	6
1.1.1. Problems with Legacy Networks.....	6
1.2. Software Defined Networking (SDN)	6
1.2.1. Benefits of SDN over Legacy Networks.....	7
1.2.2. SDN Applications.....	7
CHAPTER-2 SDN Lab Setup.....	8
2.1. Network Topology Description	8
2.1.1. Hardware Used	8
2.1.2. Virtualization Option.....	9
2.1.3. Main Components	9
2.1.3.1. Customer Edge Routers (CE).....	9
2.1.3.2. Core Provider Routers (P).....	9
2.1.3.3. Provider Edge Routers (PE)	9
2.1.3.4. Open vSwitch (OVS)	9
2.1.3.5. OpenDaylight Controller.....	11
2.1.3.6. Northbound Interface	11
2.1.3.6.1 OSGi.....	11
2.1.3.6.2 Rest API	12
2.1.3.7. Southbound Interface	12
2.1.3.7.1 OVSDB	12
2.1.3.7.2 OpenFlow (OF)	13
2.1.3.8. Underlay Network	13
2.1.3.9. Overlay Network - VXLAN Tunnel.....	13
2.1.4. Network Topology Working.....	14
CHAPTER-3 Main Implementation	15
CHAPTER-4 Result.....	25
CHAPTER-5 Conclusion	27
CHAPTER-6 Future Scope.....	28
References	29
Appendix-I Router Configurations	31

LIST OF FIGURES

Figure-1: Lab Setup	8
Figure-2: Components of Open vSwitch (OVS).....	10
Figure-3: Life of a new flow in OVS.....	11
Figure-4: OVSDDB Protocol.....	12
Figure-5: SDN- OpenFlow Protocol.....	13
Figure-6: OVSDDB Plugin in ODL	15
Figure-7: ODL connection with PE1	16
Figure-8: Automatically created bridges: br-int, br-tun on PE1	16
Figure-9: ODL connection with PE1 and PE2.....	16
Figure-10: Automatically created bridges: br-int, br-tun on PE1	17
Figure-11: Adding ports and tunnel to both switches	17
Figure-12: Configuration on PE1 through OVSDDB Plugin	18
Figure-13: OVS ports mapping to OpenFlow ports on PE1 bridge br-int.....	19
Figure-14: OVS ports mapping to OpenFlow ports on PE1 bridge br-tun	19
Figure-15: Configuration on PE2 through OVSDDB Plugin	20
Figure-16: OVS ports mapping to OpenFlow ports on PE2 bridge br-int.....	21
Figure-17: OVS ports mapping to OpenFlow ports on PE2 bridge br-tun	21
Figure-18: Topology learned by OpenDaylight, Connected Nodes shown	22
Figure-19: Example of a flow installed in Open vSwitch PE1 by the controller	23
Figure-20: Example of a flow installed in Open vSwitch PE2 by the controller	24
Figure-21: Routing table of P1 router.....	25
Figure-22: Routing table of P2 router.....	25
Figure-23: Successful ping Result from Customer CE1 to CE11.....	26
Figure-24: Successful ping Result from Customer CE11 to CE1.....	26
Figure-25: Show configuration of P1	31
Figure-26: Show configuration of P2	32
Figure-27: Show configuration of CE1.....	33
Figure-28: Routing table of CE1.....	33
Figure-29: Show configuration of CE11.....	34
Figure-30: Routing table of CE11.....	34

CHAPTER 1 – INTRODUCTION

1.1 LEGACY NETWORKS

Legacy Networks refer to the networks based on old concepts, platforms and protocols that can be superseded by far more flexible, efficient and agile networks. Legacy networks initially came into existence as a response to the short-term requirements and goals for many organizations. They were not designed to meet the increasing demands and challenges, hence does not offer improved planning, management and integration.

1.1.1 PROBLEMS WITH LEGACY NETWORKS:

Any organization's that is today dependent on legacy networks and systems face many challenges as they rely on older protocols and solutions that do not cater to the evolving requirements and rapidly growing needs. Few of the problems with legacy networks are mentioned below:

- **Network Integration:** Integrating out-dated networks and systems with newer upcoming ones is a big challenge in itself as the newer networks and systems typically are based on completely different technology.
- **Higher Costs:** Maintaining older and newer networks can result in higher costs and additional expenses for any organization.
- **Time Consuming:** Maintaining legacy networks can be a challenging task as it demands more time and effort at the same time it requires highly skilled professionals having knowledge of both older and newer technologies.
- **Network Expansion:** Legacy networks rely on passé protocols and solutions that make it harder for any organization to grow and expand in today's world.
- **Vendor Dependence:** This leads to limiting the capability of network operators to adapt the network to their individual environments.

1.2 SOFTWARE DEFINED NETWORKING (SDN)

SDN was developed to meet the peaks and demands of rapid reconfigurations for the customers by making changes and managing the networks. SDN focuses on building network that is open and programmable. It allows handling of many different services in a dynamic fashion by providing a way to consolidate multiple services onto a common infrastructure.

It aims at building a computer network by separating it into three systems: Data Plane, Control Plane and Management Plane.

- **Data Plane:** It is responsible for sending and receiving data. It also aims at packet processing and forwarding to the destination.
- **Control Plane:** It is responsible for making forwarding decisions and programming the data plane. Control plane aims at managing the configurations and network topology. It provides performance and fault management capability.
- **Management Plane:** It aims at overall management of all connections. Management plane is essentially responsible for configuration, monitoring and troubleshooting of the networks.

SDN is a modern way of managing networks that virtually separates network control from the actual packet processing and forwarding, thus making the control plane more flexible. It provides the capability to rapidly expand, modify or manage the network architectures using automated tools.

1.2.1 BENEFITS OF SDN OVER LEGACY NETWORKS:

- Scalability of Network Resources
- Flexible Networks
- Cost Effective, Cheaper hardware
- Central Control
- Secure Networks
- Faster On-demand Provisioning
- Guaranteed Data Delivery

In traditional networks, the network should be aware of the applications running, however, with SDN, the applications can be aware of the network,

Depending on an organization needs and requirements, SDN allows development and installation of the appropriate applications to cater to a specific type of network behavior that is being demanded. Common networking functions like routing, switching, virtualization; security, QoS etc. can all be covered under these applications.

1.2.2 SDN APPLICATIONS

There are various domains where SDN can be implemented:

- Data Centers
- Enterprise networks
- Wide-area backbone networks
- Internet exchange points
- Home networks

CHAPTER 2 – SDN LAB SETUP

This section describes the SDN lab setup to showcase the migration of legacy service provider network into SDN based network, depicting a general industrial solution.

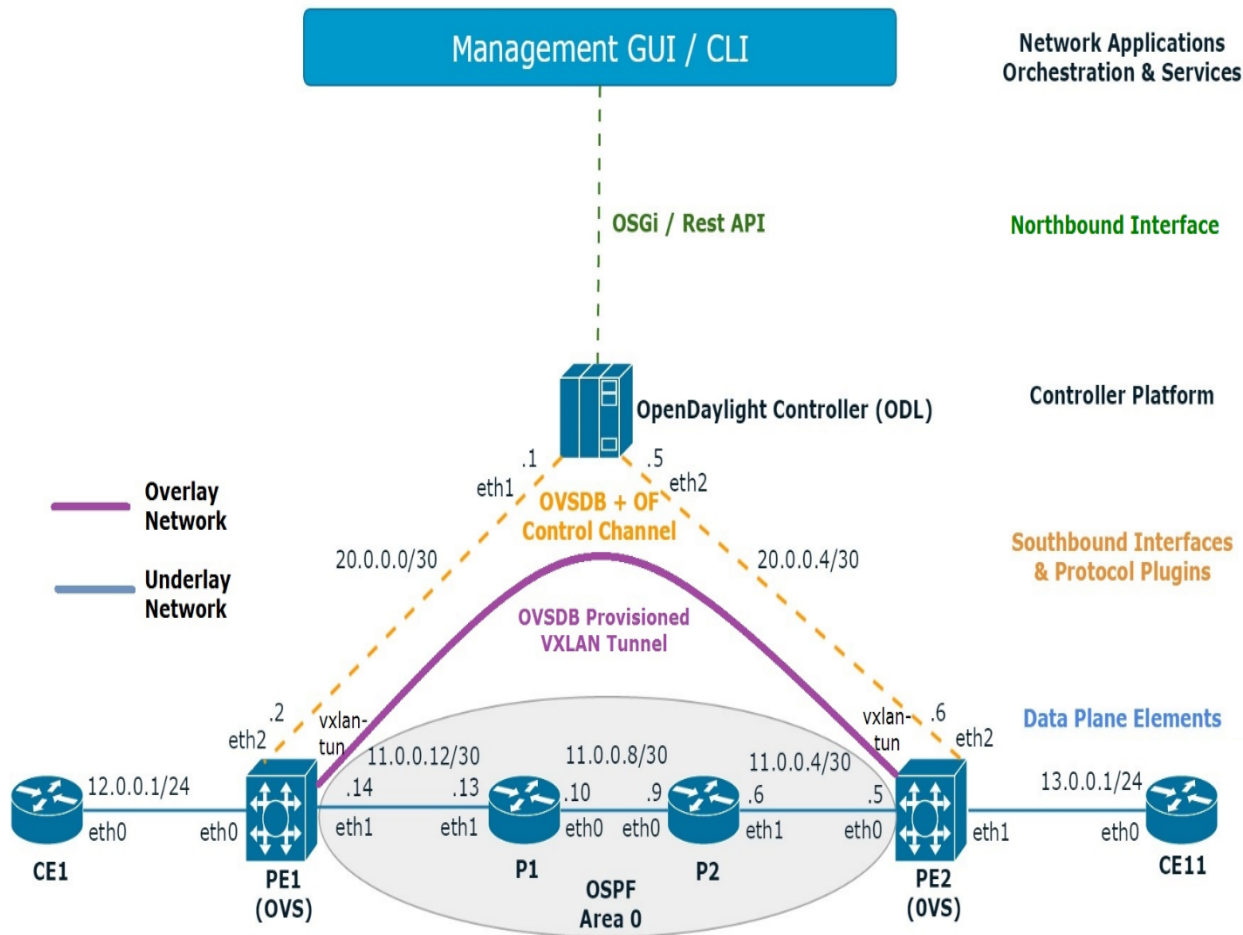


Figure-1: Lab Setup

2.1 NETWORK TOPOLOGY DESCRIPTION

This section described the hardware used, virtualization option, main components and the working of the above SDN lab setup.

2.1.1 HARDWARE USED

Dell PowerEdge R420 Server

Two of Dell PowerEdge R420 Servers were used to setup this SDN network. This server belongs to Intel® Xeon® processor E5-2400 product family and has features like 4 core, cache 2.5MB per core, 16GB DDR3 with 1U rack form factor.

2.1.2 VIRTUALIZATION OPTION

VMware ESXi 5.5

The virtualization option chosen for this setup is VMware ESXi 5.5, which is an operating system-independent, bare-metal hypervisor, deployed directly on top of the physical Dell R420 servers, partitioning it into multiple virtual machines required for this setup.

2.1.3 MAIN COMPONENTS

2.1.3.1 Customer Edge Routers (CE)

Both the Customer Edge routers (CE1 and CE11) are Brocade Vyatta 5400 vRouter, version 6.7 (64-bit), which is a high performance virtual router. Customer Edge routers are directly connected to Provider Edge routers, which in this topology are Open vSwitch instances. These CE routers are provisioned such that they send a packet out of their Ethernet interface eth0. They do not have any routes of the core IP backbone network in their routing table. CE simply sends a packet out of its eth0 interface, which is caught by OVS-PE instance and then sent over the overlay VXLAN tunnel which is on top of an underlying OSPF network to reach the desired destination customer.

2.1.3.2 Core Provider Routers (P)

Both the Core Provider routers (P1 and P2) are Brocade Vyatta 5400 vRouter, version 6.7 (64-bit), which is a high performance virtual router. OSPF routing protocol has been configured in Area 0 which consists of these core provider routers. This is used to form the underlying network so as to provide IP connectivity between the tunnel end points, which are in different subnets, deployed on both OVS-PE instances so that the tunnel can successfully be created over this underlay network. Both P1 and P2 do not have any route of the Customer Edge routers in their routing table and are totally unaware of the customer packets flowing through the tunnel created over them. They are just to provide IP reachability for the tunnel, with no knowledge of the tenant IP addressing in use.

2.1.3.3 Provider Edge Routers (PE)

In this scenario, both the Provider Edge PE1 and PE2 are Open vSwitch (OVS) instances. The open source Open vSwitch version 2.3.0 was installed on Ubuntu 12.04 (Precise Pangolin). A VXLAN tunnel is configured between PE1 and PE2 over the IP core OSPF network. Each Provider Edge OVS instance has two automatically created bridges: br-int and br-tun, because of the connection to the manager. Both the bridges are connected to each other via patch ports. VXLAN tunnel end points are created on bridge br-tun. The OVS instances support OpenFlow protocol and in this setup OF1.0 is used.

2.1.3.4 Open vSwitch (OVS)

Open vSwitch is an OpenFlow capable virtual switch which is used to connect the virtual machines. It is usually used to avoid complications in virtual networks and multi-tenant environments. Some of the features that OVS supports are:

- VLAN tagging and 802.1q trunking

- Spanning Tree Protocol 802.1D
- LACP
- Port mirroring
- Flow export (eg: sflow, netflow, ipfix)
- Tunneling (eg: GRE, VXLAN, IPSEC)
- QoS control
- OpenFlow Protocol support

Main Components of OVS

The main components of Open vSwitch are:

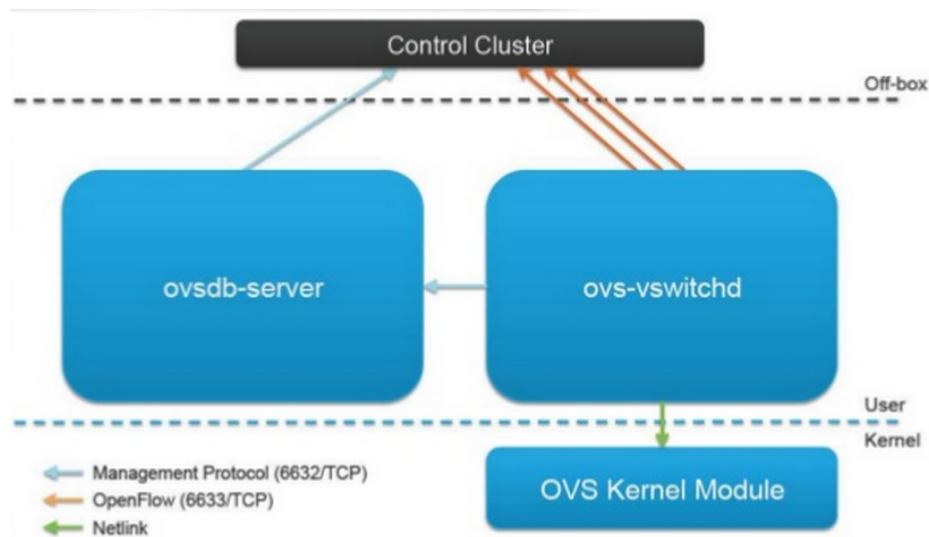


Figure-2: Components of Open vSwitch (OVS)

Ovs-vswitchd

This is a core component of OVS and runs in User space. Ovs-vswitchd communicates with the database server via OVSDb protocol (explained later), with the kernel module via Netlink, with the controller via the OpenFlow protocol (explained later) and with system via the netdev abstract interface. It is responsible for pushing and managing the flows in the kernel module.

OVS Kernel Module

The OVS kernel module is a cache of the recent traffic. If a packet arriving to a virtual switch has a cached match in the kernel module, then those cached actions will be taken to process that particular packet. This has no knowledge about OpenFlow, but talks with the ovs-vswitchd through Netlink.

Ovsdb-server

The database server consists of the switch level configuration; the changes in which remains persistent and can survive a reboot. It communicates with the manager and ovs-vswitchd via OVSDb protocol.

Life of a new flow in OVS

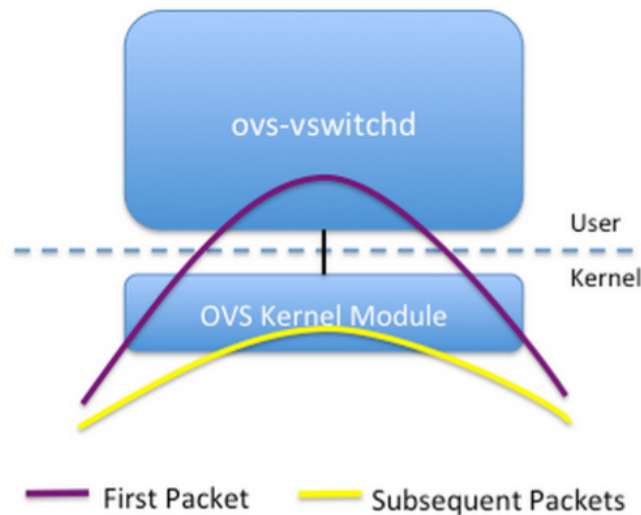


Figure-3: Life of a new flow in OVS

Whenever a packet arrives in OVS, the flow cache in kernel module is checked for that packet. If a relevant entry for that packet is found then those cached actions will be taken to process the packet. But if there is no entry for that packet then the packet of the new flow goes to ovs-vswitchd which decides the appropriate action for that packet through OpenFlow protocol and installs this entry into the flow cache so that in future a similar packet goes directly through the kernel space.

2.1.3.5 OpenDaylight Controller

In this scenario, OpenDaylight Hydrogen – Virtualization Edition is installed on Ubuntu 14.04. This edition supports both OF1.0 and OF1.3

The virtualization edition is meant for data centre environments, built on the base edition including additional functionalities like support for creating and managing Virtual Tenant Network (VTN), virtual overlays etc.

The controller communicates with the applications and the data plane elements through its intrafaces as described below:

2.1.3.6 Northbound Interface

The controller communicates with applications or higher layer control programs through its northbound interface. OpenDaylight controller supports the following for northbound API:

2.1.3.6.1 OSGi

OpenDaylight controller uses the OSGi framework for the northbound API used by the upper applications. OSGi framework is meant for application, which runs in the same address space as the controller. In this scenario, we use OSGi CLI to send management commands via OpenDaylight OVSDb plugin.

2.1.3.6.2 Rest API

OpenDaylight controller supports bidirectional Rest (web based) for the northbound API, which is used by the applications to get hold of the controller and access the network. Rest API is meant for applications that do not run in the same address space or even in the same machine as the controller.

2.1.3.7 Southbound Interface

The southbound interfaces enable communication between the OpenDaylight controller and the data plane elements (virtual switches & physical device interfaces). The two southbound interfaces implemented in this scenario are:

2.1.3.7.1 OVSDB

OVSDB stands for Open vSwitch Database Management Protocol. This protocol is used to configure the Open vSwitch instances. OVSDB has two ways to be connected to Open vSwitch agent, which is decided by the OVS configuration itself:

1. Passive Mode- Manager “ptcp:6640”

In passive management mode, controller initiates a connection to the OVS element that is listening on a specified port.

2. Active Mode- Manager “tcp:x.x.x.x:6640”

In active mode, the OVS element is set to attach to an active Daylight management controller.

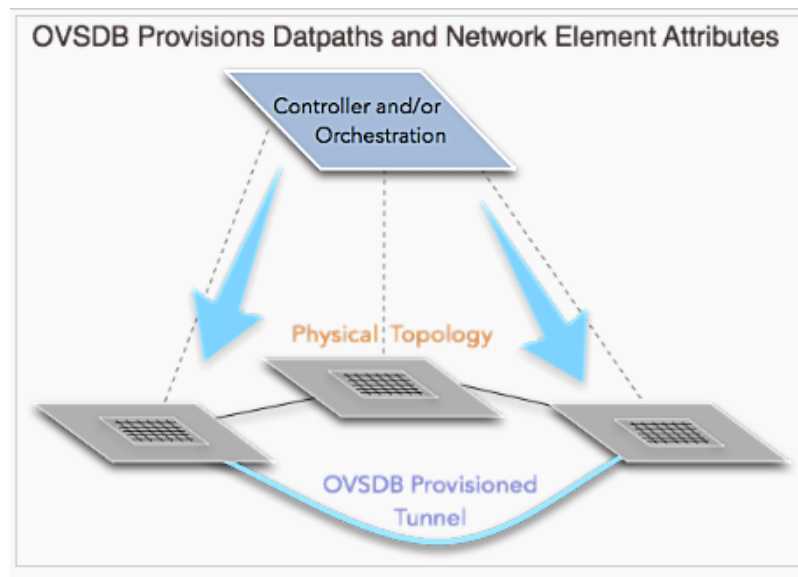


Figure-4: OVSDB Protocol

The ovsdb-server in the switch consists of the switch level configuration, the change in which remains persistent and can survive a reboot. It communicates with the manager and ovs-vswitchd via OVSDB protocol.

After the OVSDB plugin is setup, the controller can configure the bridges, ports, tunnels in Open vSwitch. The OVSDB manager programs the ovsdb-server process, which implements the configuration in OVS.

2.1.3.7.2 OpenFlow (OF)

In this scenario, we are using OpenFlow 1.0 both in the Open vSwitch and the OpenDaylight controller. OpenFlow protocol enables the interaction between the control and forwarding layers of SDN architecture.

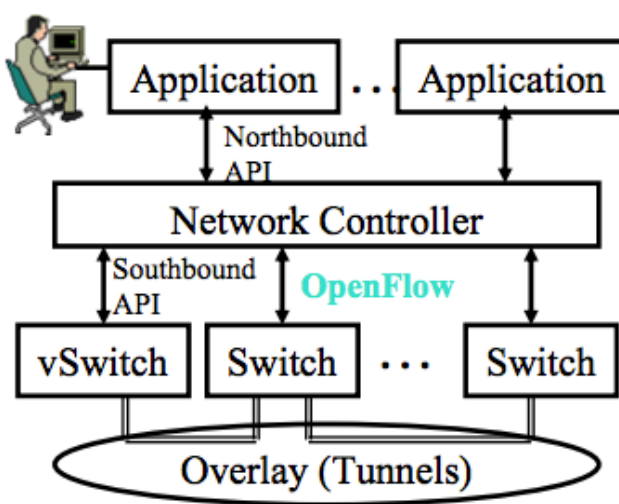


Figure-5: SDN- OpenFlow Protocol

It allows the controller to inform the Open vSwitches where to forward the packet and appropriate action to be taken while forwarding it. With the help of OpenFlow protocol, the control plane is centralized as all packet forwarding decisions are sent from there and the network can be programmed without any dependency in individual network elements.

2.1.3.8 Underlay Network

Underlay network is the network infrastructure over which the overlay network is created. In this scenario, the IP core backbone network running OSPF routing protocol in Area 0 is the underlay network. So, the core provider routers are not aware of the customer IP addressing or the customer packets flowing through the overlay network created on top of this underlay network.

2.1.3.9 Overlay Network- VXLAN Tunnel

The overlay network makes it appear to the virtual machines as if there is direct L2 connection between them because of the IP connectivity provided by the underlying network. The messages are passed between the virtual machines through the tunnel without them knowing anything about the tunnelling mechanism and network infrastructure in between over which the tunnel is

created. Also, the underlying network has no information about the IP addressing of the packets, which are flowing through the tunnel created over them. This underlying network is there to provide only IP connectivity between the tunnel end points.

Need of having overlay networks (VXLAN tunnels)

Various issues in case of multi tenancy are:

- Tenants require to have logical separation so that they have flexibility in terms of IP addressing and various tenants can use the same IP address
- Security requirement so that no other tenant can see the private traffic
- Scalability issues- 802.1q VLAN's have a 12-bit VLAN ID and partition Ethernet network into 4096 broadcast domains, which is not sufficient as far as the requirement is concerned because of virtualization and cloud deployment.
- Large forwarding tables due to many virtual machines

All these issues can be handled with the help of overlay networks creating tunnels (VXLAN, GRE, etc.). In this scenario we are deploying VXLAN tunnels. The only requirement for having tunnels is that there should be IP connectivity for the tunnels with the help of underlay network. Virtual eXtensible LAN (VXLAN) segment has 24-bit VXLAN network identifier (VNI), which means it supports over 16 million VXLAN identifiers. This means that now every tenant will have its own isolated virtual network identified by the VXLAN Network Identifier (VNI).

2.1.4 NETWORK TOPOLOGY WORKING

The Customer edge routers CE1 and CE11 attached to the Provider Edge PE1 and PE2 Open vSwitch instances send the packets out of their eth0 interface towards the OVS instances. The switches check the flow cache in its own kernel module for the packet. If it doesn't find any cached flow, the switch being dumb sends the packet over to the OpenDaylight controller through the OpenFlow protocol. In this scenario, the network application hands over the control to the OpenDaylight controller through the OSGi northbound interface. Then, this controller takes care of programming the switch level configuration, i.e., all the OVS bridges, ports, tunnels using the OVSDB plugin and finally uses OpenFlow protocol to configure the flows on br-int and br-tun of both OVS instances. So when the switches receive the flow entry from the OpenDaylight controller, they simply follow the actions stated to forward the packet through the overlay network of VXLAN tunnel created by the controller between the two OVS instances. Through this VXLAN tunnel, each tenant is logically separated to provide security so that no other tenant can see the private traffic and also gets its own isolated virtual network identified by the VXLAN Network Identifier (VNI). There is IP connectivity between the vxlan tunnel end points provided by the underlying IP backbone OSPF network. So when the packet travels across the tunnel and reaches the other tunnel end point, who reads off the VNI of the tenant to identify the tenant virtual network and strips off all the outer headers leaving the original packet for the destined customer. In this forwarding to the destined customer, the switch again checks the flow added by the controller to find out the appropriate OpenFlow output port and the actions to be taken. The destination customer has no information about the VXLAN network encapsulation that was done to forward the packet over to this. This customer can send its reply back to the source customer router by the same path and by following the actions specified in the flow table created on the PE Open vSwitches by the OpenDaylight controller.

CHAPTER 3 – MAIN IMPLEMENTATION

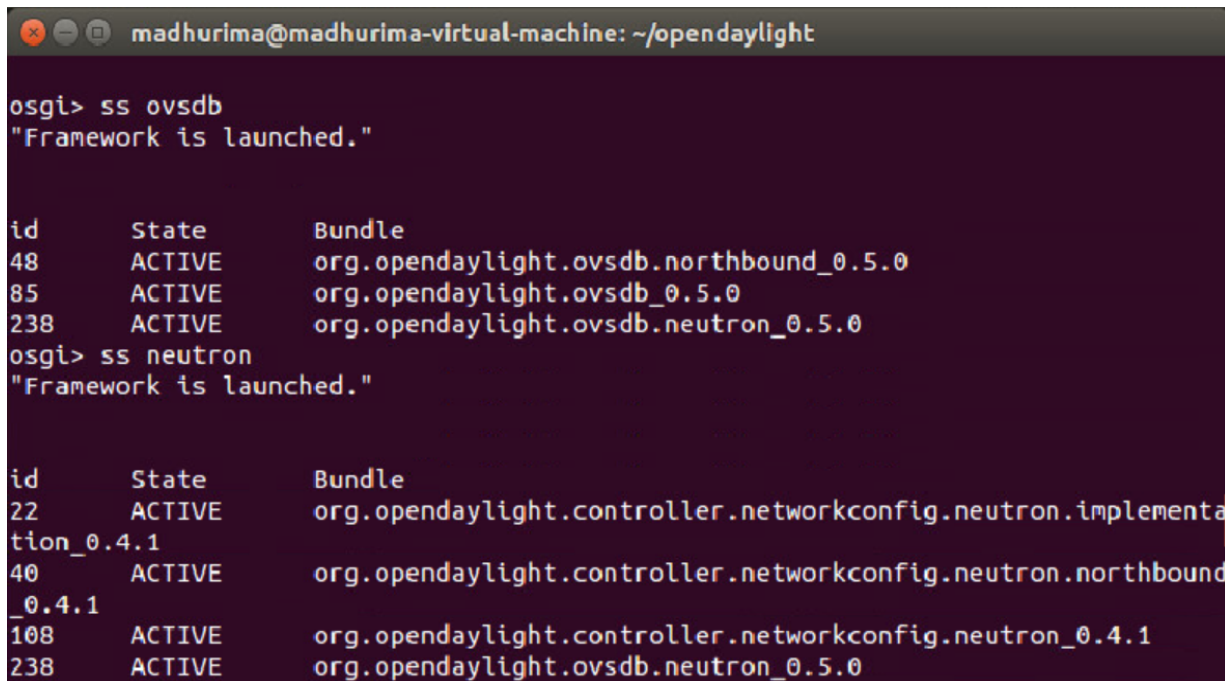
The OpenDaylight controller uses OVSDb mechanism to manage the Open vSwitch database by having the ability to view, create, modify and delete OVS objects like bridges, ports and interfaces.

OpenDaylight supports the following Northbound API, which can be used by the applications to perform some basic management plane functions:

1. Bidirectional REST-API
2. OSGi Framework

In this project I have shown this implementation using Internal OSGi commands to provision the management plane via OpenDaylight's OVSDb plugin.

First, I started the controller with OVSDb plugin. It can be checked if this plugin is up and running or not by the following command.



```
madhurima@madhurima-virtual-machine: ~/opendaylight
osgi> ss ovsdb
"Framework is launched."

id      State      Bundle
48      ACTIVE     org.opendaylight.ovsdb.northbound_0.5.0
85      ACTIVE     org.opendaylight.ovsdb_0.5.0
238     ACTIVE     org.opendaylight.ovsdb.neutron_0.5.0
osgi> ss neutron
"Framework is launched."

id      State      Bundle
22      ACTIVE     org.opendaylight.controller.networkconfig.neutron.implementa
tion_0.4.1
40      ACTIVE     org.opendaylight.controller.networkconfig.neutron.northbound
_0.4.1
108     ACTIVE     org.opendaylight.controller.networkconfig.neutron_0.4.1
238     ACTIVE     org.opendaylight.ovsdb.neutron_0.5.0
```

Figure-6: OVSDb Plugin in ODL

So it shows that OVSDb plugin is working and is active. Now, the OVSDb has two ways to be connected to Open vSwitch agent:

3. Passive Mode- Manager “ptcp:6640”
4. Active Mode- Manager “tcp:x.x.x.x:6640”

This is decided by the OVS configuration itself. In passive management mode, controller initiates a connection to the OVS element that is listening on a specified port. However, in active mode, the OVS element is set to attach to an active Daylight management controller.

I used active management mode by the following command on both OVS (PE1 and PE2).

On PE1:

```
sudo ovs-vsctl set-manager tcp:20.0.0.1:6640
```

On the OpenDaylight controller it can be seen that the OVSDDB and OpenFlow connection between the controller and Openvswitch has been made:

```
osgi> printnodes
Nodes connected to this controller :
[OVS|20.0.0.2:58579, OF|00:00:00:0c:29:a8:95:4c, OF|00:00:b2:81:f5:dd:8a:40]
```

Figure-7: ODL connection with PE1

Here, OVS node is the management node and OF nodes are the data path nodes showing the mac address in use by the two automatically created bridges: br-int and br-tun, on Open vSwitch PE1. The IP address of PE1 is 20.0.0.2.

```
osgi> getBridgeDomains "OVS|20.0.0.2:58579"
Existing Bridges: [br-tun, br-int]
```

Figure-8: Automatically created bridges: br-int, br-tun on PE1

This shows the name of the automatically created bridges on the newly connected Open vSwitch instance PE1.

So, Node OF|00:00:00:0c:29:a8:95:4c is br-int of PE1

Node OF|00:00:b2:81:f5:dd:8a:40 is br-tun of PE1

Similarly on PE2:

```
sudo ovs-vsctl set-manager tcp:20.0.0.5:6640
```

So, on controller it shows all the connections to both switches PE1 and PE2 along with the bridges that were created automatically in the switches:

```
osgi> printnodes
Nodes connected to this controller :
[OF|00:00:00:0c:29:29:59:ca, OVS|20.0.0.6:47719, OVS|20.0.0.2:58579, OF|00:00:00:0c:29:a8:95:4c, OF|00:00:72:51:14:8f:4b:48, OF|00:00:b2:81:f5:dd:8a:40]
osgi>
```

Figure-9: ODL connection with PE1 and PE2

In total there are two OVS management nodes and four OF data path nodes in this topology. One OVS instance is associated with two OF data paths. The newly added OVS instance is PE2 with IP address of 20.0.0.6.


```
osgi>
osgi> getBridgeDomains "OVS|20.0.0.6:47719"
Existing Bridges: [br-tun, br-int]
```

Figure-10: Automatically created bridges: br-int, br-tun on PE1

This shows the name of the automatically created bridges on the newly connected Open vSwitch instance PE2.

So, Node OF|00:00:00:0c:29:29:59:ca is br-int of PE2

Node OF|00:00:72:51:14:8f:4b:48 is br-tun of PE2

Now, we add the eth ports and tunnel ports to both the bridges of both switches using OSGi framework. The commands used here to configure ports on the Open vSwitch bridges from OSGi framework are as follows-

To add a port to a bridge:

```
addPort <Node> <BridgeName> <PortName> <type> <options pairs>
```

This command simply creates on a bridge a new port with the specified port name from the network device of the same name.

To add a vlan tag to a port:

```
addPortVlan <Node> <BridgeName> <PortName> <vlan>
```

This command would make this port an access port for the specified vlan tag.

To add a tunnel:

```
addTunnel <Node> <Bridge> <Port> <tunnel-type> <remote-ip>
```

This command adds a tunnel port with the port name as specified, to remote IP address as specified, to the bridge specified. The tunnel-type specifies the type of tunnel to be created. Here, the type is VXLAN tunnel.

```
osgi> addPortVlan "OVS|20.0.0.2:58579" br-int eth0 1
Port creation status : Success: Success (0)
osgi>
osgi> addTunnel "OVS|20.0.0.2:58579" br-tun vxlan-tun vxlan 11.0.0.5
Port creation status : Success: Success (0)
osgi>
osgi> addPortVlan "OVS|20.0.0.6:47719" br-int eth1 1
Port creation status : Success: Success (0)
osgi>
osgi> addTunnel "OVS|20.0.0.6:47719" br-tun vxlan-tun vxlan 11.0.0.14
Port creation status : Success: Success (0)
```

Figure-11: Adding ports and tunnel to both switches

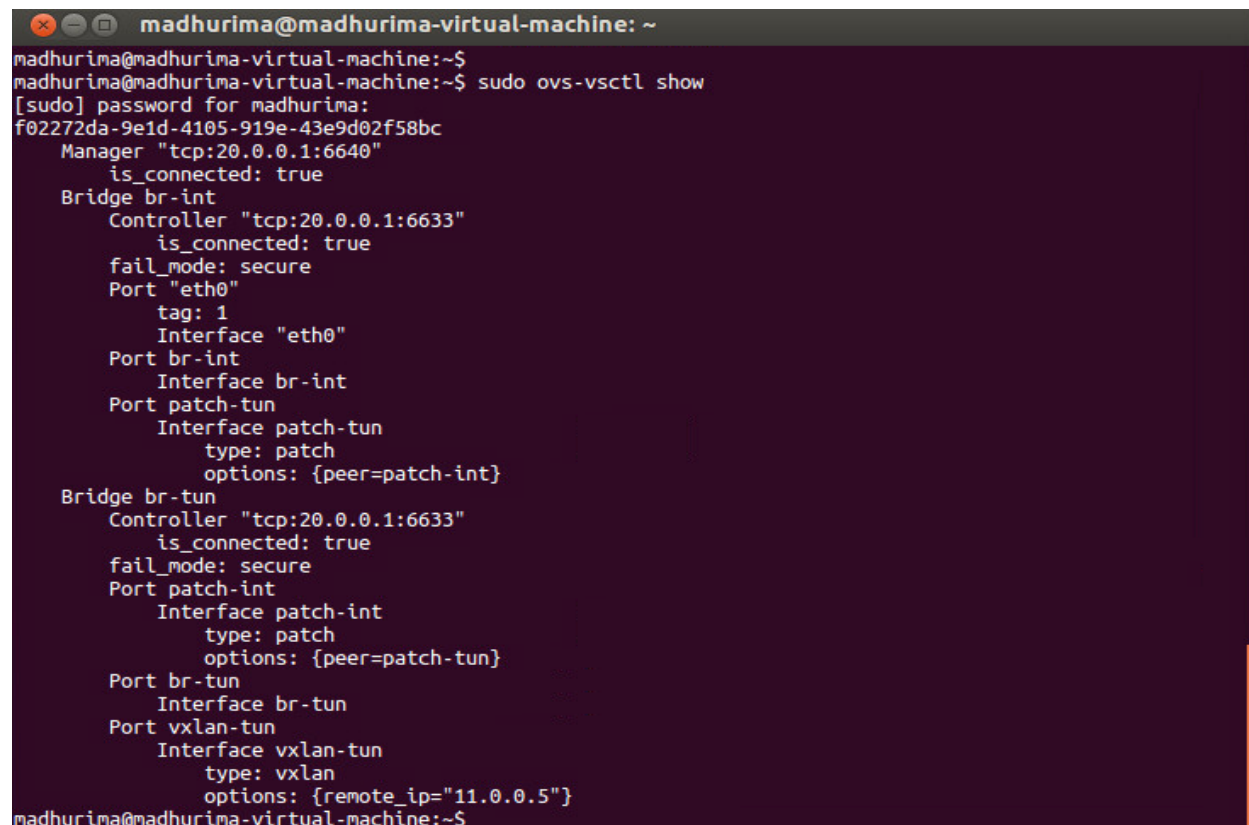
So, through the commands shown in the above figure, an Ethernet port eth0 is tagged with vlan id=1, and is created on bridge br-int of PE1 Open vswitch. Note that OVS|20.0.0.2:58579 belong to PE1.

A vxlan tunnel with tunnel end point named vxlan-tun is created on bridge br-tun of PE1 Open vswitch, having remote IP address as 11.0.0.5, which is the IP address of the other side of tunnel.

An Ethernet port eth1 is tagged with vlan id=1, and is created on bridge br-int of PE2 Open vswitch. Note that OVS|20.0.0.6:47719 belong to PE2.

A vxlan tunnel with tunnel end point named vxlan-tun is created on bridge br-tun of PE2 Open vswitch, having remote IP address as 11.0.0.14, which is the IP address of the other side of tunnel.

Checking the result of these commands configured on Open vSwitch PE1 through OVSDB Plugin of OpenDaylight Controller:



```
madhurima@madhurima-virtual-machine: ~
madhurima@madhurima-virtual-machine:~$ sudo ovs-vsctl show
[sudo] password for madhurima:
f02272da-9e1d-4105-919e-43e9d02f58bc
  Manager "tcp:20.0.0.1:6640"
    is_connected: true
  Bridge br-int
    Controller "tcp:20.0.0.1:6633"
      is_connected: true
    fail_mode: secure
    Port "eth0"
      tag: 1
      Interface "eth0"
    Port br-int
      Interface br-int
    Port patch-tun
      Interface patch-tun
        type: patch
        options: {peer=patch-int}
  Bridge br-tun
    Controller "tcp:20.0.0.1:6633"
      is_connected: true
    fail_mode: secure
    Port patch-int
      Interface patch-int
        type: patch
        options: {peer=patch-tun}
    Port br-tun
      Interface br-tun
    Port vxlan-tun
      Interface vxlan-tun
        type: vxlan
        options: {remote_ip="11.0.0.5"}
madhurima@madhurima-virtual-machine:~$
```

Figure-12: Configuration on PE1 through OVSDB Plugin

From the above figure, it can be seen that two automatically created bridges br-int and br-tun are also automatically connected to the controller at tcp:20.0.0.1:6633.

There are two patch ports also created automatically, so as to connect the two bridges together. These are patch-tun in bridge br-int and patch-int in bridge br-tun. It can be seen that in the options field, the name of the peer patch port is specified to have the connection up and running.

It also shows the port eth0 and tunnel end point vxlan-tun with remote IP of 11.0.0.5, on switch PE1 that we created through the OVSDb plugin by OSGi commands. This shows that this end of the tunnel is successfully created.

Checking the mapping of OVS ports to OpenFlow Ports on PE1 bridge br-int:

```
madhurima@madhurima-virtual-machine:~$ sudo ovs-ofctl show br-int
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000c29a8954c
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
1(patch-tun): addr:ea:13:51:03:9f:4c
    config: 0
    state: 0
    speed: 0 Mbps now, 0 Mbps max
3(eth0): addr:00:0c:29:a8:95:4c
    config: 0
    state: 0
    current: 10GB-FD COPPER
    advertised: COPPER
    supported: 1GB-FD 10GB-FD COPPER
    speed: 10000 Mbps now, 10000 Mbps max
LOCAL(br-int): addr:00:0c:29:a8:95:4c
    config: PORT_DOWN
    state: LINK_DOWN
    speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

Figure-13: OVS ports mapping to OpenFlow ports on PE1 bridge br-int

So, in bridge br-int, port eth0 is mapped to OF port 3 and port patch-tun (which is used to connect the two bridges) is automatically assigned OF port 1. The MAC address associated with these ports can also be seen.

Checking the mapping of OVS ports to OpenFlow Ports on PE1 bridge br-tun:

```
madhurima@madhurima-virtual-machine:~$ sudo ovs-ofctl show br-tun
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000b281f5dd8a40
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
1(patch-int): addr:e6:22:a9:71:bd:eb
    config: 0
    state: 0
    speed: 0 Mbps now, 0 Mbps max
3(vxlan-tun): addr:ca:e8:9d:0b:88:82
    config: 0
    state: 0
    speed: 0 Mbps now, 0 Mbps max
LOCAL(br-tun): addr:b2:81:f5:dd:8a:40
    config: PORT_DOWN
    state: LINK_DOWN
    speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

Figure-14: OVS ports mapping to OpenFlow ports on PE1 bridge br-tun

So, in bridge br-tun, port vxlan-tun, which is the vxlan tunnel end point, is mapped to OF port 3 and port patch-int (which is used to connect the two bridges) is automatically assigned OF port 1. The MAC address associated with these ports can also be seen.

Similarly, checking the result of these configurations on Open vSwitch PE2 through OVSDB Plugin of OpenDaylight Controller:

```
madhurima@madhurima-virtual-machine:~$  
madhurima@madhurima-virtual-machine:~$ sudo ovs-vsctl show  
9870128f-4ad5-4073-8628-fac90b455190  
  Manager "tcp:20.0.0.5:6640"  
    is_connected: true  
  Bridge br-tun  
    Controller "tcp:20.0.0.5:6633"  
      is_connected: true  
    fail_mode: secure  
  Port vxlan-tun  
    Interface vxlan-tun  
      type: vxlan  
      options: {remote_ip="11.0.0.14"}  
  Port br-tun  
    Interface br-tun  
  Port patch-int  
    Interface patch-int  
      type: patch  
      options: {peer=patch-tun}  
  Bridge br-int  
    Controller "tcp:20.0.0.5:6633"  
      is_connected: true  
    fail_mode: secure  
  Port br-int  
    Interface br-int  
  Port "eth1"  
    tag: 1  
    Interface "eth1"  
  Port patch-tun  
    Interface patch-tun  
      type: patch  
      options: {peer=patch-int}
```

Figure-15: Configuration on PE2 through OVSDB Plugin

From the above figure, it can be seen that two automatically created bridges br-int and br-tun are also automatically connected to the controller at tcp:20.0.0.5:6633.

There are two patch ports also created automatically, so as to connect the two bridges together. These are patch-tun in bridge br-int and patch-int in bridge br-tun. It can be seen that in the options field, the name of the peer patch port is specified to have the connection up and running.

It also shows the port eth1 and tunnel end point vxlan-tun with remote IP of 11.0.0.14, on switch PE2 that we created through the OVSDB plugin by OSGi commands. This shows that this end of the tunnel is also successfully created.

So, the whole tunnel is up and successfully created and ready for use.

Checking the mapping of OVS ports to OpenFlow Ports on PE2 bridge br-int:

```
madhurima@madhurima-virtual-machine:~$ sudo ovs-ofctl show br-int
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000c292959ca
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
1(patch-tun): addr:4e:31:56:12:51:f5
    config: 0
    state: 0
    speed: 0 Mbps now, 0 Mbps max
2(eth1): addr:00:0c:29:29:59:ca
    config: 0
    state: 0
    current: 10GB-FD COPPER
    advertised: COPPER
    supported: 1GB-FD 10GB-FD COPPER
    speed: 10000 Mbps now, 10000 Mbps max
LOCAL(br-int): addr:00:0c:29:29:59:ca
    config: PORT_DOWN
    state: LINK_DOWN
    speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

Figure-16: OVS ports mapping to OpenFlow ports on PE2 bridge br-int

So, in bridge br-int, port eth1 is mapped to OF port 2 and port patch-tun (which is used to connect the two bridges) is automatically assigned OF port 1.

Checking the mapping of OVS ports to OpenFlow Ports on PE2 bridge br-tun:

```
madhurima@madhurima-virtual-machine:~$ sudo ovs-ofctl show br-tun
OFPT_FEATURES_REPLY (xid=0x2): dpid:00007251148f4b48
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
1(patch-int): addr:ba:87:c2:7b:01:b6
    config: 0
    state: 0
    speed: 0 Mbps now, 0 Mbps max
2(vxlan-tun): addr:7a:c2:16:31:16:2c
    config: 0
    state: 0
    speed: 0 Mbps now, 0 Mbps max
LOCAL(br-tun): addr:72:51:14:8f:4b:48
    config: PORT_DOWN
    state: LINK_DOWN
    speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

Figure-17: OVS ports mapping to OpenFlow ports on PE2 bridge br-tun

So, in bridge br-tun, port vxlan-tun, which is the vxlan tunnel end point, is mapped to OF port 2 and port patch-int (which is used to connect the two bridges) is automatically assigned OF port 1.

Since OpenDaylight is multi-protocol capable, it supports OVSDB as well as OpenFlow.

Open vSwitch uses OpenFlow for forwarding message in the control plane for both virtual and physical ports. ODL adds flows to OVS through OpenFlow protocol.

The topology learned by OpenDaylight controller is depicted below:

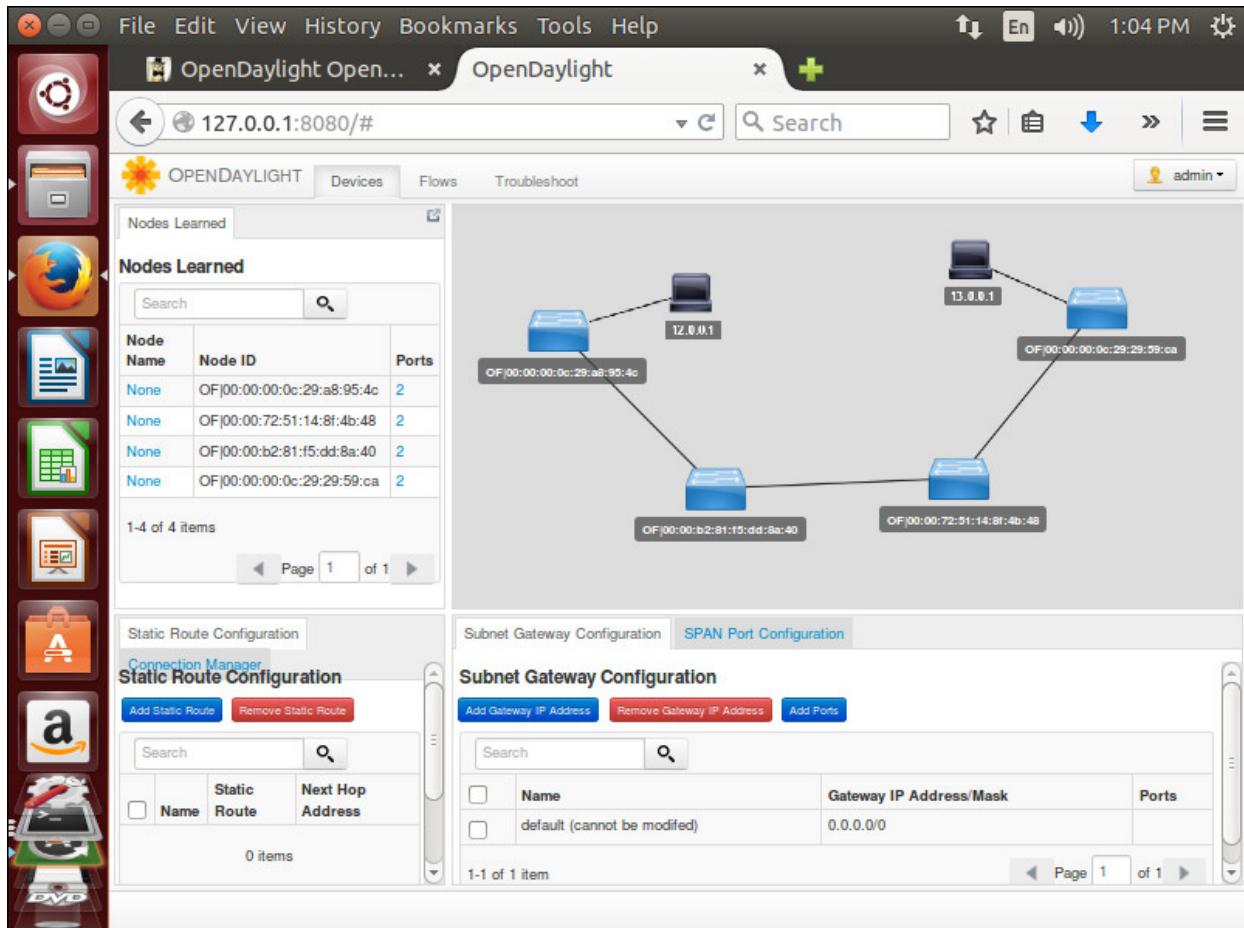


Figure-18: Topology learned by OpenDaylight, Connected Nodes shown

This shows the nodes learnt by the OpenDaylight Controller and also the hosts attached to the nodes. So here there are two OVS instances (PE1 and PE2), each instance having two bridges (br-int and br-tun) which are connected to each other by automatically created patch ports (patch-tun and patch-int respectively), and the hosts connected to br-int of each instance.

So the sequence in which they are shown in the network topology learnt by ODL are:

CE1 → br-int (PE1) → br-tun (PE1)====br-tun (PE2) → br-int (PE2) → CE11

The pipe shown here between the two br-tun represents the vxlan tunnel. The Node ID displayed on the left panel is actually the mac address used by every bridge. These OF nodes are the data path nodes of the topology.

In the next figure, we can see the flows added by OpenDaylight into the dumb switches PE1 and PE2 flow table. Since here OpenFlow1.0 is used, so there is just table=0. If OpenFlow1.3 was used in the network, then we could have multiple tables for flows.

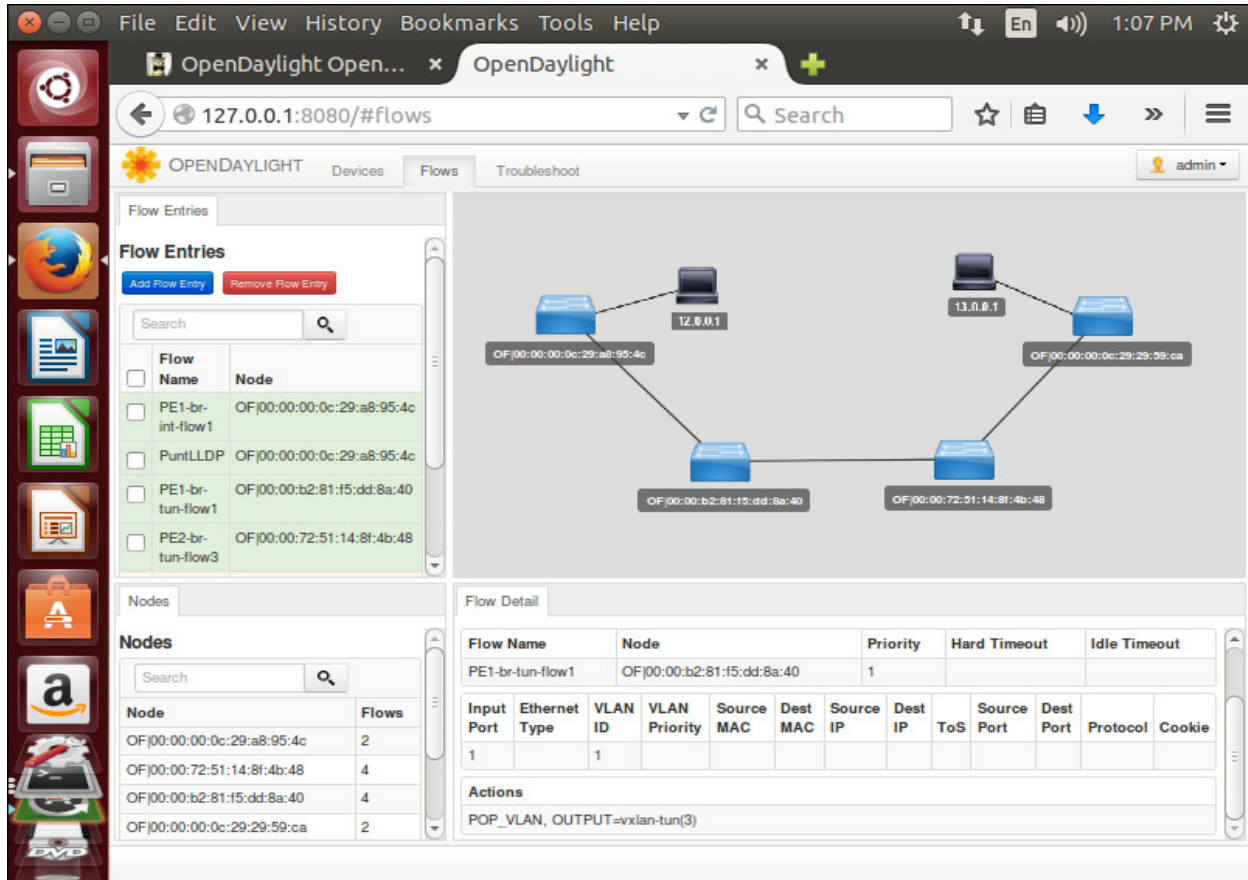


Figure-19: Example of a flow installed in Open vSwitch PE1 by the controller

In the above figure, the flow shown is of bridge br-tun of switch PE1.

It states that any packet incoming at input OF port=1 (which is patch-int of br-tun), with vlan id=1 should be forwarded to Output OF port=3 (which is vxlan tunnel end point vxlan-tun) after stripping off the vland id=1 from the packet. The vxlan tunnel end points have IP network connectivity because of the underlying OSPF network of Core P1, P2 routers.

The next figure shows another flow entry added to the Open vSwitch in this topology:

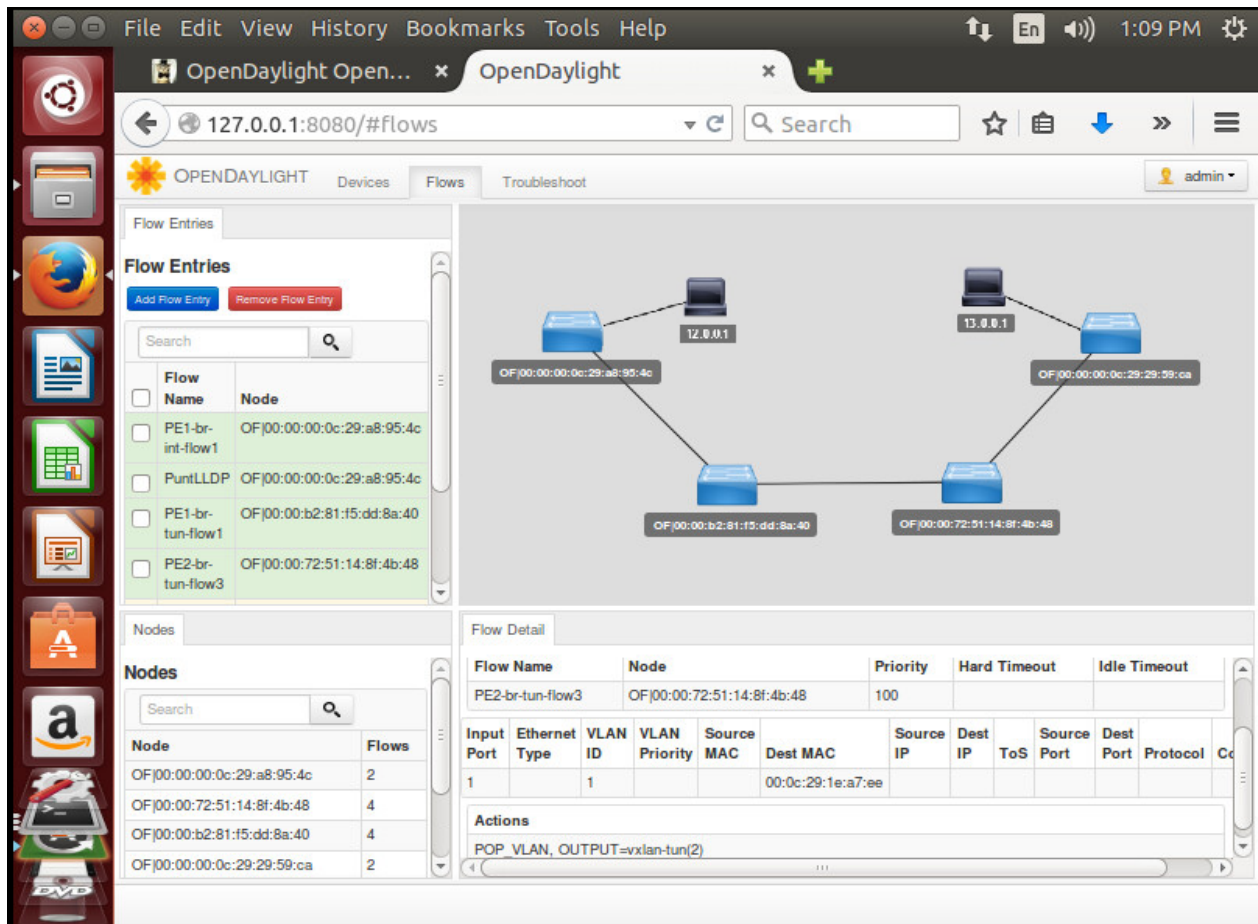


Figure-20: Example of a flow installed in Open vSwitch PE2 by the controller

In the above figure, the flow shown is of bridge br-tun of switch PE2. It states that any packet incoming at input OF port=1 (which is patch-int of br-tun), with vlan id=1 for destination Mac address of 00:0c:29:1e:a7:ee, should be forwarded to Output OF port=2 (which is vxlan tunnel end point vxlan-tun) after stripping off the internal vland id=1 and slabbing on from the packet.

CHAPTER 4 – RESULT

So, in this scenario, the network application hands over the control to the OpenDaylight controller through the OSGi northbound interface. Then, this controller takes care of programming the switch level configuration, i.e., all the OVS bridges, ports, tunnels using the OVSDb plugin and finally uses OpenFlow protocol to configure the flows on br-int and br-tun of both OVS instances.

The figure below shows the routing table of the IP core router P1 by the command – show ip route. It can be seen that the core router does not have any routes of the Customer Edge routers CE1 (whose IP address is 12.0.0.1/24) and CE11 (whose IP address is 13.0.0.1/24)

```
vyatta@vyatta:~$ show ip route
Codes: K - kernel, C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       > - selected route, * - FIB route, p - stale info

C    *> 2.2.2.2/32 is directly connected, lo
O    *> 3.3.3.3/32 [110/11] via 11.0.0.9, eth0, 05w0d12h
O    *> 11.0.0.4/30 [110/2] via 11.0.0.9, eth0, 05w0d12h
O    11.0.0.8/30 [110/1] is directly connected, eth0, 05w0d12h
C    *> 11.0.0.8/30 is directly connected, eth0
O    11.0.0.12/30 [110/1] is directly connected, eth1, 05w0d12h
C    *> 11.0.0.12/30 is directly connected, eth1
C    *> 127.0.0.0/8 is directly connected, lo
vyatta@vyatta:~$
```

Figure-21: Routing table of P1 router

The figure below shows the routing table of the IP core router P2 by the command –show ip route. It can be seen that this core router does not have any routes of the Customer Edge routers CE1 (whose IP address is 12.0.0.1/24) and CE11 (whose IP address is 13.0.0.1/24)

```
vyatta@vyatta:~$ show ip route
Codes: K - kernel, C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       > - selected route, * - FIB route, p - stale info

O    *> 2.2.2.2/32 [110/11] via 11.0.0.10, eth0, 05w0d12h
C    *> 3.3.3.3/32 is directly connected, lo
O    11.0.0.4/30 [110/1] is directly connected, eth1, 05w0d12h
C    *> 11.0.0.4/30 is directly connected, eth1
O    11.0.0.8/30 [110/1] is directly connected, eth0, 05w0d12h
C    *> 11.0.0.8/30 is directly connected, eth0
O    *> 11.0.0.12/30 [110/2] via 11.0.0.10, eth0, 05w0d12h
C    *> 127.0.0.0/8 is directly connected, lo
vyatta@vyatta:~$
```

Figure-22: Routing table of P2 router

So, both P1 and P2 have routes of Area 0 OSPF network including each other's loopback which also lies in OSPF. Because of this IP connectivity the tunnel end points are connected to each other.

Now let's try to ping from CE1 to CE11

```
vyatta@CE1# run ping 13.0.0.1
PING 13.0.0.1 (13.0.0.1) 56(84) bytes of data.
64 bytes from 13.0.0.1: icmp_req=1 ttl=64 time=0.349 ms
64 bytes from 13.0.0.1: icmp_req=2 ttl=64 time=0.386 ms
64 bytes from 13.0.0.1: icmp_req=3 ttl=64 time=0.416 ms
64 bytes from 13.0.0.1: icmp_req=4 ttl=64 time=0.338 ms
64 bytes from 13.0.0.1: icmp_req=5 ttl=64 time=0.381 ms
64 bytes from 13.0.0.1: icmp_req=6 ttl=64 time=0.362 ms
64 bytes from 13.0.0.1: icmp_req=7 ttl=64 time=0.342 ms
^C
--- 13.0.0.1 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 5994ms
rtt min/avg/max/mdev = 0.338/0.367/0.416/0.034 ms
[edit]
vyatta@CE1# _
```

Figure-23: Successful ping Result from Customer CE1 to CE11

Now, obviously it would ping backwards from CE11 to CE1 as well as shown below.

```
vyatta@CE11# run ping 12.0.0.1
PING 12.0.0.1 (12.0.0.1) 56(84) bytes of data.
64 bytes from 12.0.0.1: icmp_req=1 ttl=64 time=0.413 ms
64 bytes from 12.0.0.1: icmp_req=2 ttl=64 time=0.342 ms
64 bytes from 12.0.0.1: icmp_req=3 ttl=64 time=0.378 ms
64 bytes from 12.0.0.1: icmp_req=4 ttl=64 time=0.409 ms
64 bytes from 12.0.0.1: icmp_req=5 ttl=64 time=0.372 ms
64 bytes from 12.0.0.1: icmp_req=6 ttl=64 time=0.351 ms
64 bytes from 12.0.0.1: icmp_req=7 ttl=64 time=0.361 ms
64 bytes from 12.0.0.1: icmp_req=8 ttl=64 time=0.385 ms
^C
--- 12.0.0.1 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 6996ms
rtt min/avg/max/mdev = 0.342/0.376/0.413/0.029 ms
[edit]
vyatta@CE11# _
```

Figure-24: Successful ping Result from Customer CE11 to CE1

So, it can be seen that even though the IP backbone core routers P1 and P2 do not have any routes of Customers CE1 and CE11, still the customers can ping each other. This shows that the packet is travelling through the successfully created VXLAN tunnel between the two Open vSwitch instances PE1 and PE2, through the OVSDb plugin, and the flows are added through OpenFlow protocol. The packets between the customers travel through the VXLAN tunnel which is the overlay network built over the underlying IP backbone core OSPF network, where the latter is there to provide the IP connectivity for the tunnel without having any knowledge of the Customer IP addressing and Customer packets flowing through them. This is the beauty of overlay networks built using Software Defined Networking.

CHAPTER 5 – CONCLUSION

SDN decouples the control plane from the forwarding plane in the network, enabling the network intelligence to be centralized in the SDN based controller while having the underlying infrastructure to be abstracted for applications and network services.

By this implementation, it is clear that since we have the management power in our hands with the help of OVSDB protocol, there is no need to go back to the Ubuntu machine VM's having the Open vSwitches to configure any of their bridges or ports or tunnels. OVS instances are configured and managed by the OVSDB mechanism, which keeps updating the client about any change happening in the OVSDB instance. So, through SDN, all the three planes are nicely meshed together. The management channel is driven through CLI or GUI i.e., through OSGi Framework or Rest API; the control channel works through OVSDB and OpenFlow Protocols and finally the data channel through the flows being programmed which tell the Open vSwitch about the forwarding decision to be taken and the action to be implemented. So in a nutshell, we have successfully driven all the three channels (management, control and data channel) and meshed them together in this project through Software Defined Networking (SDN).

Apart from providing many clear advantages in a conventional network, like cost saving, central control, dynamic response, flexible and secure networks, etc., SDN still suffers from some limitations like some effort will be required from the company to implement the topology change in terms of clearing up old systems, training for new system; enterprise grade issues i.e., not all legacy system's protocols and standards might be compatible with SDN; central point of failure.

CHAPTER 6 – FUTURE SCOPE

The future scope of this project can be implementing this scenario using OpenFlow1.3 plugin and making use of the integration mechanism of OpenDaylight, OpenStack Neutron ML2, OVSDb. This uses Model Driven Service Abstraction Layer (MDSAL) as compared to API Driven Service Abstraction Layer (ADSAL) used in my topology. Provider Edge can be made as Openflow switches running in Hybrid mode. There could be a routing server, which keeps track of the routing database and produce routing table accordingly, which can be sent to ODL controller through Rest API. This project scenario can be integrated and implemented in future and extensions can be made according to the need.

REFERENCES

1. Network virtualization now, and the overlay network future
<http://searchsdn.techtarget.com/video/Network-virtualization-now-and-the-overlay-network-future>
2. Northbound interface / southbound interface
<http://whatis.techtarget.com/definition/northbound-interface-southbound-interface>
3. Getting Started with OVSDB
<http://networkstatic.net/getting-started-ovsdb/>
4. The Basic Introduction of Open vSwitch
<http://www.slideshare.net/teyenliu/the-basic-introduction-of-open-vswitch>
5. OpenDaylight OVSDB and OpenStack Neutron Integration
<https://www.youtube.com/watch?v=NayuY6J-AMA>
6. Project Proposals:OVSDB-Integration
https://wiki.opendaylight.org/view/Project_Proposals:OVSDB-Integration
7. Advantages, drawbacks and why do we actually need SDN.
<https://www.youtube.com/watch?v=TuvHxJrezpI>
8. OpenDaylight- Technical Overview
<http://www.opendaylight.org/project/technical-overview>
9. Release/Helium/Virtualization/User Guide
https://wiki.opendaylight.org/view/Release/Helium/Virtualization/User_Guide
10. Lightness-Low latency and high throughput dynamic network infrastructures for high performance datacentre interconnects
<http://cordis.europa.eu/docs/projects/cnect/6/318606/080/deliverables/001-lightnessD43finalpdf.pdf>
11. OVSDB Integration:Mininet OVSDB Tutorial
https://wiki.opendaylight.org/view/OVSDB_Integration:Mininet_OVSDB_Tutorial
12. SDN and Legacy Network Infrastructure
<http://www.enterprisenetworkingplanet.com/datacenter/sdn-and-legacy-network-infrastructure.html>
13. OpenFlow, Software Defined OpenFlow, Software Defined Networking (SDN) and Network Networking (SDN) and Network Function Virtualization (NFV)
http://www.cse.wustl.edu/~jain/tutorials/ftp/sd_hs14.pdf

14. Legacy Network
<http://www.techopedia.com/definition/25121/legacy-network>
15. Five SDN Benefits Enterprises Should Consider
<http://www.networkcomputing.com/networking/five-sdn-benefits-enterprises-should-consider/a/d-id/1234292>
16. OpenStack Neutron ML2 + OpenDaylight + OVSDb + OF13 End-to-End
<https://www.youtube.com/watch?v=0APNbgiRD8o>
17. Introduction to Cloud Overlay Networks-VXLAN
https://www.youtube.com/watch?v=Jqm_4TMmQz8
18. VXLAN Overlay Networks with Open vSwitch
<https://www.youtube.com/watch?v=tnSkHhsLqpM>
19. Introduction to Open vSwitch
<https://www.youtube.com/watch?v=rYW7kQRyUvA>
20. Open vSwitch Deep Dive The Virtual Switch for OpenStack
<https://www.youtube.com/watch?v=x-F9bDRxjAM>
21. Introduction to SDN
https://www.youtube.com/watch?v=DiChnu_PAzA
22. OpenFlow flow entries on Open vSwitch (OVS)
<https://www.youtube.com/watch?v=FyV4MoQ3T0I>
23. Network Virtualization – Openstack+ODL integration
<https://screeninet.wordpress.com/2014/04/19/network-virtualization-and-odlopenstack-integration/>
24. Dell PowerEdge R420
http://www.dell.com/downloads/global/products/pedge/r420_spec_sheet.pdf
25. Management, Control and Data Planes in Network devices and systems
<http://blog.ipspace.net/2013/08/management-control-and-data-planes-in.html>
26. Software Defined Networking
<https://www.coursera.org/course/sdn1>
27. Accelerating Open vSwitch to “Ludicrous Speed”
<http://networkheresy.com/2014/11/13/accelerating-open-vswitch-to-ludicrous-speed/>
28. Software-Defined Networking (SDN) Definition
<https://www.opennetworking.org/sdn-resources/sdn-definition>

APPENDIX – I

ROUTER CONFIGURATIONS

Router P1-

```
interfaces {
  ethernet eth0 {
    address 11.0.0.10/30
    hw-id 00:0c:29:7f:53:bc
  }
  ethernet eth1 {
    address 11.0.0.13/30
    hw-id 00:0c:29:7f:53:9e
  }
  loopback lo {
    address 2.2.2.2/32
  }
}
protocols {
  ospf {
    area 0 {
      network 11.0.0.12/30
      network 11.0.0.8/30
      network 2.2.2.2/32
    }
    parameters {
      router-id 2.2.2.2
    }
    redistribute {
      connected {
      }
    }
  }
}
system {
  host-name P1
  login {
    user vyatta {
      authentication {
        encrypted-password *****
      }
    }
  }
  syslog {
    global {
      facility all {
        level notice
      }
      facility protocols {
        level debug
      }
    }
    user all {
      facility all {
        level emerg
      }
    }
  }
}
```

Figure-25: Show configuration of P1

Router P2-

```
interfaces {
  ethernet eth0 {
    address 11.0.0.9/30
    hw-id 00:0c:29:92:32:7e
  }
  ethernet eth1 {
    address 11.0.0.6/30
    hw-id 00:0c:29:92:32:60
  }
  loopback lo {
    address 3.3.3.3/32
  }
}
protocols {
  ospf {
    area 0 {
      network 11.0.0.8/30
      network 11.0.0.4/30
      network 3.3.3.3/32
    }
    parameters {
      router-id 3.3.3.3
    }
    redistribute {
      connected {
      }
    }
  }
}
system {
  host-name P2
  login {
    user vyatta {
      authentication {
        encrypted-password *****
      }
    }
  }
  syslog {
    global {
      facility all {
        level notice
      }
      facility protocols {
        level debug
      }
    }
    user all {
      facility all {
        level emerg
      }
    }
  }
}
```

Figure-26: Show configuration of P2

Router CE1-

```
interfaces {
    ethernet eth0 {
        address 12.0.0.1/24
        hw-id 00:0c:29:1e:a7:ee
    }
}
protocols {
    static {
        interface-route 0.0.0.0/0 {
            next-hop-interface eth0 {
            }
        }
    }
}
system {
    host-name CE1
    login {
        user vyatta {
            authentication {
                encrypted-password ****
            }
        }
    }
    syslog {
    }
}
:~
```

Figure-27: Show configuration of CE1

```
vyatta@vyatta:~$ show ip route
Codes: K - kernel, C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       > - selected route, * - FIB route, p - stale info

Gateway of last resort is 0.0.0.0 to network 0.0.0.0

S    *> 0.0.0.0/0 [1/0] is directly connected, eth0
C    *> 12.0.0.0/24 is directly connected, eth0
C    *> 127.0.0.0/8 is directly connected, lo
vyatta@vyatta:~$
```

Figure-28: Routing table of CE1

Router CE11-

```
interfaces {
    ethernet eth0 {
        address 13.0.0.1/24
        hw-id 00:0c:29:b1:bb:c6
    }
}
protocols {
    static {
        interface-route 0.0.0.0/0 {
            next-hop-interface eth0 {
            }
        }
    }
}
system {
    host-name CE11
    login {
        user vyatta {
            authentication {
                encrypted-password ****
            }
        }
    }
    syslog {
    }
}
```

Figure-29: Show configuration of CE11

```
vyatta@vyatta:~$ show ip route
Codes: K - kernel, C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       > - selected route, * - FIB route, p - stale info

Gateway of last resort is 0.0.0.0 to network 0.0.0.0

S    *> 0.0.0.0/0 [1/0] is directly connected, eth0
C    *> 13.0.0.0/24 is directly connected, eth0
C    *> 127.0.0.0/8 is directly connected, lo
vyatta@vyatta:~$
```

Figure-30: Routing table of CE11