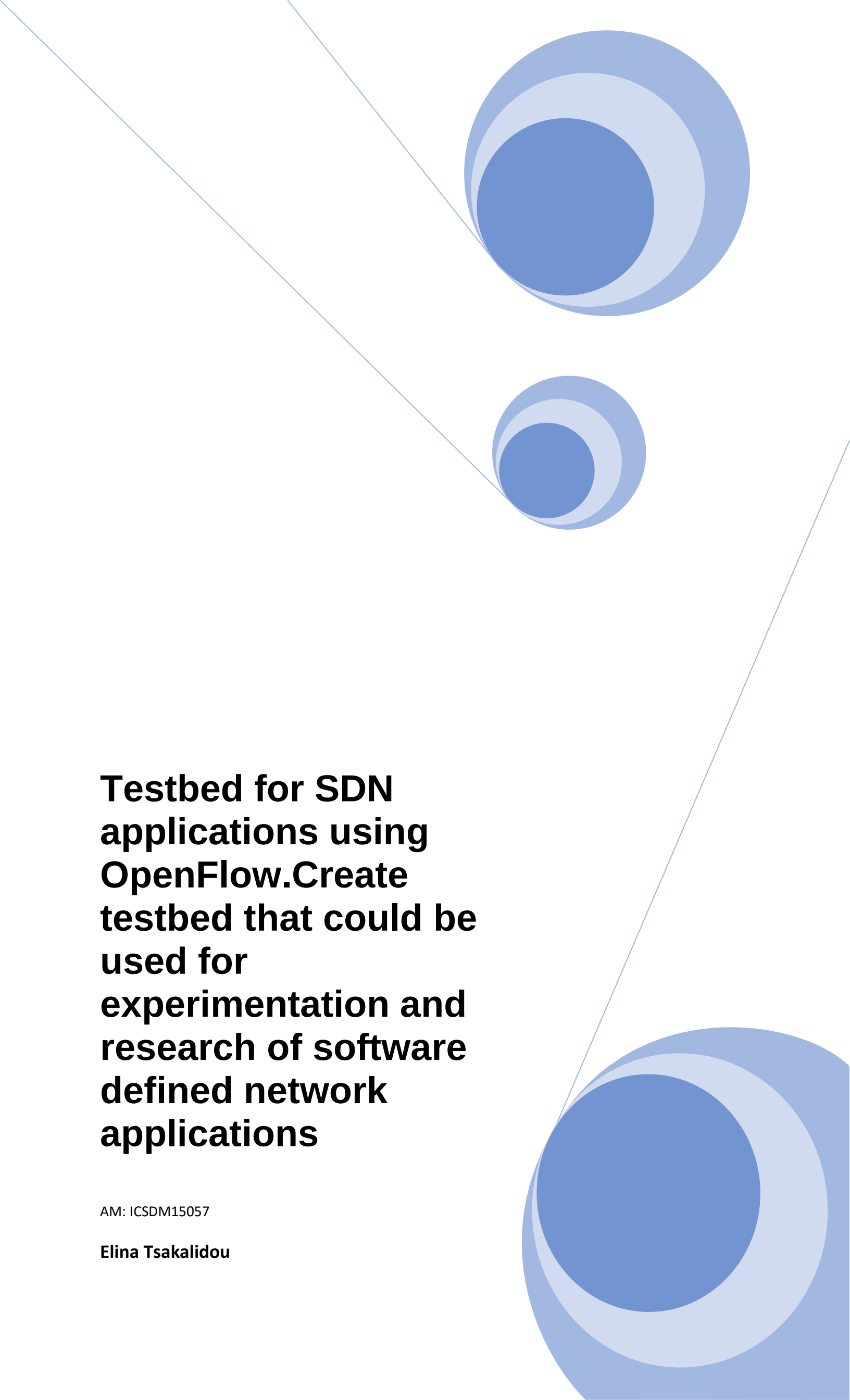


A decorative graphic featuring three concentric blue circles of varying sizes. Two circles are in the upper right quadrant, and one is in the lower right quadrant. Thin blue lines extend from the top left and bottom right towards the circles.

Testbed for SDN applications using OpenFlow.Create testbed that could be used for experimentation and research of software defined network applications

AM: ICSDM15057

Elina Tsakalidou

Contents

1 Introduction 4

 1.1 Traditional legacy network vs Software Defined Network 4

2 SDN Fundamentals..... 5

 2.1 SDN basic programming logic 5

3 Understanding OpenFlow Protocol..... 6

 3.1 OpenFlow Components 6

 3.1.1 OpenFlow Flow Tables 6

 3.1.2 OpenFlow Meters 8

 3.1.3 OpenFlow Groups 8

 3.2 OpenFlow Architecture 9

4 OpenDaylight Fundamentals 10

 4.1 OpenDaylight Architecture 11

 4.2 OpenDaylight Features-Applications 11

 4.2.1 DLUX..... 12

 4.2.2 L2 Switch 12

 4.2.3 OpenFlow Plugin 13

5 OpenDaylight Deployment in VirtualBox 16

 5.1 OpenDaylight Deployment Karaf Distribution 16

 5.1.1 Preparing the VM machine to host the ODL controller 16

 5.1.2 Java Installation..... 20

 5.1.3 Downloading and running of the karaf container..... 20

 5.1.4 Installing karaf features 21

 5.2 ODL Deployment Clustering..... 22

 5.2.1 Clustering Specifications 22

 5.2.2 Clustering in Practice 23

6 Mininet..... 27

 6.1 Why to use Mininet..... 27

 6.2 Mininet-Deployment..... 27

 6.2.1 Installing Mininet VM..... 27

 6.3 Mininet build-in Tools 30

7 Integration of Mininet with OpenDaylight controller 33

Analyzing a random node from Table 0 matching rules action sets of each flow follows the OpenFlow protocol rules for **openflow :1** and **openflow:2** ,after ping command. 56

8 Host isolation with Virtual Tenant Network (VTN) 66

9 AAA (Authentication-Authorization, Accounting)..... 80

Abstract

Software-defined network (SDN) has become one of the most important architectures for the management of largescale complex networks, which may require re-policing or reconfigurations from time to time. SDN achieves easy re-policing by decoupling the control plane from data plane. Thus, the network routers/switches just simply forward packets by following the flow table rules set by the control plane. Currently, OpenFlow is the most popular SDN protocol/standard and has a set of design specifications. The *Intorduction* of this thesis will present the basic advantages to choose the SDN configuration instead of traditional network configuration. *Chapter 2* presents the SDN basic concepts and architectures. *Chapter 3* introduces the SDN protocol OpenFlow protocol, its usage and components. *Chapter 4* presents one widely used SDN controller the OpenDaylight which is used in this testbed and is the control plane of the SDN architecture. *Chapter 5* the SDN deployment on a virtual cloud infractructure. *Chapter 6* presents the Mininet tool which is the data plane of the SDN architecture. *Chapter 7* presents the deployment of the SDN testbed with Mininet and OpenDaylight controllep. *Chapter 8* presents hot to isolate a network via OpenDaylight feature Virtual tenant manager. And finally *Chapter 9* uses the AAA service of OeprnDaylight to create a user.

1 Introduction

1.1 Traditional legacy network vs Software Defined Network

The last years in networking industry there is a tendency to have a centralized management system that allows the network programmability and automation in order to develop intent based network. Software-Defined Network (SDN) is a flourishing technology that approaches this type of management and more network providers are convinced to build confidence on how SDN works and what are the benefits of it. Demand for SDN solutions are rising rapidly due to existing problems in traditional legacy networks. The advantages of the SDN are presented and they explain why SDN wins against the traditional network.

Infrastructure: In order to deploy SDN solutions it does not require the existence of physical hardware (switches, routers, cables, etc). On the other hand, SDN is composed of software-based infrastructure. Devices of the network are software-based virtual entities both in control and data plane able to support any SDN deployed application.

Scalability: SDN is more scalable in contrast with a traditional network is thee drawn inference that is drawn from previous statement. It is easier and faster to add and remove resources without generating side effects for the rest of the network resources and functions. Resources demands are solved from mouse clicks, in a traditional network however this means money cost and manual configuration that takes more time. another advantage of SDN is the in integration with cloud applications. SND provides integration with cloud applications and network virtual functions (NFV).

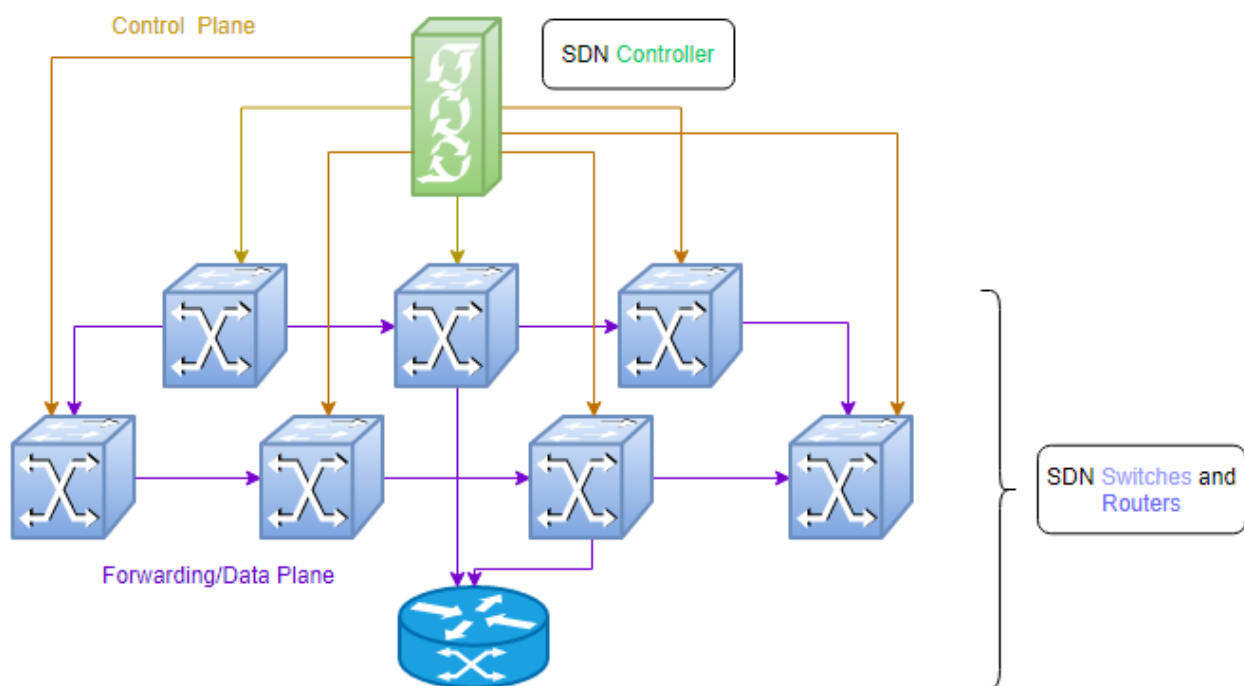
Traffic Management: Another main difference is that in a traditional network the decisions about traffic management are configured in data plane. The data plane and control plane are in same box. In others words, there is already build in software logic in switches that will handle the traffic. In SDN the traffic management is configured from control plane. The data plane is not responsible to program the forwarding logic of the traffic, it sends the packets as they are programmed from control plane.

Security: SDN provides customized security between the end user, the data center and the network traffic. Security policies that are easily defined in end-to-end network comparing to the legacy network [1]

2 SDN Fundamentals

Software Defined Network (SDN) was launched from collaboration of Stanford University and the University of California at Berkeley in 2008. SDN is a dynamic network that separates the control plane from data plane and its components consists of two main parts:

- The SDN controller or control system which is always located on the top layer in SDN architecture refers to control plane.
- The SDN switches or forwarding elements which are always located in a layer below from the control plane in the SDN architecture.
- Interface between the control plane and data plane, a common interface that is connects the SDN elements is the OpenFlow protocol.



The network **control plane** is composed of the SDN controller which is the “brain” of the network and is responsible to manage the functions of data plane components. SDN uses the OpenFlow protocol in order to manage configurations of switches and/or routers. Nowadays, there are plenty of open-source SDN controllers, but also many companies have privatized they own controllers for commercial purposes.

The **data plane** is a lower layer that is included the SDN switches. The management of the forwarding packets take place in the data plane, however how the packets will be forwarded is decision of the control plane using the OpenFlow protocol or any other SDN protocol.

2.1 SDN basic programming logic

Traditional network switches have already pre-installed programming logic in their software. Also, a traditional network switch contains fixed table entries that define the routing rules by switch software, consequently a switch has already a prior-knowledge on how to forward the received frame/packet. SDN has different programming logic comparing to traditional network, there is no built-in programmed traffic logic in switches and the controller decides what to do with the received packet from switch and then fills in the tables that are called in SDN world **flow tables** with **flow entries** of the switch. The controller programs the traffic logic of the switch based on SDN southbound protocols. There are plenty of southbound protocols, a widely-known southbound protocol is the **OpenFlow** protocol. OVSDB protocol along with the NETCONF are also familiar SDN southbound protocols. Next chapter analyzes the OpenFlow protocol, since it is used in testbed. The rest of protocols will be presented in the section of the OpenDaylight features.

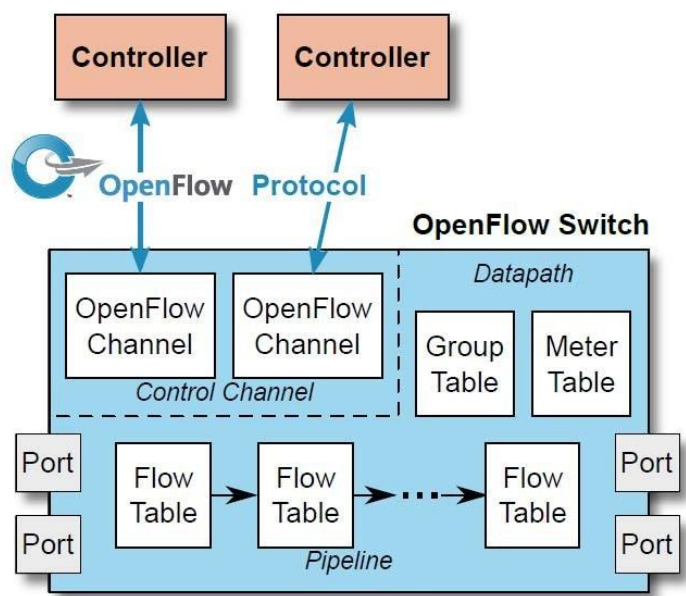
3 Understanding OpenFlow Protocol

OpenFlow protocol was the first protocol that was adopted from SDN, it was created by ONF (Open Networking Foundation) which is an organization that creates standards for SDN. OpenFlow Protocol is a key component in SDN solutions. OpenFlow protocol stands between the data and control plane. Its initial intention was slightly different comparing to the current functionality. The first version 1.0 of the OpenFlow protocol specification was released in December 2009, the latest version is 1.6. In aspect of this research the used version is 1.3.

The main purpose of the OpenFlow is to update the flow tables of the switch or router through SDN controller involvement, since the control plane is the one that configures how the flow tables will be updated. OpenFlow is a software running on each switch in SDN and communicates with the SDN controller.

3.1 OpenFlow Components

OpenFlow defines flow tables, groups tables, and meter tables. The next figure presents how they distributed in an SDN switch [2].

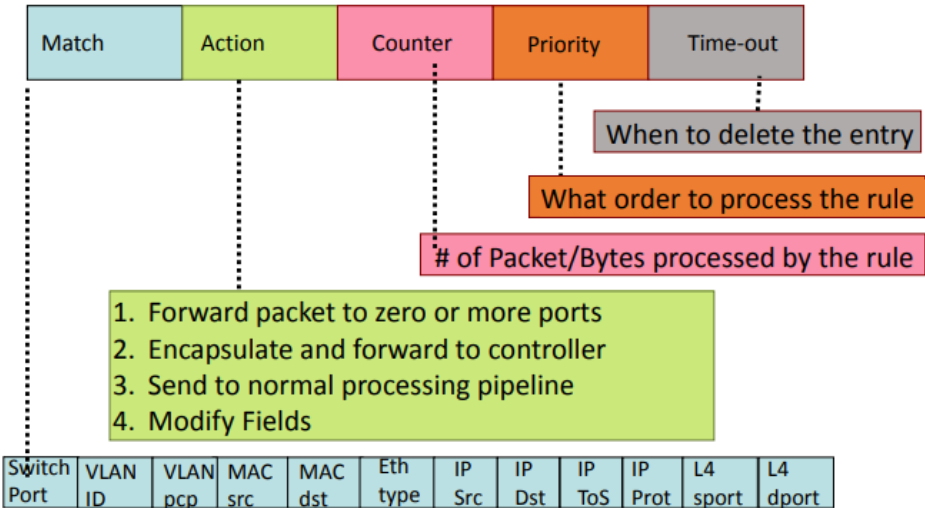


3.1.1 OpenFlow Flow Tables

OpenFlow tables define a pipeline to process a packet header. A pipeline may contain one or many flow tables. Every table in the pipeline handles the input received from the previous flow table.

Each flow table consist of table flow entries, flow entry has data such as:

- Matching rules: When a packet is reached the port the packer header is matched regarding the fields it has in its header e.g port number, destination port, source port etc.
- Instructions: Another important field in a flow table entry is the instruction, which is a decision taken on what to do with the packet obeying the matching fields. The instructions field is a set of actions. An action can be anything among apply_actions, clear_actions, write_actions, write_metadata, goto_table.
- Statistics: Keeps track of the number of times the flow has been matched.



Example of matching rules:

The next figures present the taken action based on the matching rule.

Switching is relevant to L2 switching the matching rule is the destination MAC address (00:1f) and the action is to forward the packet to port 6 of the switch. In other words, this means that when the packet header contains 00:1f... for a destination MAC address then this packet must be forwarded to port 6 of switch.

Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6

Flow switching complies with the complex combination of matching rules in order for the packet to be forwarded to port 6.

Flow Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:20:..	00:1f:..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

The next flow entry describes a firewall that will drop the packet.

Firewall

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

L3 routing the matching rule is the destination IP address (5.6.7.8) and the action is to forward the packet to port 6 of the switch.

Routing

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	5.6.7.8	*	*	*	port6

When the packet header contains the VLAN it isolates the network. As a result, the next figure present that a flow entry with VLAN in header and destination MAC address will be outcast to port

Routing

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	5.6.7.8	*	*	*	port6

VLAN Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f..	*	vlan1	*	*	*	*	*	port6, port7, port9

3.1.2 OpenFlow Meters

OpenFlow meters is another component of the OpenFlow protocol. It was first introduced in OpenFlow version 1.3.0 as an optional feature. A meter is a switch element which measures and controls the ingress rate of traffic of packets assigned to it. Ingress rate is the rate of packets prior to the output. Similar to flows meters are generated in the meter table and consist of meter entries which define the meters. The meters are attached directly to the flow entries. Each flow entry can specify a meter in its instructions set. The meter measures and controls the rate of the aggregate of all flow entries to which it is attached. Flows direct packets to the specified meter using the goto-meter instruction, as a result the meter can perform operation based on the rate it receives. Per-flow meters enable OpenFlow to implement various **Quality of Service** operations, such as rate-limiting which is the main application of the meters. However, meters can be combined with another features like queues to provide more advanced services.

A meter entry in the meter table is composed of the following elements:

- **meter identifier:** a 32 bit unsigned integer uniquely identifying the meter
- **meter bands:** an unordered list of meter bands, where each meter band specifies the rate of the band and the way to process the packet
- **counters:** updated when packets are processed by a meter

The main element of the meter entry is the meter band which specifies the rate at which meter is applied and the way packets should be processed. A meter can have one or more-meter bands but only a single band is applied for a flow at a time based on the measured packets rate. The meter applies the meter band with the highest configured rate that is lower than the current measured rate. If the current rate is lower than any specified meter band rate, no meter band is applied. The meter triggers a meter band if the packet rate or byte rate passing through the meter exceeds a predefined threshold. If the meter band drops the packet, it is called a rate limiter

Each meter band is identified by its rate and contains:

- **band type:** defines how packets are processed
- **rate:** used by the meter to select the meter band, defines the lowest rate at which the band can apply
- **counters:** updated when packets are processed by a meter band
- **type specific arguments:** some band type has optional arguments

A meter is for example a simple token bucket policer that can be instantiated and configured to a certain rate and burst. Whenever a flow exceeds the bucket’s rate, the packet is dropped. In this case the meter is identified as late limiter and this is the main application of the meters. If the packet complies with its traffic definition and the burst is not exceeded, the remaining actions in the action set will be executed. Another functionality of the meters is to achieve a specific (Quality of Service) [4] [5].

3.1.3 OpenFlow Groups

OpenFlow groups was first introduced in version 1.1. Similar to a Flow and Meter, a group also consists of entries, as result the group entries make the Group table. OpenFlow groups are also elements defined from the OpenFlow specification and they created to support functions that flows are unable to execute. OpenFlow groups provide advanced services in order to solve real-time networking issues. Groups are forwarding the packets when the flows are unable to perform any actions to them. Unlike flows

OpenFlow groups do not define matching rules nor instructions. OpenFlow specification supports different group types and each group type is dedicated to apply specific actions to the packet.

When the packet enters the group table, it receives actions, however the group is not allowed to forward the packet to any flow table, neither meter tables. Each group contains a list of actions lists that are known as **list of buckets** and they are applied to the ingress packets. Each group may define zero to many buckets. When a group does not contain any bucket, this means the packet remains untouched. Also, there are cases that the bucket contains a list of actions that order the packets to be sent to the next groups.

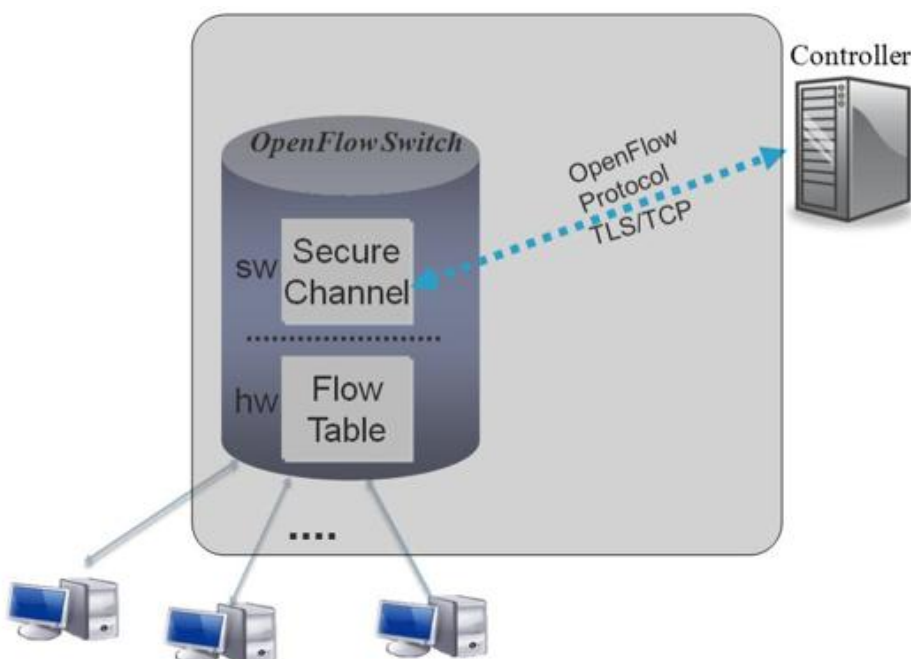
The group types are classified in four categories [6]

- **ALL:** This is the simplest group type. Takes as an input the ingress packets and reproduces it in order to handle it in each bucket. As a result, for each replica of the original packet different set of actions are performed.
- **SELECT:** This group is using for load balancing. Every bucket that contains a list of actions has a specified weight. An ingress packet is forwarded to a single bucket, target bucket is selected based on the bucket weight.
- **INDIRECT:** This group contains only one bucket and all packets are transferred to this bucket. This group consolidate a common set of actions, as a result memory consumption is significantly reduced.
- **FAST-FAILOVER:** This is the most significant group of all group types dedicated to handle the cases of network failures. This group has many buckets and each bucket is defined from watch port and an optional watch group. The watch port and/or group detects the active status of the indicated port and/or group. Only in case the port is active the bucket is usable. When a specific bucket is used other buckets cannot be used. Bucket will be replaced with other when the watch port or group will be deactivated. The bucket selection of the FAST-FAILOVER will be the nearest bucket in the bucket list with a watch port or group that is up.

3.2 OpenFlow Architecture

Based on ONF OpenFlow protocol is an interface that communicates the control plane with the data plane of an SDN architecture. The OpenFlow architecture us a composition of three elements see following figure [7]:

- The OpenFlow controller
- The OpenFlow switch
- The OpenFlow channel



4 OpenDaylight Fundamentals

This chapter provides general information about the OpenDaylight or ODL which is an open-source SDN project implemented in Java language. It was created by the Linux Foundation and its first release (Hydrogen) was announced in February 2014. The purpose of the ODL project is to decouple the networking hardware from the software and allow the end users to build networking applications with the concept of plug-n-play architecture. The ODL controller platform is considered as a modular SDN controller due to many modules that are embraced in one single platform. It can be installed on Linux, Windows, Macintosh Operating Systems and any other that supports Java. Up to this time, there are ten releases Hydrogen, Helium, Lithium, Beryllium, Boron, Carbon, Nitrogen, **Oxygen**, Fluorine and Neon. Each release name of the ODL is based on the periodic table elements. ODL community announces at least two releases every year. The Oxygen SR4 (Stable Release 4) will be used as an SDN controller.

OpenDaylight supports [8]:

- **OSGi container:** OSGi (Open Services Gateway Initiative) is a framework, also known as the Dynamic System for Java defines a specification for deploying modular applications. Allows to break the applications into many modules that can be dynamically loaded and managed as bundles in the container. OSGi bundles are .JAR files with a MANIFEST.MF file with the last containing configuration for the OSGi. When a bundle is dependent from other bundles OSGi will start first these dependencies and next the bundle itself, otherwise the bundle will not start. As a result, a user can start, stop, install and uninstall modules without affecting the container. Currently, there are many open-source OSGi containers. Apache Karaf is a bundle used by the ODL in order to create the OSGi container where all OSGi bundles can be loaded and started [9] [10] [11].
- **Maven:** Maven is a tool for build automation usually for Java applications. Maven uses pom.xml (Project Object Model) to define the dependencies which are nothing that already implemented libraries to be used between the modules. It also can download libraries from a remote repository. Currently the most used remote repository for the ODL dependencies is the Nexus <https://nexus.opendaylight.org/>. Maven contains many phases which are the build lifecycles like install, test, clean, deploy, generate-sources etc. The next examples show how to define a dependency in a pom.xml and how to execute maven goal phase.

```
<dependency>
  <groupId>org.opendaylight.mdsal.binding.model.ietf</groupId>
  <artifactId>rfc8345-ietf-network-topology</artifactId>
  <version>1.2.6</version>
</dependency>
```

A maven dependency contains a **groupId**, an **artifactId** and **version** all fields included in tag <dependency> defined in XML language.

The next command shows how to build a maven project “testbed”.

```
sdn@sdn-Lenovo-G50-45:~/testbed$ mvn clean install
```

- **Java Interfaces:** Java interfaces are used for event listening, specifications, and forming patterns. This is the main way in which specific bundles implement call-back functions for events and also to indicate awareness of specific state.
- **Rest APIs:** These are part northbound interface. These RESTful APIs are implemented in order to be integrated custom applications. They also support GUI (Graphical User Interface) for ODL.

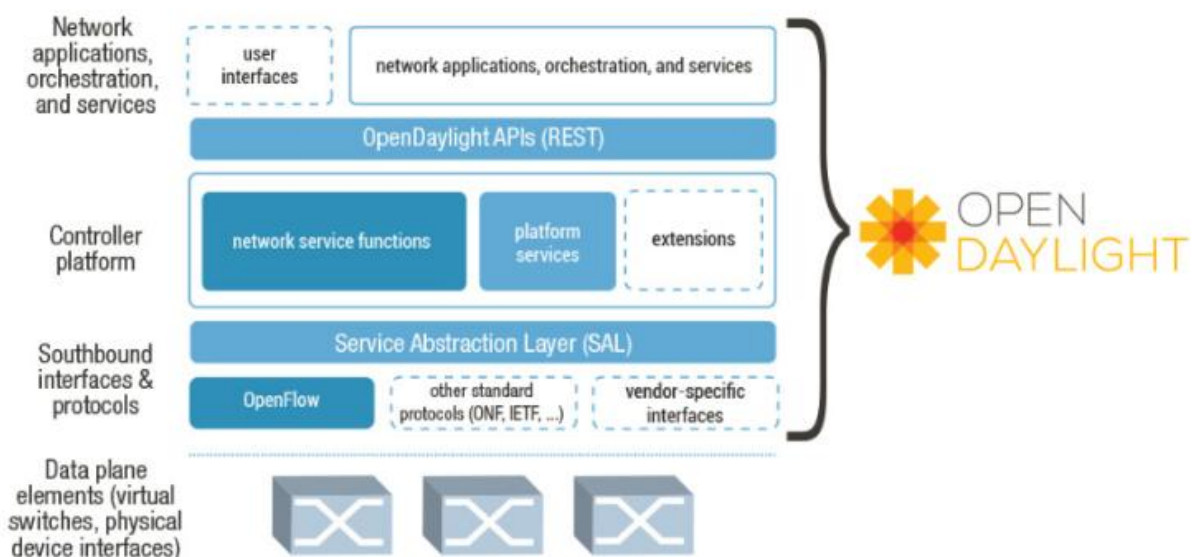
YANG: ODL platform is supports the YANG (Yet Another Next Generation) language, used for data modeling and generated payload for NETCONF protocol.

ODL supports the southbound OpenFlow protocol as well as other protocols. ODL allows to develop new applications and also use the already build applications to make an enhancement of any feature.

4.1 OpenDaylight Architecture

ODL supports a layered architecture with clear integration points and APIs that allow end users and networking vendors to participate in the power SDN capabilities of ODL. ODL supports a layered architecture with clear integration points and APIs that allow end users and networking vendors to participate in the power SDN capabilities of ODL. In general, ODL architecture consists of next four layers, each layer will be described separately [12]:

- **Northbound Layer:** is meant for communication with upper, Application layer and would be in general realized through REST APIs of SDN controllers.
- **Controller Platform Layer:** is meant for communication with lower layers, Infrastructure layer of network elements and would be in general realized through southbound protocols
- **Service Abstraction Layer:** Service abstraction layer is a component that is introduced only from ODL controller and it is located between the southbound protocols and the northbound protocols where third-party applications are supported. The first release of the ODL launched the API-driven Service Abstraction Layer (AD-SAL) which in next releases was enhanced and renamed to Model-driven service abstraction layer (MD-SAL).
- **Data plane Layer:** is composed of various networking equipment which forms underlying network to forward network traffic. It could be a set of network switches and routers in the data center. This layer would be the physical one over which network virtualization would be laid down through the control layer (where SDN controllers would sit and manage underlying physical network)



4.2 OpenDaylight Features-Applications

This section will sum up all ODL existing features and their functions in alphabetical order. The features used in testbed will be analyzed in detail:

- **Networking:** ALTO, BGPLS PCEP, BIER, CAPWAP, DIDM, FaaS, L2-Switch, LACP, LISP, NATApp Plugin, NETCONF, OF-CONFIG, OpenFlow, OpFlex, OVSDb, NetVirt, NIC, Neutron Northbound, P4, Packet Cable, SFC, TTP, VTN, VPN Service, Unimrg.
- **Security:** AAA, Controller Shield, USC.
- **Management:** Cardinal, DluxApps, EMAN, Federation, GBR, IoTDM, NEMO, NetIDE, OCP, SNMP, SNMP4SDN, SXP.
- **Core:** MD-SAL, OpenDaylight Controller, ODL-SDNi, YANG Tools.
- **Data Collectors:** Centinel, TSDR

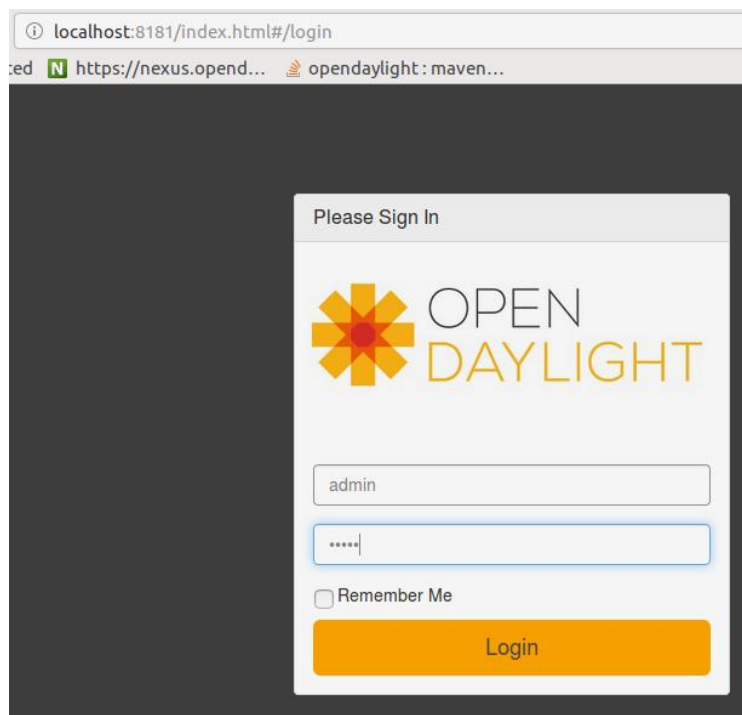
4.2.1 DLUX

This module is the web user interface of the ODL implemented in Angular JS. It is an OpenFlow management application for the ODL [13]. It provides authentication, navigation and lists the following features:

- Topology: Shows the OpenFlow topology components.
- Nodes: This is a very simple inventory node manager.
- YANG visualizer: This provides visualization of YANG models in graphical form.
- YANGMAN: This is an advanced and more user-friendly YANG UI replacement.
- YANG GUI: This is a simple UI for interaction with the controller. It is based on Yang models, and it renders a form so that users can read or write data even if they have no knowledge of the models.

For accessing the DLUX any web browser will work by entering the URL <http://localhost:8181/index.html/> providing the credentials “admin” for both fields access to this feature is offered.

The figure below shows how is the index page of the web interface.



Then the next figure shows the next page after login. The topology tab is empty, due to absence of devices in the network.

4.2.2 L2 Switch

L2 switch is an ODL module that provides Layer 2 switch functionality specifying how the packets should be forwarded [14]. L2 Switch comes along with other useful features:

- Packet Handler: This feature processes and decodes the incoming packets and forwards them appropriately.
- Loop Remover: Removed loops from the network.
- Arp Handler: Manages the decoded ARP packets
- Address Tracker: Retrieves the MAC Addresses and IP addresses of the elements existing in the network.
- Host Tracker: Tracks the host locations in the network

- L2 Switch Main: Installs flows on switches based on specific rules that must follow the network traffic.

When L2 switch receives a packet that does not match any entry in flow table it encapsulates the packet in an OpenFlow PACKET_IN message and sends this packet to the controller. Then L2 switch feature finds where it should be sent. The MAC address must be identified through OpenFlow PACKET_IN message. The next table summarizes how the L2 switch module identifies the MAC address.

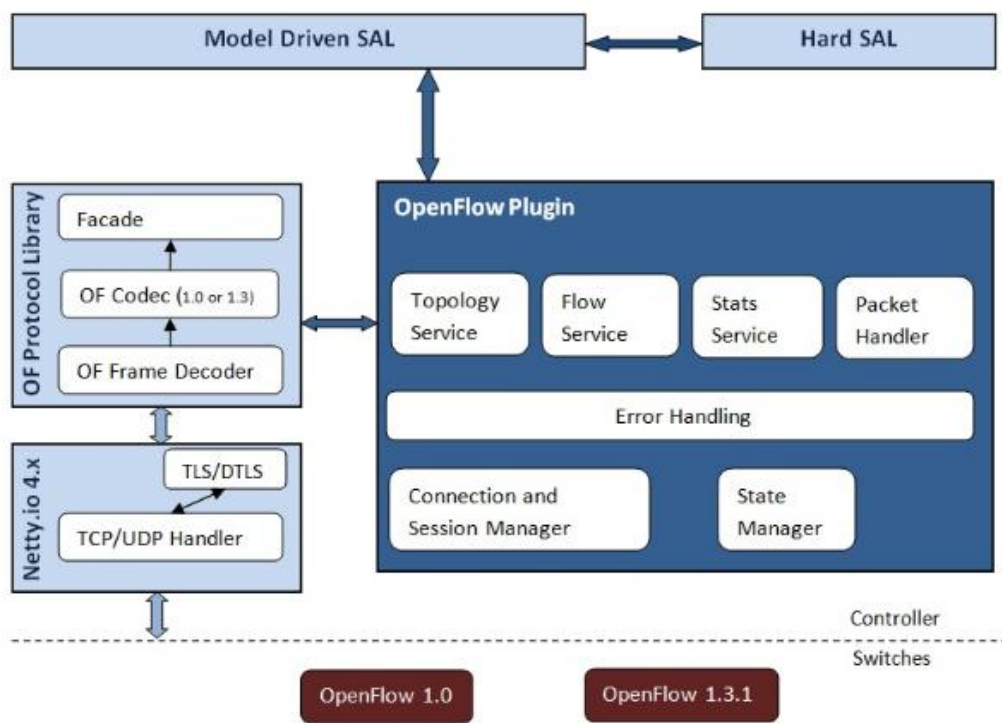
Source MAC	Destination MAC	Action
Unknown	Unknown	Broadcast the packet to all external ports except the ingress port
Unknown	Known	L2 module sends the packet to the node where the target is attached. The attachment point refers to the target that is physically attached.
Known	Unknown	Broadcast packets to all external ports. L2 switch module knows the source MAC.
Known	Known	Packet forwarded from source MAC to target MAC and installed flows in the flow tables of switches.

4.2.3 OpenFlow Plugin

The OpenFlow plugin is belongs to a southbound plugin of the ODL and defines is a communication Interface that allows interaction between the control and forwarding plane of an SDN. This plugin implements the OpenFlow standard [15]. The current versions of OpenFlow 1.0. and 1.3.x are supported, however it gives the opportunity to adopt the other version too. Similar to other modules of the ODL this plugin also is based on the Model Driven Service Abstraction Layer (MD-SAL). It allows TLS secure connection on port 6633 and non-secure connection on port 6653 to listen for OpenFlow messages coming from OpenFlow devices.

The following features are supported from the ODL plugin:

- Connection Handling
- Session Management
- State Management.
- Error Handling.
- Mapping function (Infrastructure to OF structures).
- Connection establishment will be handled by the OpenFlow library using opensource netty.io library.
- Message handling for example Packet in.
- Event handling and propagation to upper layers.
- Plugin will support both MD-SAL and Hard SAL.
- Will be backward compatible with OF 1.0.



5 OpeDaylight Deployment in VirtualBox

Nowadays, plenty amount of virtualization software exists free to download. The chosen virtualization program for this testbed is the **VirtualBox** that will host the ODL controller and next the Mininet tool. It can be installed easily without many manual configurations. Similar to other virtualization solutions, it provides specific network card, hard disk, graphics and RAM for every virtual machine.

This section prepares the VM that will host the ODL controller. Presents step-by-step guide to start and deploy features of SDN ODL controller.

Host operating system is Windows 10 Pro and hypervisor software is VirtualBox for hosting a virtual machine (Ubuntu 16.04) for ODL controller. The environment settings of VM are as follows:

Host Operating System Settings:

- Operating System Windows 10 Pro Version 1903
- Installed Memory RAM: 16GB
- Processor: Inter(R) Core(TM) i5-9600K CPU @ 3.70 GHz
- Operating System type: 64-bit x64-based processor
- Disk: 400GB

Software Settings:

- Operating System: Windows 10 Professional
- Hypervisor: Oracle VM VirtualBox Manager version 6.1
 - Operating System: Ubuntu Desktop Image 16.04.1 LTS
 - ODL Version: Oxygen SR4 having:
 - 40GB hard disk, 2048 MB RAM, 2 CPUs

There are two options to deploy the ODL controller:

- **Standalone** deployment: The ODL controller will run as one server. Used for simple use cases. Karaf container will be used in order to install any ODL feature.
- **Distributed** deployment: In distributed deployment there is a cluster where exist many ODL server instances that are working together as one entity and sharing a common configuration. Deploying ODL servers in a cluster assures there will be at least one ODL instance running in case of any other ODL server failure occurs. This is very important for real enterprise network systems that cannot accept failure. Consequently, when multiple ODL instances are running there are some advantages such as:
 - Scaling
 - High-Availability
 - Data Persistence

5.1 OpenDaylight Deployment Karaf Distribution

This section will present how to start the ODL controller as karaf distribution. The ODL as karaf distribution is an OSGI container that provides all features available to install, but none of them will start automatically, only after user command. The ODL karaf distribution version that is used in the scope of this investigation is Oxygen SR4 karaf-0.8.4.zip or karaf-0.8.4.tar.gz file format downloaded from [16]

5.1.1 Preparing the VM machine to host the ODL controller

The next images show step-by –step ODL deployment in virtual host.

?

×

← Create Virtual Machine

Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name:

ODL_VM

Machine Folder:

C:\Users\pnbuser\VirtualBox VMs

Type:

Linux

Version:

Ubuntu (64-bit)

Expert Mode

Next

Cancel

?

×

← Create Virtual Machine

Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **1024 MB**.

4 MB

16384 MB

2048 MB

Next

Cancel

?

×

← Create Virtual Machine

Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **10,00 GB**.

☐ Do not add a virtual hard disk

☒ Create a virtual hard disk now

☐ Use an existing virtual hard disk file

Empty

Create

Cancel

?

×

← Create Virtual Hard Disk

Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

☒ VDI (VirtualBox Disk Image)

☐ VHD (Virtual Hard Disk)

☐ VMDK (Virtual Machine Disk)

Expert Mode

Next

Cancel

?

×

← Create Virtual Hard Disk

Storage on physical hard disk

Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

☒ Dynamically allocated

☐ Fixed size

Next

Cancel

?

×

← Create Virtual Hard Disk

File location and size

Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

C:\Users\pnbuser\VirtualBox VMs\ODL_VM\ODL_VM.vdi

Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

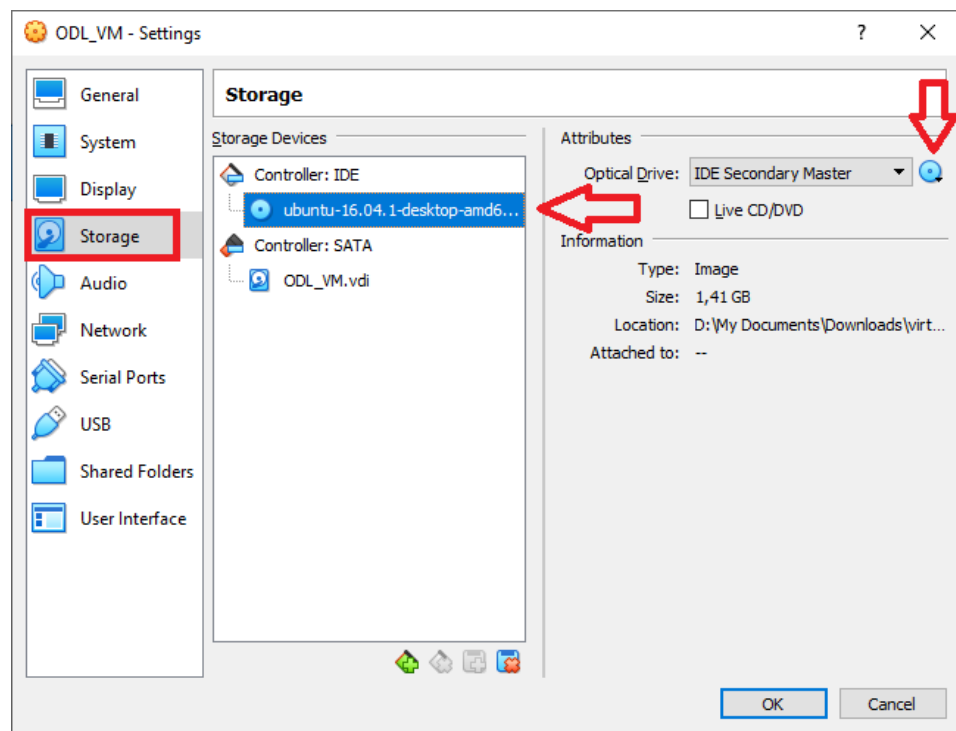
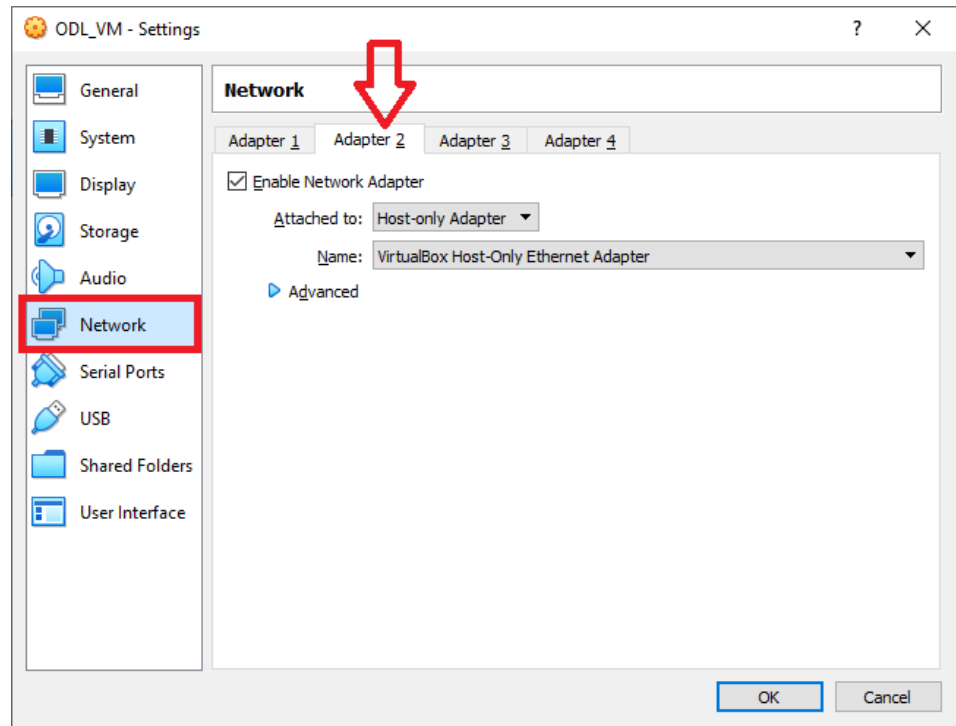
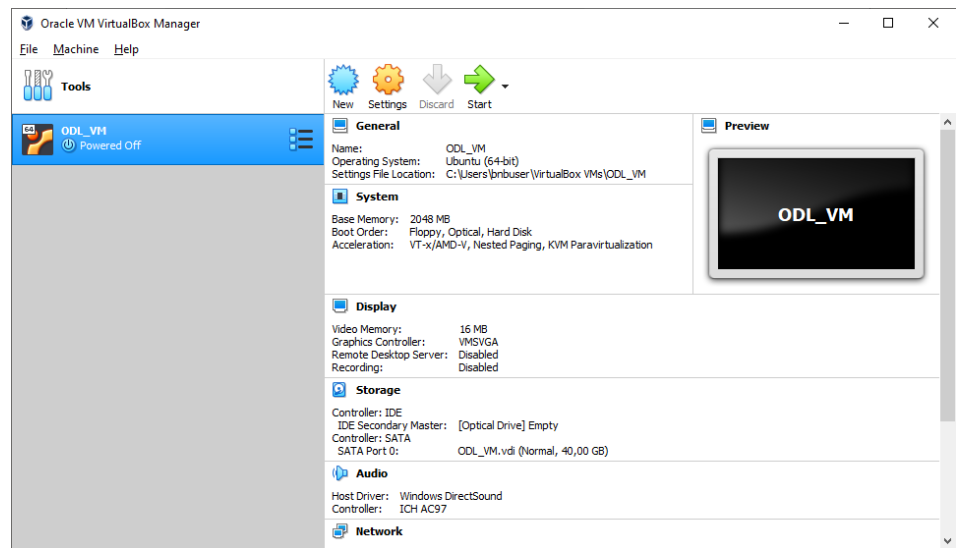
4,00 MB

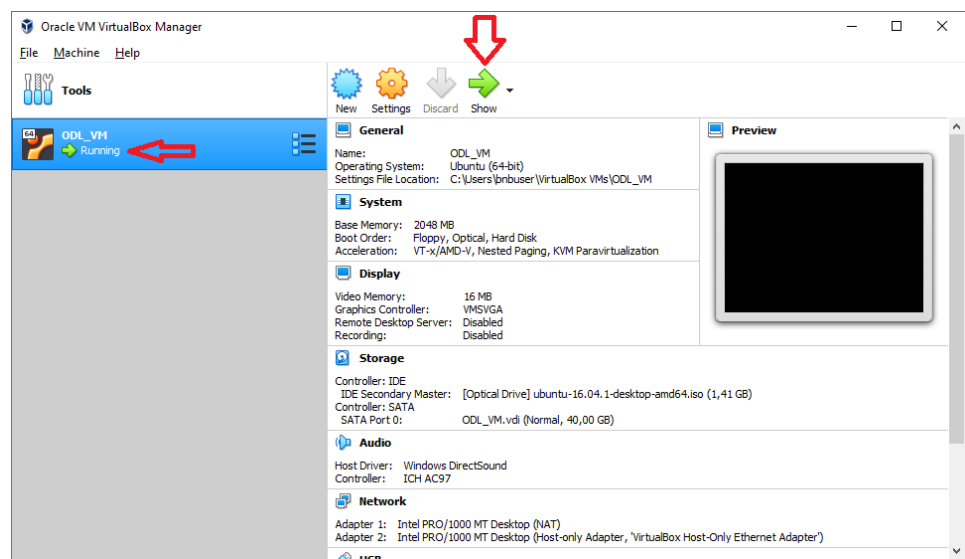
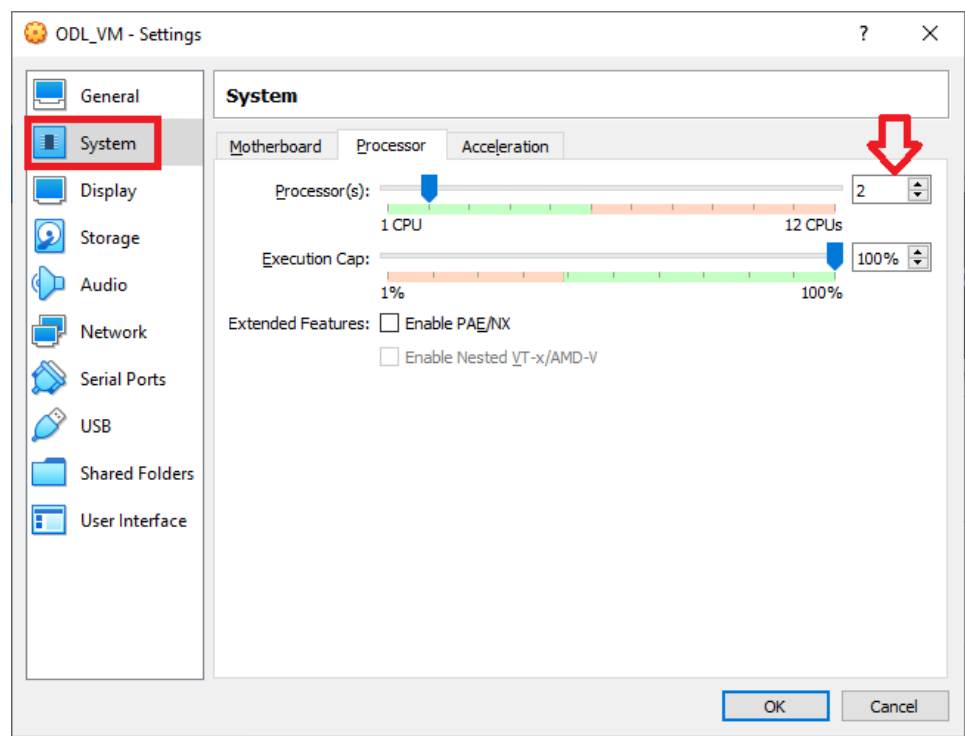
2,00 TB

40,00 GB

Create

Cancel





```
sdn@sdn-opendaylight:~$ ifconfig
enp0s3  Link encap:Ethernet  HWaddr 08:00:27:67:9f:12
        inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
        inet6 addr: fe80::2cda:5256:ef60:62c1/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:526351 errors:0 dropped:0 overruns:0 frame:0
        TX packets:204735 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:492862787 (492.8 MB)  TX bytes:12449398 (12.4 MB)

enp0s8  Link encap:Ethernet  HWaddr 08:00:27:57:f2:a4
        inet addr:192.168.56.101 Bcast:192.168.56.255 Mask:255.255.255.0
        inet6 addr: fe80::512b:b930:f993:b03b/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:910 errors:0 dropped:0 overruns:0 frame:0
        TX packets:67 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:274746 (274.7 KB)  TX bytes:8260 (8.2 KB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:594 errors:0 dropped:0 overruns:0 frame:0
        TX packets:594 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:60000 (60.0 KB)  TX bytes:60000 (60.0 KB)

sdn@sdn-opendaylight:~$
```

The main requirement for the deployment is the proper JDK version.

5.1.2 Java Installation

The latest releases of the ODL controller features require Java Development Kit (JDK) version 1.8. or later. The OpenJDK 1.8 will be installed, to resolve this requirement with the next command:

```
sdn@sdn-opendaylight:~$ sudo apt-get install openjdk-8-jdk
```

Verification of the JDK installed version.

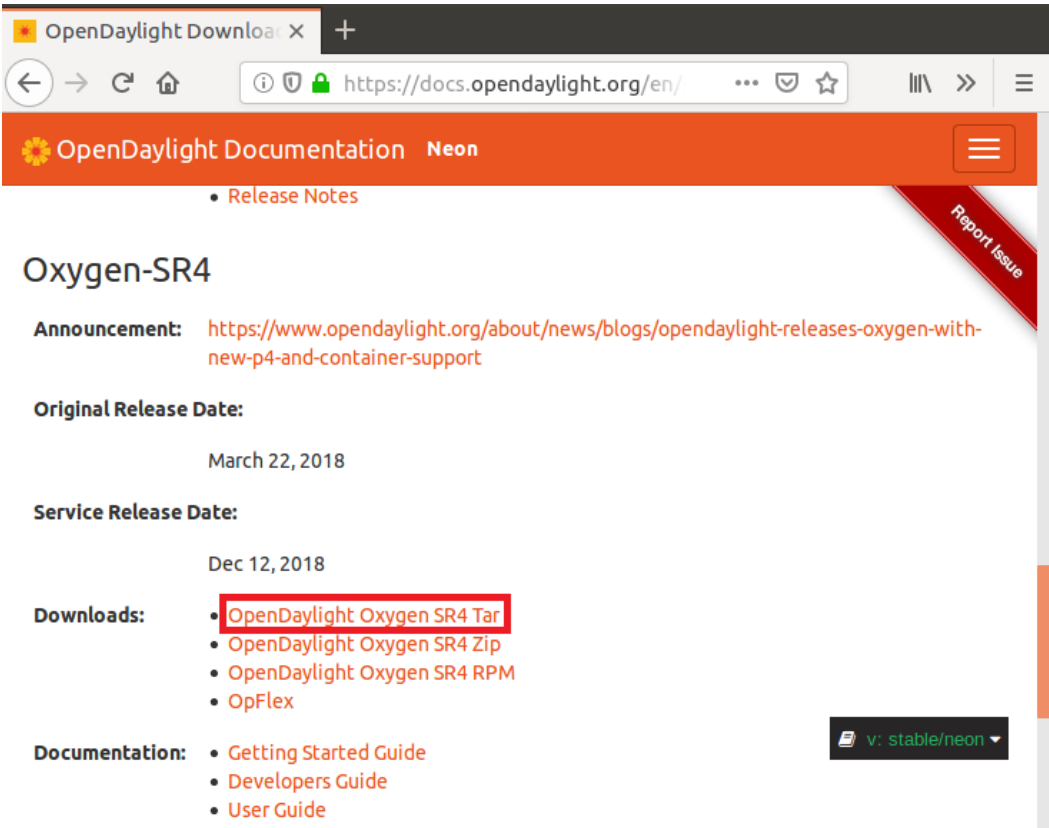
```
sdn@sdn-opendaylight:~$ java -version
openjdk version "1.8.0_222"
OpenJDK Runtime Environment (build 1.8.0_222-8u222-b10-1ubuntu1~16.04.1-b10)
OpenJDK 64-Bit Server VM (build 25.222-b10, mixed mode)
sdn@sdn-opendaylight:~$
```

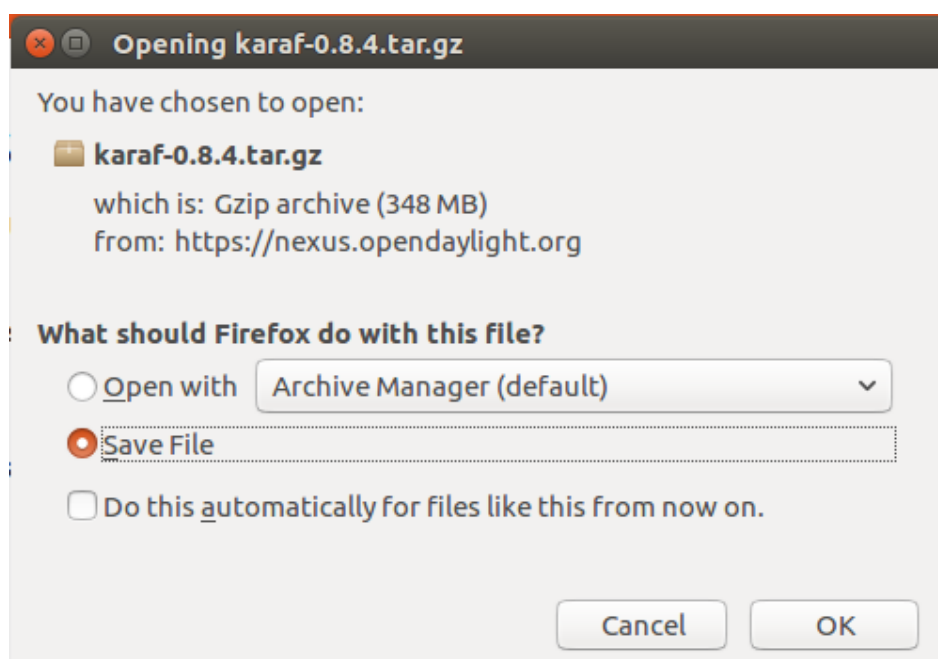
Settings for the environment variables JAVA_HOME to JAVA installed location and PATH.

```
sdn@sdn-opendaylight:/usr/lib/jvm/java-8-openjdk-amd64$ cd
sdn@sdn-opendaylight:~$ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
sdn@sdn-opendaylight:~$ export PATH=$JAVA_HOME/bin:$PATH
sdn@sdn-opendaylight:~$ echo $JAVA_HOME
/usr/lib/jvm/java-8-openjdk-amd64
sdn@sdn-opendaylight:~$
```

5.1.3 Downloading and running of the karaf container.

The next images show how to downloaad the distributed karaf Oxygen version [17].





Moving the **karaf-0.8.4.tar.kar** file to VM home directory.

```
sdn@sdn-opendaylight:~$ cd Downloads/
sdn@sdn-opendaylight:~/Downloads$ ll
total 356672
drwxr-xr-x  2 sdn sdn      4096 ΣEn  22 16:06 ./
drwxr-xr-x 16 sdn sdn      4096 ΣEn  22 15:51 ../
-rw-rw-r--  1 sdn sdn 365223735 Σen  22 16:06 karaf-0.8.4.tar.gz
sdn@sdn-opendaylight:~/Downloads$ mv karaf-0.8.4.tar.gz ~/
sdn@sdn-opendaylight:~$ tar -xzf karaf-0.8.4.tar.gz
```

Once the downloaded file is unzipped. Starting the ODL karaf can take place next. The following command shows how to start the ODL container.

```
$ sdn@sdn-openshift:~/karaf-0.8.4$ ./bin/karaf
```

[illegible]

5.1.4 Installing karaf features

Once the OD is in running state any feature can be installed and used as an SDN application. Any ODL feature can be activated by the following command in ODL CLI, where feature1 is the feature name.

```
opendaylight-user@root>feature:install <feature1>
```

There is an option to install many features simultaneously, by separating the features names with space, the next command shows how to install all DLUX modules.

```
opendaylight-user@root>feature:install odl-dluxapps-applications features-dluxapps
```

Another useful ODL commands are:

```
opendaylight-user@root>feature:uninstall <feature1> : uninstalls the feature1 from ODL karaf.
```

`opendaylight-user@root>feature:list`: shows all active features to be installed in ODL karaf.

`opendaylight-user@root>feature:list -i`: shows all installed features of ODL karaf.

5.2 ODL Deployment Clustering

5.2.1 Clustering Specifications

ODL clustering is using the AKKA technology which is compatible with the design of the MD-SAL. In order to deploy distributed environment, at least three nodes of ODL must be configured. ODL require at least three nodes in order to verify high-availability, however if in a 4-node cluster two of nodes crash, again this cluster is not functional. The clustering mechanism switches between nodes when the minimum number of nodes in a cluster is valid. The next table shows how many nodes must exist in a cluster [18].

Node number	Minimum number of servers must exist
2	2
3	2
4	3
5	3
6	3
7	4

Before setting the cluster, a brief description of ODL clustering mechanism will be described.

Shards: The MD-SAL datastore uses chunks to store data, in ODL word they are known as shards. Shard is a partition of data that can be stored either on one server or many servers. For example, one shard can contain all the inventory data while another shard contains all of the topology data. As a result, the data are stored in default shard unless, a specific shard configuration is done then, the data will be stored in a datastore regarding the shard configuration too. Shards configuration takes place in a *modules-shards.conf* file. This file allows to configure shards replicas for the clustering mechanism. A X-node cluster to be able to tolerate any single node crashing, a replica of every defined data shard must be running on all three cluster nodes.

Roles: Another detail that must be clarified is the *role*. Assuming that, a cluster consists of three nodes there must be a way to identify each node. Every node in a cluster must have unique identifier. ODL has introduced the concept of node *role*. In particular, the roles of nodes are defined as member-X depending on X number of nodes exist in a cluster. This configuration takes place in an *akka.conf* file. For example, if the nodes-1 role is defined as member-1, ODL recognizes the node-1 by the member-1.

To make a cluster operational multiple seed node must be configured. When a cluster member is started, it sends a message to all its seed nodes. Once the seed node (any of them) responds, the cluster member sends a join command to the first seed node that initiated the response. If none of the seed nodes respond, the cluster member repeats the process until it successfully establishes a connection with one of the seed nodes else, it remains shutdown. In case a any node fails for any reason, it needs to be restarted to be able active and take part in a cluster. When a node is restarted after any failure first it searches for lead node and then joins the cluster.

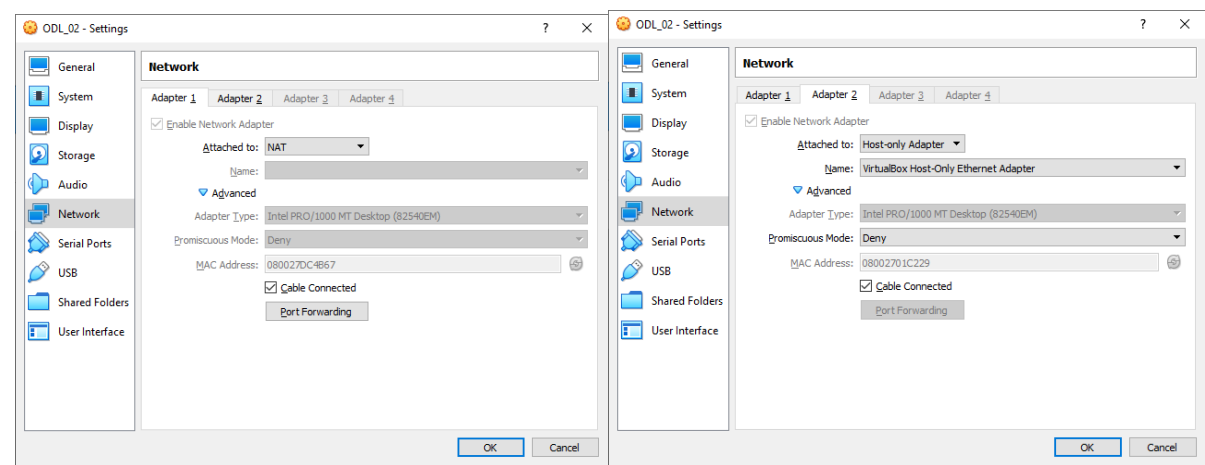
This means that for a particular shard you need to verify that member-1 is hosting (lead node) and the replica of this shard is stored on both member-2 and member-3 servers (seed node).

5.2.2 Clustering in Practice

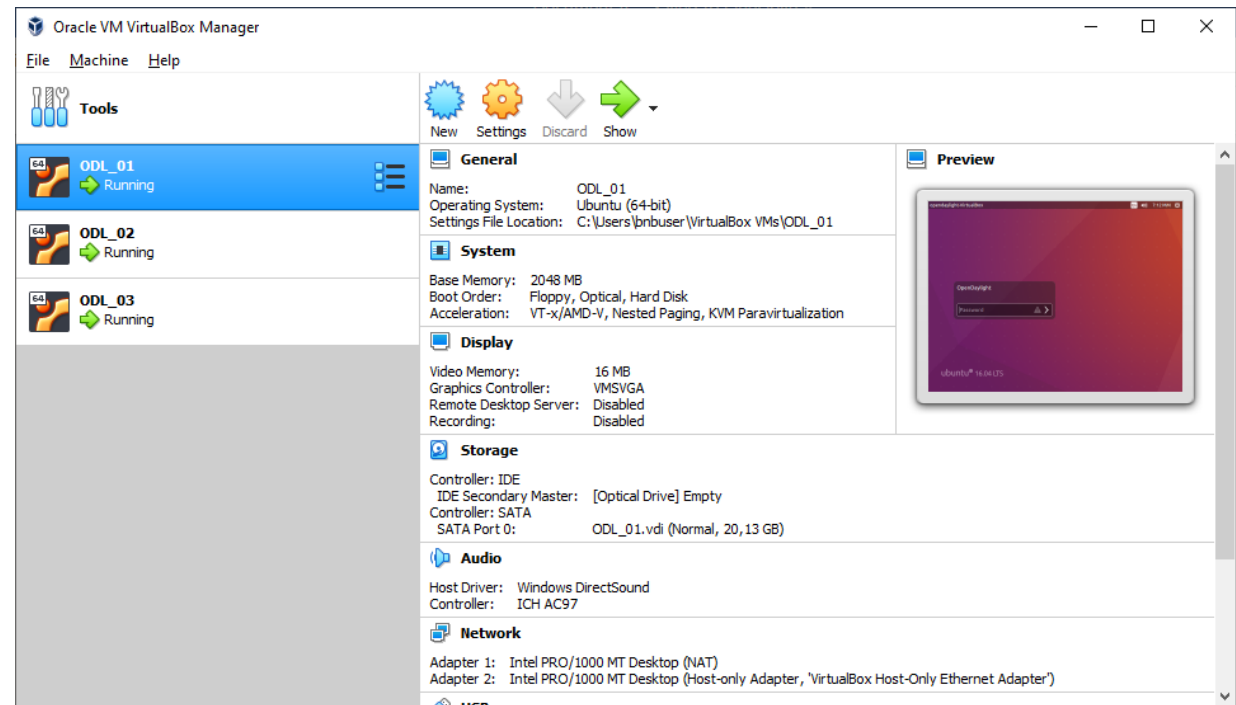
The clustering environment that will be demonstrated will contain three ODL nodes running in the same hypervisor as depicted in table.

Cluster Nodes	Virtual Machine Name	IP-Address	RAM (MB)	Hard Disk (GB)
ODL Server 1	ODL_01	192.168.56.101	2048	20
ODL Server 2	ODL_02	192.168.56.102	2048	20
ODL Server 3	ODL_03	192.168.56.103	2048	20

Network configurations adding the host only adapter to ODL_VMs



The following figure shows the configured ODL VMs for the clustering environment:



All ODL VMs are in a running state and each node has the ODL Oxygen SR4 release hosted on it.

There are several steps in order to manage an ODL cluster. For this purpose, ODL allows to configure a clustering with build-in scripts.

Step 1: This step defines which are the seed nodes and which are the lead node. As has been mentioned before on seed nodes replicas of data shards will be stored. In this step *akka.conf* and *module-shards.conf* files will be configured. In */home/opendaylight/karaf-0.8.4/bin* directory exist an executable file with a name *configure_cluster.sh*. This file allows to define the clustering parameters. **The command must be executed as following:**

```
./configure_cluster.sh <index> <seed_nodes_list>
```

Where

<index>: is the number that defines the seed nodes number. This indicates which controller node is configured by the script and

<seed_nodes_list>: defines the seed nodes IP addresses separated by comma.

The IP address at the provided index should belong to the member executing the script. When running this script on multiple seed nodes, keep the seed_node_list the same, and vary the index from 1 through N.

The next command shows an example of the aforementioned command

opendaylight@opendaylight-VirtualBox:~/karaf-0.8.4/bin\$./configure_cluster.sh 1 192.168.56.101 192.168.56.102 192.168.56.103

The above command will configure the member 1 (IP address 192.168.56.102) of a cluster made of 192.168.56.101 192.168.56.12 192.168.56.103.

```
opendaylight@opendaylight-VirtualBox:~/karaf-0.8.4/bin$ ./configure_cluster.sh 1 192.168.56.101 192.168.56.102 192.168.56.103
#####
##          Configure Cluster          ##
#####
Configuring unique name in akka.conf
Configuring hostname in akka.conf
Configuring data and rpc seed nodes in akka.conf
modules = [
    {
        name = "inventory"
        namespace = "urn:opendaylight:inventory"
        shard-strategy = "module"
    },
    {
        name = "topology"
        namespace = "urn:TBD:params:xml:ns:yang:network-topology"
        shard-strategy = "module"
    },
    {
        name = "toaster"
        namespace = "http://netconfcentral.org/ns/toaster"
        shard-strategy = "module"
    }
]
Configuring replication type in module-shards.conf
#####
## NOTE: Manually restart controller to ##
## apply configuration.                ##
#####
opendaylight@opendaylight-VirtualBox:~/karaf-0.8.4/bin$
```

Navigating the directories of the **/configuration/initial/** all configuration files for this ODL controller have been created.

```
opendaylight@opendaylight-VirtualBox:~/karaf-0.8.4/configuration/initial$ ll
total 20
drwxrwxr-x 2 opendaylight opendaylight 4096 Σεπ 12 21:49 ./
drwxr-xr-x 3 opendaylight opendaylight 4096 Σεπ 12 21:48 ../
-rw-r--r-- 1 opendaylight opendaylight 1438 Σεπ 12 21:49 akka.conf
-rw-r--r-- 1 opendaylight opendaylight 336 Σεπ 12 21:49 modules.conf
-rw-r--r-- 1 opendaylight opendaylight 555 Σεπ 12 21:49 module-shards.conf
opendaylight@opendaylight-VirtualBox:~/karaf-0.8.4/configuration/initial$
```

The **akka.conf** file verifies that the applied configuration by the script has been set. The next figure shows that the IP address of ODL node is 192.168.56.101 (**netty.tcp** field). The **seed-nodes** field indicates that ODL controllers join the cluster are these that defined running the script file. Finally, the ODL_01 is assigned to “member-1” role.

```

-rw-r--r-- 1 opendaylight opendaylight 336 Σεπ 12 23:23 modules.conf
-rw-r--r-- 1 opendaylight opendaylight 555 Σεπ 12 23:23 module-shards.conf
opendaylight@opendaylight-VirtualBox:~/karaf-0.8.4/configuration/initial$ cat akka.conf

odl-cluster-data {
  akka {
    remote {
      artery {
        enabled = off
        canonical.hostname = "192.168.56.101"
        canonical.port = 2550
      }
      netty.tcp {
        hostname = "192.168.56.101"
        port = 2550
      }
      # when under load we might trip a false positive on the failure detector
      # transport-failure-detector {
      #   heartbeat-interval = 4 s
      #   acceptable-heartbeat-pause = 16s
      # }
    }
  }

  cluster {
    # Remove ".tcp" when using artery.
    seed-nodes = ["akka.tcp://opendaylight-cluster-data@192.168.56.101:2550",
                  "akka.tcp://opendaylight-cluster-data@192.168.56.102:2550",
                  "akka.tcp://opendaylight-cluster-data@192.168.56.103:2550"]
  }

  roles = ["member-1"]
}

persistence {
  # By default the snapshots/journal directories live in KARAF_HOME. You can choose to
  # put it somewhere else by

```

Checking the next file **modules-shards.conf** assignment for the replicas has been set.

```

opendaylight@opendaylight-VirtualBox:~/karaf-0.8.4/configuration/initial$ cat module-shards.conf
module-shards = [
  {
    name = "default"
    shards = [
      {
        name = "default"
        replicas = ["member-1",
                   "member-2",
                   "member-3"]
      }
    ]
  },
  {
    name = "inventory"
    shards = [
      {
        name="inventory"
        replicas = ["member-1",
                   "member-2",
                   "member-3"]
      }
    ]
  },
  {
    name = "topology"
    shards = [
      {
        name="topology"
        replicas = ["member-1",
                   "member-2",
                   "member-3"]
      }
    ]
  },
  {
    name = "toaster"
    shards = [
      {
        name="toaster"
        replicas = ["member-1",
                   "member-2",
                   "member-3"]
      }
    ]
  }
]

```

The next configuration is to run the ODL instances on every VM with the following command:

```
opendaylight@opendaylight-VirtualBox:~/karaf-0.8.4$ JAVA_MAX_MEM=4G JAVA_MAX_PERM_MEM=512m
./bin/karaf
```

And the final command is to install odl-mdsal-clustering.

```
opendaylight-user@root>feature:install odl-mdsal-clustering
```

[illegible]

Same configuration must be applied for the rest of ODL_01 and ODL_02 servers changing just the index parameter.

Using the ODL CLI command `opendaylight-user@root>log:tail` the logging messages verify that a candidate node can be a lead node and backward.

Finally, if an ODL_01 node will be crashed in the cluster, the logs of any of the rest ODL nodes will notify that the ODL_01 node is not running. However, when this node will be recovered it immediately became again a candidate node.

```

- com.typesafe.akka.slf4j - 2.5.11 | Association with remote system [akka.tcp://opendaylight-cluster-data@192.168.56.101:2550] has failed, address is now gated for [5000] ms. Reason: [Association failed with [akka.tcp://opendaylight-cluster-data@192.168.56.101:2550]] Caused by: [Connection refused: /192.168.56.101:2550]
2019-09-12 23:56:08,264 | INFO | lt-dispatcher-18 | akka://opendaylight-cluster-data | 68
- com.typesafe.akka.slf4j - 2.5.11 | Cluster Node [akka.tcp://opendaylight-cluster-data@192.168.56.102:2550] - Leader can currently not perform its duties, reachability status: [akka.tcp://opendaylight-cluster-data@192.168.56.102:2550 -> akka.tcp://opendaylight-cluster-data@192.168.56.101:2550: Unreachable [Unreachable] (1), akka.tcp://opendaylight-cluster-data@192.168.56.103:2550 -> akka.tcp://opendaylight-cluster-data@192.168.56.101:2550: Unreachable [Unreachable] (1)], member status: [akka.tcp://opendaylight-cluster-data@192.168.56.101:2550 Up seen=false, akka.tcp://opendaylight-cluster-data@192.168.56.102:2550 Up seen=true, akka.tcp://opendaylight-cluster-data@192.168.56.103:2550 Up seen=true]
2019-09-12 23:56:10,459 | WARN | lt-dispatcher-30 | NettyTransport | 68
- com.typesafe.akka.slf4j - 2.5.11 | Remote connection to [null] failed with java.net.ConnectException: Connection refused: /192.168.56.101:2550
2019-09-12 23:56:10,460 | WARN | lt-dispatcher-30 | ReliableDeliverySupervisor | 68
- com.typesafe.akka.slf4j - 2.5.11 | Association with remote system [akka.tcp://opendaylight-cluster-data@192.168.56.101:2550] has failed, address is now gated for [5000] ms. Reason: [Association failed with [akka.tcp://opendaylight-cluster-data@192.168.56.101:2550]] Caused by: [Connection refused: /192.168.56.101:2550]
2019-09-12 23:56:15,661 | WARN | lt-dispatcher-18 | NettyTransport | 68
- com.typesafe.akka.slf4j - 2.5.11 | Remote connection to [null] failed with java.net.ConnectException: Connection refused: /192.168.56.101:2550
2019-09-12 23:56:15,663 | WARN | lt-dispatcher-18 | ReliableDeliverySupervisor | 68
- com.typesafe.akka.slf4j - 2.5.11 | Association with remote system [akka.tcp://opendaylight-cluster-data@192.168.56.101:2550] has failed, address is now gated for [5000] ms. Reason: [Association failed with [akka.tcp://opendaylight-cluster-data@192.168.56.101:2550]] Caused by: [Connection refused: /192.168.56.101:2550]
2019-09-12 23:56:20,794 | WARN | ult-dispatcher-3 | NettyTransport | 68
- com.typesafe.akka.slf4j - 2.5.11 | Remote connection to [null] failed with java.net.ConnectException: Connection refused: /192.168.56.101:2550
2019-09-12 23:56:20,797 | WARN | ult-dispatcher-3 | ReliableDeliverySupervisor | 68
- com.typesafe.akka.slf4j - 2.5.11 | Association with remote system [akka.tcp://opendaylight-cluster-data@192.168.56.101:2550] has failed, address is now gated for [5000] ms. Reason: [Association failed with [akka.tcp://opendaylight-cluster-data@192.168.56.101:2550]] Caused by: [Connection refused: /192.168.56.101:2550]

```

6 Mininet

Mininet is an open source tool that creates virtual network environment with one single command. In particular it is used for creating the data plane elements (switches) for the SDN environment. Mininet networks usually are a composition of hosts, switches, routes, controllers and links with the last being represented as virtual ethernet connections. Mininet not only creates a network but it also allows to configure it and test it. Using Mininet it is possible to develop a network based on a single GNU/Linux kernel [19].

6.1 Why to use Mininet

Mininet is the best choice to simulate virtual networks because it is compatible with many SDN controllers and switches. Also, it comes with built-in SDN switch Open vSwitch that supports the OpenFlow protocol and many other utilities that will be presented afterwards. Also, it allows easy to create custom topologies based on Python language. Furthermore, the command line interface (CLI) allows to test and configure the network topologies with real conditions, such as setting up link bandwidth, link delay, and loss characteristics. Finally, it supports the *miniedit* GUI (Graphical User Interface) to create a network topology [19].

6.2 Mininet-Deployment

In order to set up a Mininet tool that will be used to create virtual network topology, a virtualization platform is required. However, the Mininet tool can be used without being hosted in any hypervisor. There are three options to install the Mininet tool [19]:

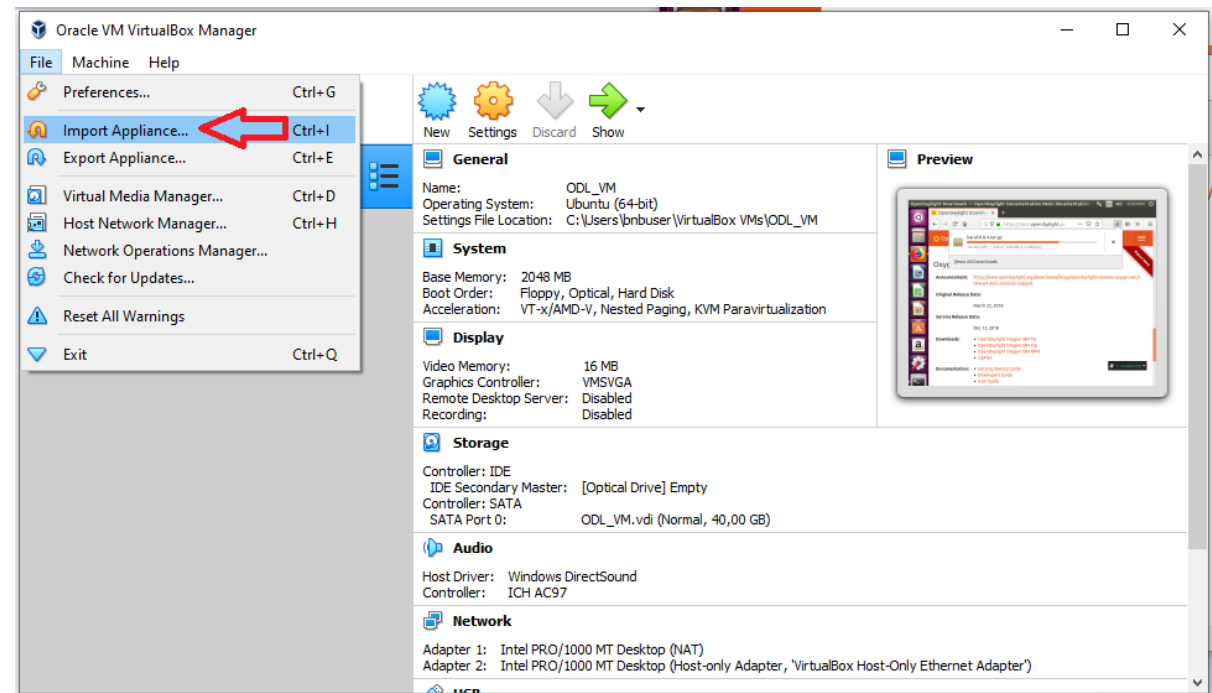
- Installing Mininet VM in hypervisor which is the recommended
- Native Installation from Source
- Installing Mininet from packages

6.2.1 Installing Mininet VM

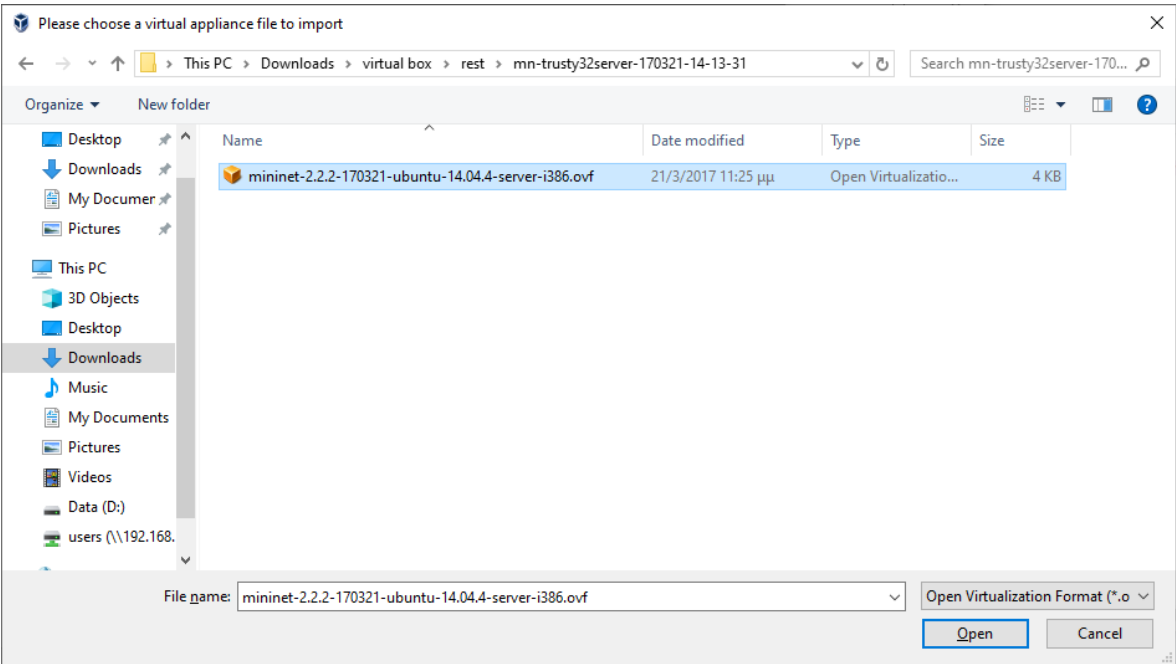
The required VM images are downloaded from [19] in order to set up the Mininet VM. The retrieved zip file contains two files as demonstrated in following figure.

Next Mininet Deployment in VirtualBox hypervisor will be presented.

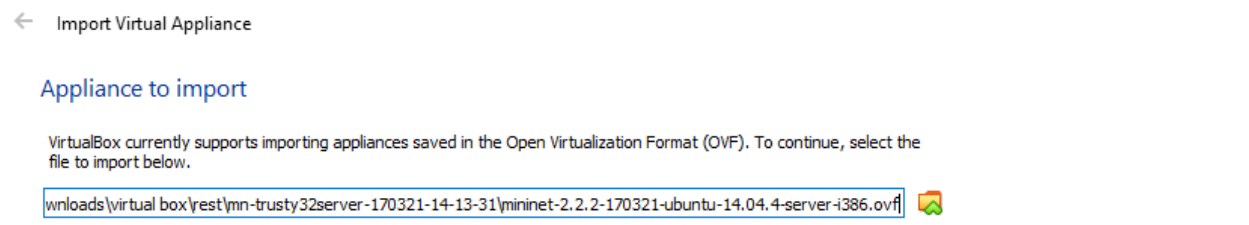
Step 1. Since the VirtualBox is in running state selecting from File-> Import Appliance a new wizard pops-up to browse an appliance.



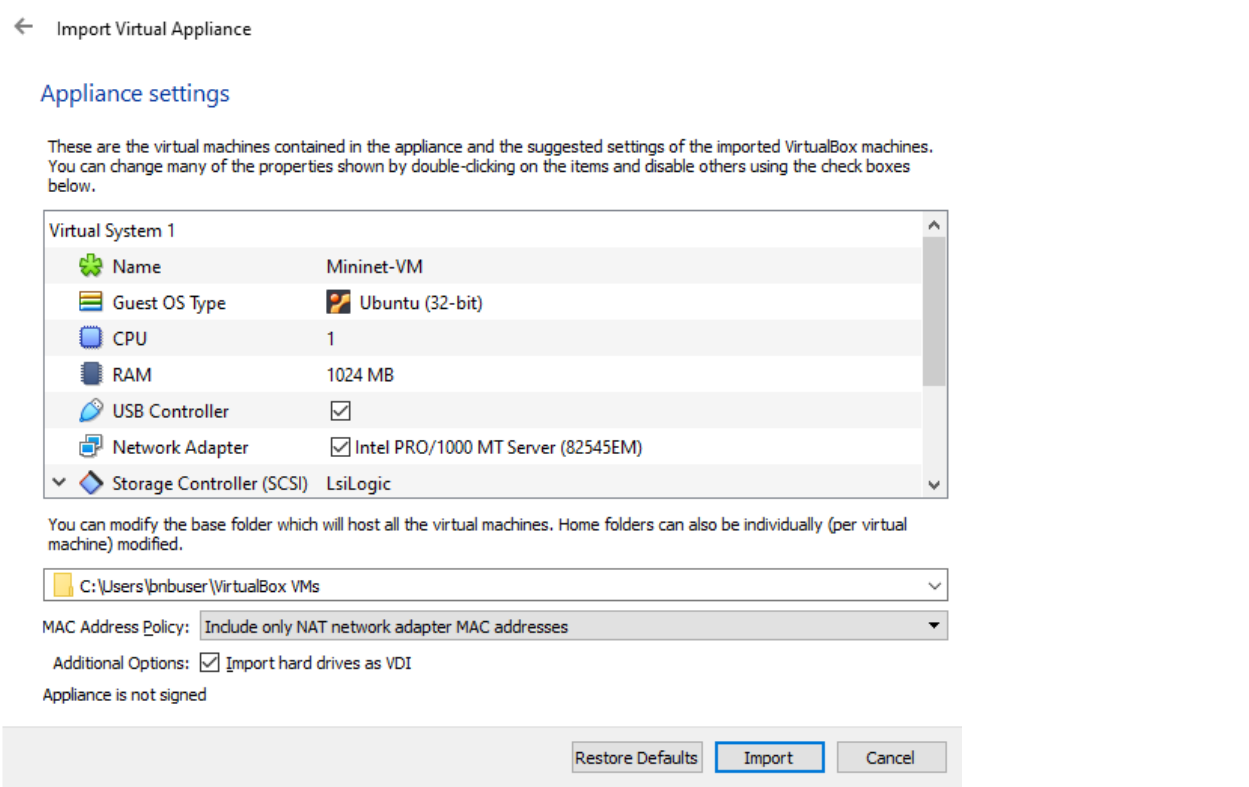
Step 2. The next wizard browses the .ovf (Open Virtualization Format) image.



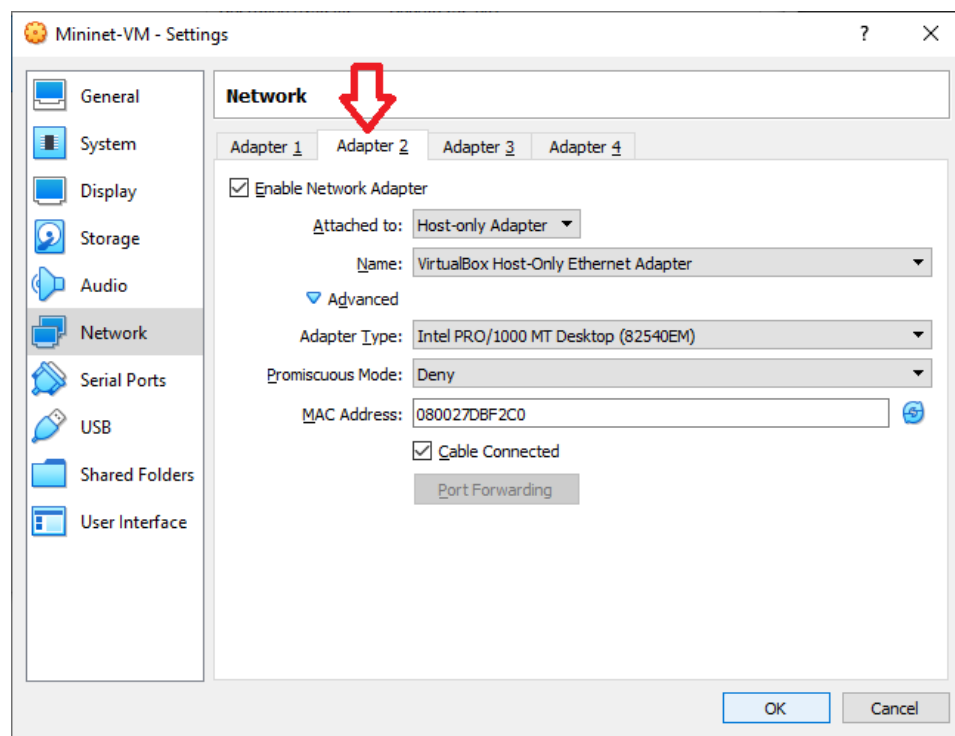
This will unpack and import the VM in your local machine. It will take a while, as the unpacked image is about 3 GB. Once the Mininet VM is completed, Mininet version 2.2.2 is installed along with Wireshark, Openflow13, Open vSwitch, a POX and NOX SDN controllers and other useful utilities on Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64) operating system.



Step 3. This step shows the assigned memory to 1GB RAM, 1 CPU and OS type Ubuntu for the VM. The provided values are sufficient to assure that the Mininet tool will work effectively.



Step 4. Similar to ODL configuration Host-only Adapter is configured.



When the Mininet VM is running it starts booting and throws a login prompt, providing the default credentials for username and password “*mininet*” allows access to It. This user is a sudoer, as a result, root permissions are available for any command. It is worth to mention, that this VM does not include Graphical User Interface, so the built-in X server of the host machine will be used to solve this problem.

```
Ubuntu 14.04.4 LTS mininet-vm tty1
mininet-vm login: mininet
Password:
Last login: Tue Mar 21 21:13:43 PDT 2017 on ttyS0
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic i686)

* Documentation:  https://help.ubuntu.com/
```

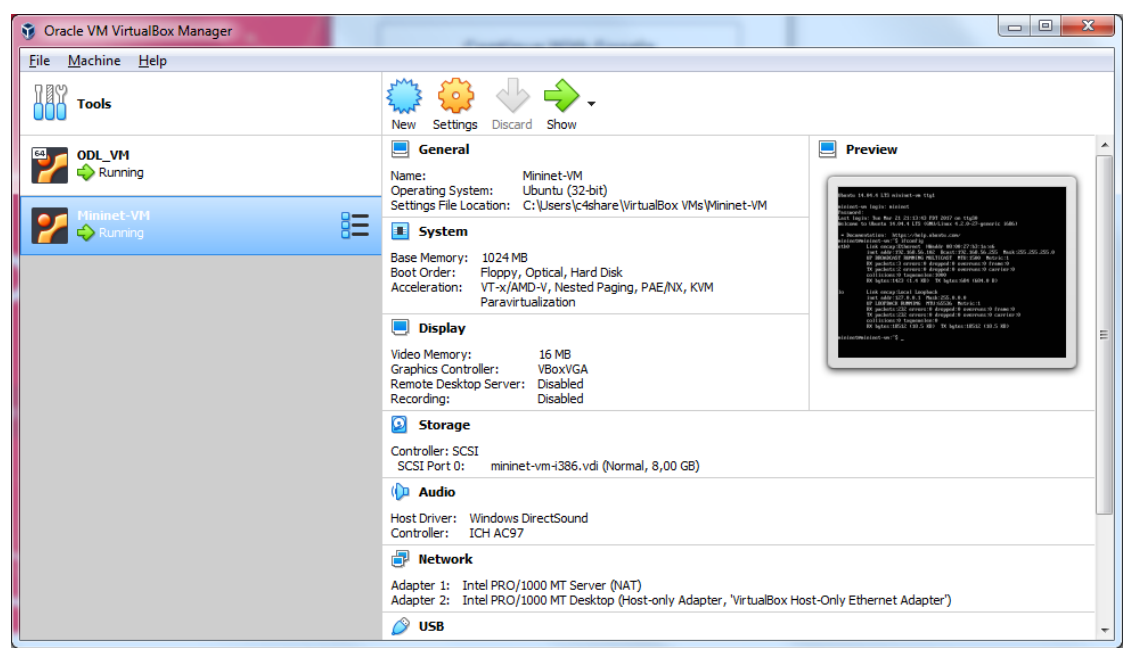
Next, the IP address is retrieved in order to set up the X forwarding. The next figure shows the eth0 interface with IP address 192.168.56.102, which means this is the Mininet VM IP address is 192.168.56.102 and there is access to it via SSH (Secure Shell).

```
Ubuntu 14.04.4 LTS mininet-vm tty1
mininet-vm login: mininet
Password:
Last login: Tue Mar 21 21:13:43 PDT 2017 on ttyS0
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic i686)

* Documentation:  https://help.ubuntu.com/
mininet@mininet-vm:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:90:f8:5c
          inet addr: 192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:21 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6975 (6.9 KB)  TX bytes:684 (684.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:104 errors:0 dropped:0 overruns:0 frame:0
          TX packets:104 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:8312 (8.3 KB)  TX bytes:8312 (8.3 KB)

mininet@mininet-vm:~$
```



From now on, all actions that will take place in this VM will be accomplished after SSH (Secure Shell) connection and forwarding the X server from the host machine terminal as shown from figure bellow.

```
sdn@sdn-opendaylight:~$ ssh -X mininet@192.168.56.102
The authenticity of host '192.168.56.102 (192.168.56.102)' can't be established.
ECDSA key fingerprint is SHA256:NGco/tf+bXePQQT4AqoCLIM090SuWE2Vj7FTogGXbHg.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.56.102' (ECDSA) to the list of known hosts.
mininet@192.168.56.102's password:
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic i686)

 * Documentation:  https://help.ubuntu.com/
Last login: Mon Sep 23 20:36:04 2019
/usr/bin/xauth: file /home/mininet/.Xauthority does not exist
mininet@mininet-vm:~$
```

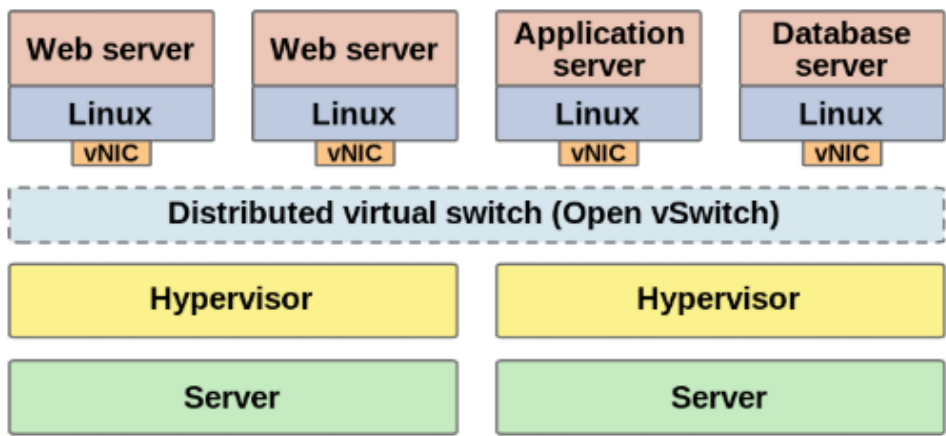
6.3 Mininet build-in Tools

Mininet VM has already pre-installed many tools networking presented next:

- **Mininet:** command line tool, creates a virtual network that is composed of controller, virtual switches, hosts, and links.
- **Open vSwitch (OVS):** Is a virtual OpenFlow-enabled switch and it is used in many open source and commercial networks and virtualization platforms. It was implemented by Nicira company. OVS in based on Linux Kernel Module. It supports different technologies and protocols, such as 802.1Q, BFD, NetFlow, sFlow, port mirroring, VLANs, LACP, VXLAN, GENEVE GRE Overlays, STP, and IPv6. Virtual Ethernet ports pair are used in order to connect hosts by OVS. Virtual Ethernet ports are equivalent to a pair of physical Ethernet interfaces interconnected by a cable however, they are implemented using software. The virtual port connection is implemented at a link layer. OVS works like a regular MAC learning and forwarding switch when no controller is configured and OpenFlow rules are not programmed (standalone). It programs the OpenFlow flow tables when it receives inputs from the SDN OpenFlow-enabled controller. It also supports the OVSDb southbound protocol. OVS is always layered below the OpenFlow interface. The release of the switch that is hosted in Mininet -VM is 2.0.2 The next figure verifies this version.

```
mininet@mininet-vm:~$ ovs-dpctl -V
ovs-dpctl (Open vSwitch) 2.0.2
Compiled Dec  9 2015 14:08:08
mininet@mininet-vm:~$
```

Figure illustrates the structure of the OVS:



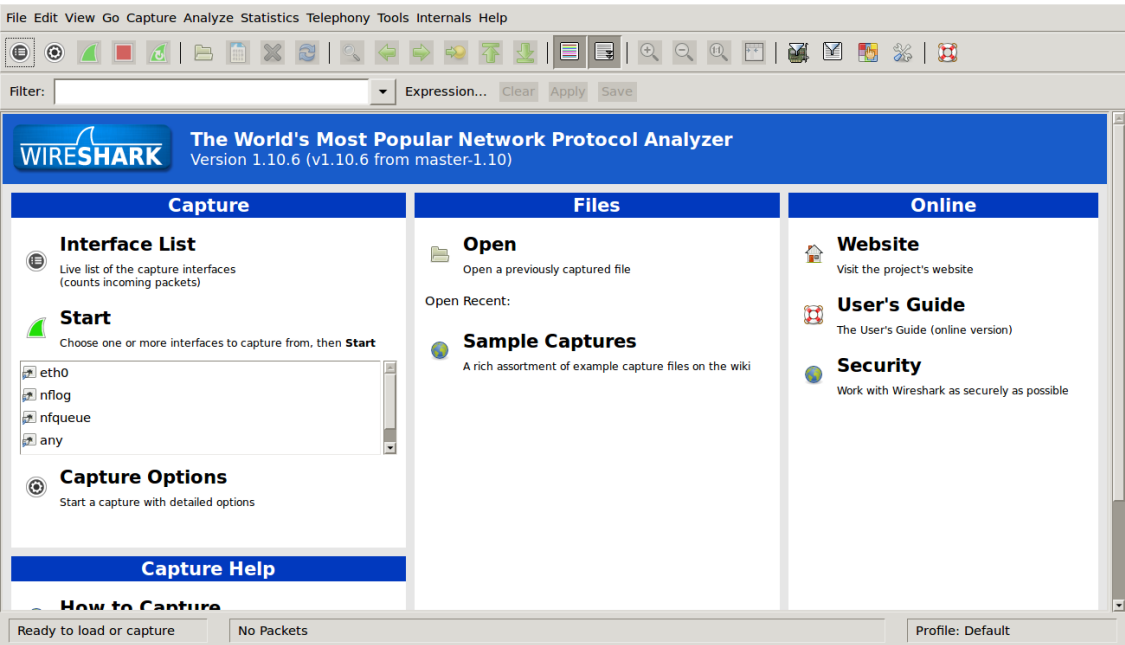
- **POX Controller:** Is a built-in OpenFlow controller, but can also function as an OpenFlow switch that resides in Mininet VM. In terms of this research the ODL controller will be used. Every OpenFlow controller is located above of the OpenFlow interface. The controller communicated with the switch with the OpenFlow protocol.
- **dpctl:** Is a command line tool that configures flow tables in OpenFlow switch. It allows to adds flows, modifies the flows, queries for switch features and status [20].
- **ovs-ofctl:** command line utility that sends quick OpenFlow messages, useful for viewing switch port and flow stats or manually inserting flow entries.
- **ovs-vsctl:** command line utility that allows queries and configuration on ovs-vswitchd which is an OVS daemon.# TODO the rest of commands
- **Wireshark:** Tool with GUI for analyzing the packets. In particular it will dissect OpenFlow packets.

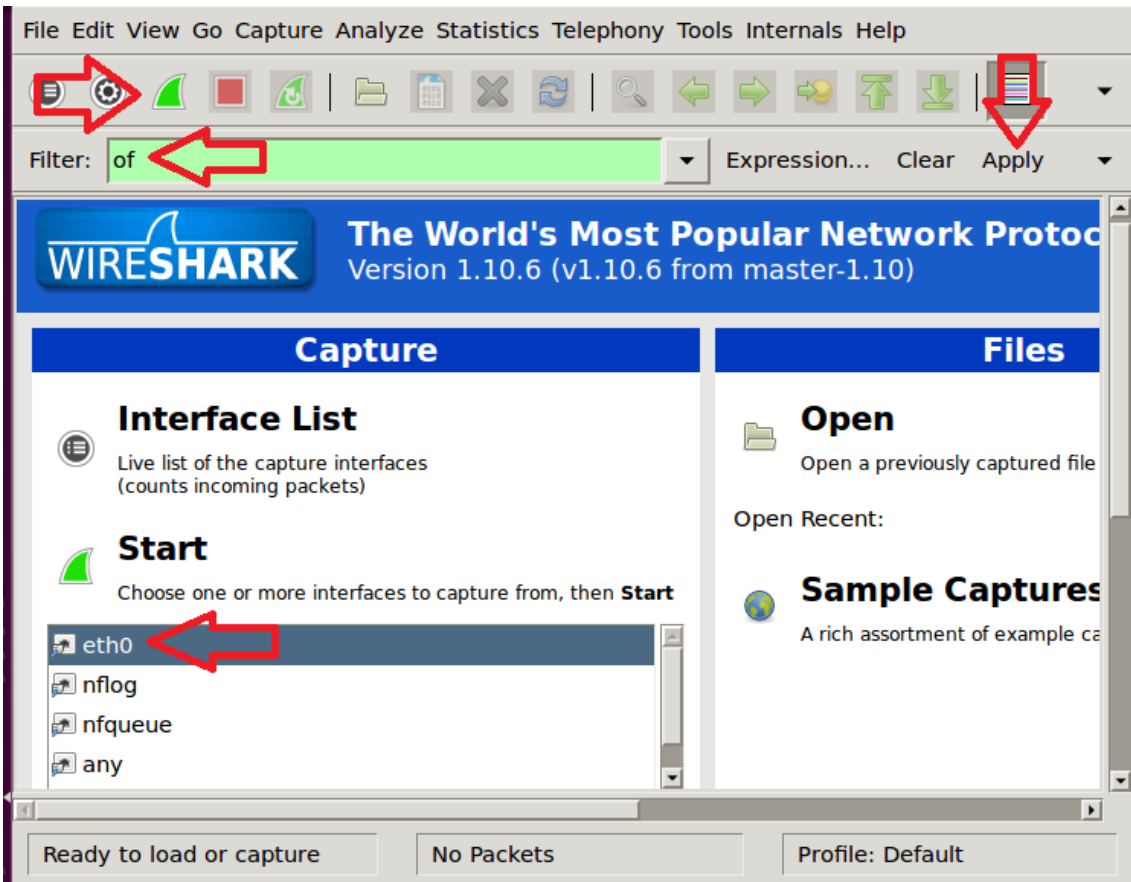
Before starting the Wireshark, capture privileges and permissions to specific files /usr/bin/dumpcap directory must be set by following commands

```
mininet@mininet-vm:~$ sudo chgrp mininet /usr/bin/dumpcap
mininet@mininet-vm:~$ sudo chmod 754 /usr/bin/dumpcap
mininet@mininet-vm:~$ sudo setcap 'CAP_NET_RAW+eip CAP_NET_ADMIN+eip' /usr/bin/dumpcap
```

The bellow command starts the Wireshark:

```
mininet@mininet-vm:~$ wireshark
```





- **iperf**: Tool for testing the speed of hosts.
- **cbench**: Cbench is a software for testing OpenFlow controllers by generating packet-in events for new flows. Cbench emulates a bunch of switches, which connect to a controller, sends packet-in messages, and waits for flow-mods to get pushed down.

7 Integration of Mininet with OpenDaylight Controller

In this chapter, will present how to build a virtual SDN lab using ODL and Mininet. Mininet is a tool for virtualizing OVS-based virtual switches and Linux container hosts. ODL and Mininet communicate with each other and how hosts in a virtual lab can ping each other by leveraging the SDN controller to program the flows inside the switches. In order to create the virtual network, the Mininet VM and the Opendaylight controller must be in running state. Connect to Mininet VM with default username:mininet and password:mininet

The ODL controller must be in running state. In addition, L2Switch feature must be installed, along with DLUX Web Interface and restconf API.

The next steps verify that required ODL modules are installed and are accessible. Also verify that Mininet VM is connected to ODL via OpenFlow.

- *Running ODL*

The following command starts the ODL container.

```
sdn@sdn-opensaylight:~/Downloads/karaf-0.8.4$ ./bin/karaf
```

```
sdn@sdn-opendaylight:~/karaf-0.8.4$ ./bin/karaf
Apache Karaf starting up. Press Enter to open the shell now...
100% [=====]

Karaf started in 0s. Bundle stats: 13 active, 13 total

      _____
     /         \
    /           \
   /             \
  /               \
 /                 \
/                   \
 \                   /
  \                 /
   \               /
    \             /
     \           /
      \         /
       \       /
        \     /
         \   /
          \ /
           /
          /
         /
        /
       /
      /
     /
    /
   /
  /
 /
/

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
```

Since the controller is in running state, it allows to install any ODL module.

- *Enabling L2Switch*

In order to make Mininet to connect with the Opendaylight we need to install the l2switch feature.

After installing the I2switch in OpenDaylight karaf, the port 6633 will be activated to accept incoming TPC/TLS connections with mininet. Also, there is another TCP/TLS port which is 6653 and establishes a secure channel connection with the Opendayligh and Mininet.

```

OpenDaylight-user@root>feature:install odl-l2switch-switch
OpenDaylight-user@root>feature:list | grep l2switch
odl-l2switch-all | 0.7.4 | | Uninstalled | odl-l2switch-all
  OpenDaylight :: L2Switch :: All | 0.7.4 | |
odl-l2switch-packethandler | 0.7.4 | | Started | odl-l2switch-packethandler
  OpenDaylight :: L2Switch :: PacketHandler | 0.7.4 | |
odl-l2switch-switch | 0.7.4 | x | Started | odl-l2switch-switch
  OpenDaylight :: L2Switch :: Switch | 0.7.4 | |
odl-l2switch-switch-ui | 0.7.4 | | Uninstalled | odl-l2switch-switch-ui
  OpenDaylight :: L2Switch :: Switch UI | 0.7.4 | |
odl-l2switch-addresstracker | 0.7.4 | | Started | odl-l2switch-addresstracker
  OpenDaylight :: L2Switch :: AddressTracker | 0.7.4 | |
odl-l2switch-loopremover | 0.7.4 | | Started | odl-l2switch-loopremover
  OpenDaylight :: L2Switch :: LoopRemover | 0.7.4 | |
features-l2switch | 0.7.4 | | Uninstalled | features-l2switch
  features-l2switch | 0.7.4 | |
odl-l2switch-arphandler | 0.7.4 | | Started | odl-l2switch-arphandler
  OpenDaylight :: L2Switch :: ArpHandler | 0.7.4 | |
odl-l2switch-hosttracker | 0.7.4 | | Started | odl-l2switch-hosttracker
  OpenDaylight :: L2Switch :: HostTracker | 0.7.4 | |
odl-l2switch-switch-rest | 0.7.4 | | Uninstalled | odl-l2switch-switch-rest
  OpenDaylight :: L2Switch :: Switch REST | 0.7.4 | |
OpenDaylight-user@root>

```

```
opendaylight-user@root>feature:list -i | grep openflow
odl-openflowjava-protocol | 0.6.4 | | Started |
odl-openflowjava-0.6.4 | ODL :: openflowjava :: odl-openflowjava-protocol
odl-openflowplugin-app-arbitratorreconciliation | 0.6.4 | | Started |
odl-openflowplugin-app-arbitratorreconciliation | OpenDaylight :: Openflow Plugin :: Application
odl-openflowplugin-nsf-model | 0.6.4 | | Started |
odl-openflowplugin-nsf-model | OpenDaylight :: OpenflowPlugin :: NSF :: Model
odl-openflowplugin-southbound | 0.6.4 | | Started |
odl-openflowplugin-0.6.4 | OpenDaylight :: Openflow Plugin :: Southbound
odl-openflowplugin-libraries | 0.6.4 | | Started |
odl-openflowplugin-libraries | OpenDaylight :: Openflow Plugin :: Libraries
odl-openflowplugin-app-reconciliation-framework | 0.6.4 | | Started |
odl-openflowplugin-app-reconciliation-framework | OpenDaylight :: Openflow Plugin :: Application
odl-openflowplugin-flow-services | 0.6.4 | | Started |
odl-openflowplugin-flow-services | OpenDaylight :: Openflow Plugin :: Flow Services
odl-openflowplugin-app-topology | 0.6.4 | | Started |
odl-openflowplugin-app-topology | OpenDaylight :: Openflow Plugin :: Application
odl-openflowplugin-app-forwardingrules-manager | 0.6.4 | | Started |
odl-openflowplugin-app-forwardingrules-manager | OpenDaylight :: Openflow Plugin :: Application
odl-openflowplugin-app-config-pusher | 0.6.4 | | Started |
odl-openflowplugin-app-config-pusher | OpenDaylight :: Openflow Plugin :: Application
opendaylight-user@root>
```

```
4 - org.eclipse.jetty.util - 9.3.24.v20180605 | Started HttpServiceContext{httpContext=WebAppHttpContext{org.opendaylight.netconf.restconf-nb-bierman02 - 297}}
2019-09-19 02:37:05,713 | INFO | Thread-40 | TcpHandler | 41
l - org.opendaylight.openflowplugin.openflowjava.openflow-protocol-impl - 0.6.4 | Switch l
stener started and ready to accept incoming tcp/tls connections on port: 6633 ←
2019-09-19 02:37:05,725 | INFO | Thread-39 | OpenFlowPluginProviderImpl | 40
l - org.opendaylight.openflowplugin.impl - 0.6.4 | All switchConnectionProviders are up an
l running (2).
2019-09-19 02:37:05,726 | INFO | Thread-39 | TcpHandler | 41
l - org.opendaylight.openflowplugin.openflowjava.openflow-protocol-impl - 0.6.4 | Switch l
stener started and ready to accept incoming tcp/tls connections on port: 6653 ←
```

After installing the odl-l2switch-switch, verify that the ports 6653 and 6633 are activated in order to receive calls from Mininet.

```
sdn@sdn-opendaylight:~$ ps -eaf | grep ":6633"
```

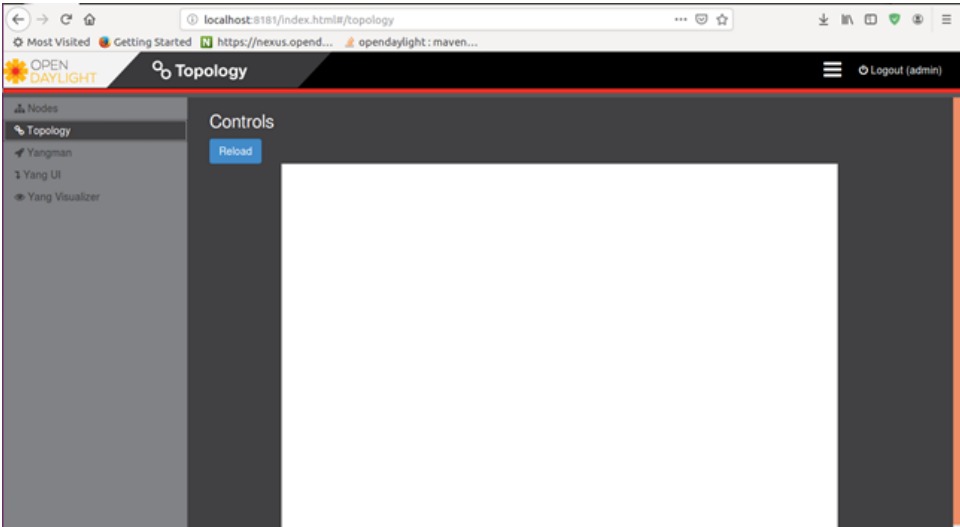
```
sdn@sdn-opendaylight:~$ ps -eaf | grep ":6633"
sdn      3803  3061  0 17:32 pts/6    00:00:00 grep --color=auto :6633
sdn@sdn-opendaylight:~$ ps -eaf | grep ":6653"
sdn      3809  3061  0 17:33 pts/6    00:00:00 grep --color=auto :6653
sdn@sdn-opendaylight:~$
```

- Enabling DLUX web interface

Enable the DLUX web interface through command.

```
opendaylight-user@root>feature:install features-dlux
```

The “Started” annotation indicates that dlux modules have been started successfully. The next figure verifies that the DLUX Web interface is accessible.



- *Enabling restconf A PI*

To access the REST ODL apply navigate to <http://localhost:8181/apidoc/explorer/index.html>

localhost:8181/apidoc/explorer/index.html

https://nexus.opend...

opendaylight: maven...

OpenDaylight RestConf API Documentation

Controller Resources

Mounted Resources

Below are the list of APIs supported by the Controller.

aaa(2016-12-14)	Show/Hide	List Operations	Expand Operations	Raw
aaa-app-config(2017-06-19)	Show/Hide	List Operations	Expand Operations	Raw
aaa-cert(2015-11-26)	Show/Hide	List Operations	Expand Operations	Raw
aaa-cert-mdsal(2016-03-21)	Show/Hide	List Operations	Expand Operations	Raw
aaa-cert-rpc(2015-12-15)	Show/Hide	List Operations	Expand Operations	Raw
aaa-encrypt-service-config(2016-09-15)	Show/Hide	List Operations	Expand Operations	Raw
address-tracker-config(2016-06-21)	Show/Hide	List Operations	Expand Operations	Raw
arbitrator-reconcile(2018-02-27)	Show/Hide	List Operations	Expand Operations	Raw
arp-handler-config(2014-05-28)	Show/Hide	List Operations	Expand Operations	Raw
cluster-admin(2015-10-13)	Show/Hide	List Operations	Expand Operations	Raw
config(2013-04-05)	Show/Hide	List Operations	Expand Operations	Raw
entity-owners(2015-08-04)	Show/Hide	List Operations	Expand Operations	Raw
flow-capable-transaction(2015-03-04)	Show/Hide	List Operations	Expand Operations	Raw
flow-topology-discovery(2013-08-19)	Show/Hide	List Operations	Expand Operations	Raw
forwarding-rules-manager-config(2016-05-11)	Show/Hide	List Operations	Expand Operations	Raw
frm-reconciliation(2018-02-27)	Show/Hide	List Operations	Expand Operations	Raw
general-entity(2015-08-20)	Show/Hide	List Operations	Expand Operations	Raw
host-tracker-config(2014-05-28)	Show/Hide	List Operations	Expand Operations	Raw
ietf-access-control-list(2016-02-18)	Show/Hide	List Operations	Expand Operations	Raw
ietf-interfaces(2014-05-08)	Show/Hide	List Operations	Expand Operations	Raw

From here on, a virtual network topology can be created and every switch will be defined as I2-learning-enabled switch. The controller is responsible to handle the forwarding rules of tables.

In order to create a virtual network in the VM, the following command.

```
mininet@mininet-vm:~$ sudo mn --topo=linear,3 --mac --controller=remote,ip=192.168.56.102 --switch ovsk,protocols=OpenFlow13
```

where

mac: will assign MAC address for every host equal to its IP address e.g. 00:00:00:00:00:01

ip: defines the IP address of the remote controller.

controller: is the IP address of the ODL controller where the virtual switches are connected,

topo: linear defines a network topology with three switches and three hosts, switches and hosts are connected with a virtual ethernet cable.

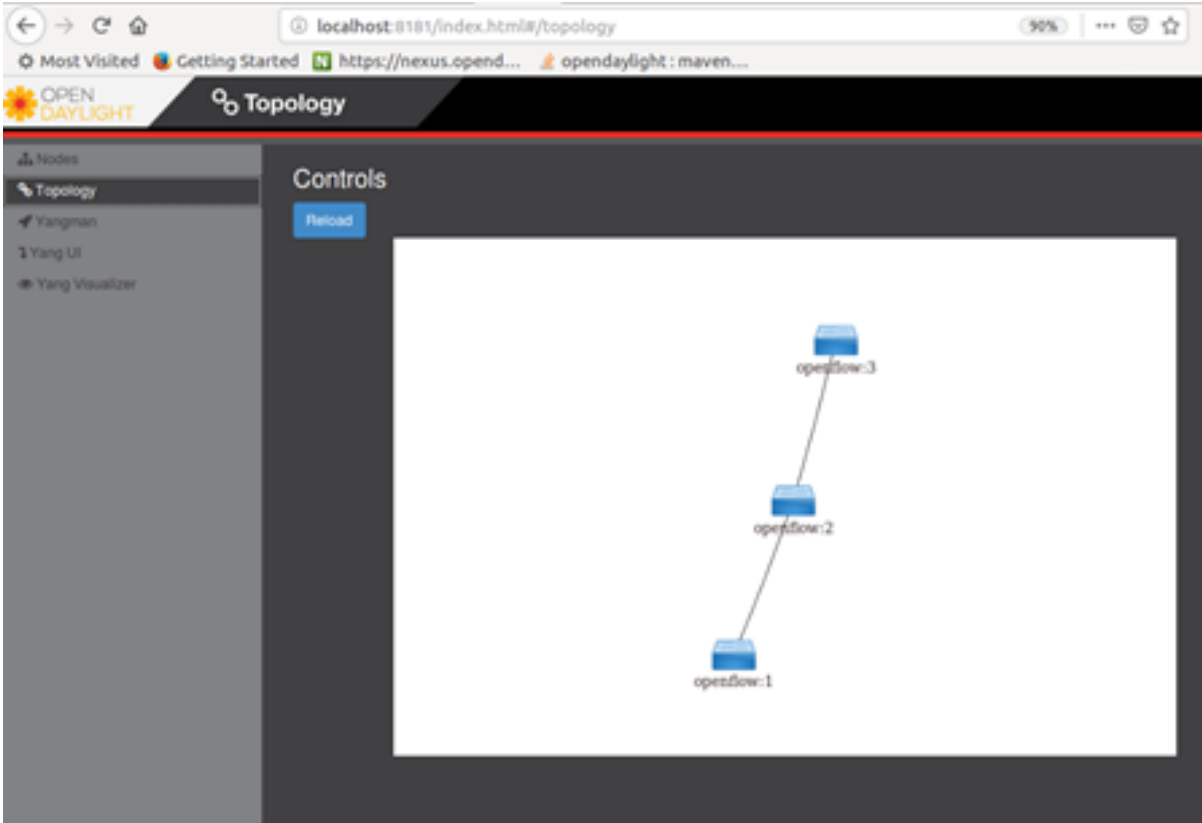
switch: is a parameter to identify the switch type in this case is an OpenFlow-enabled Open vSwitch.

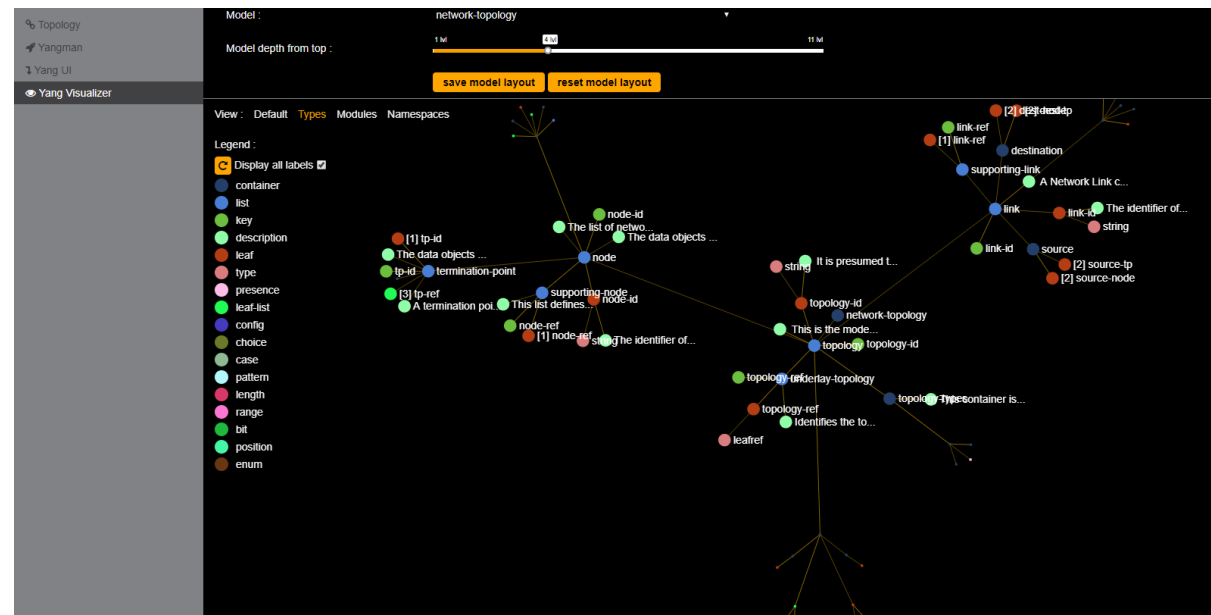
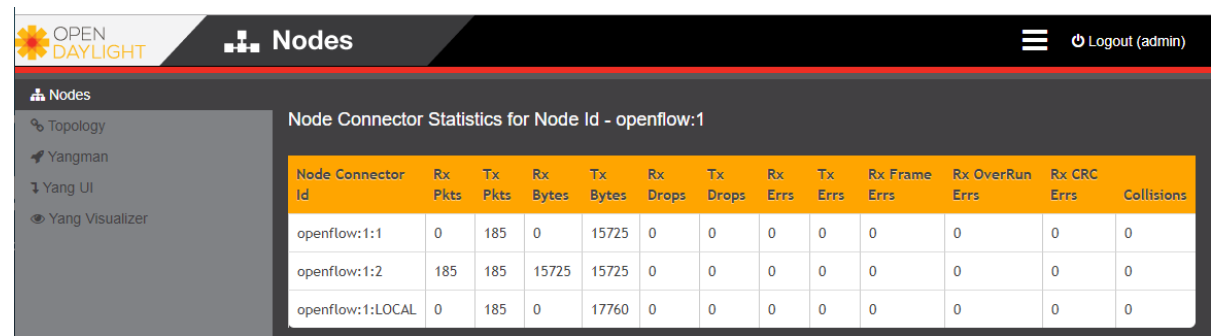
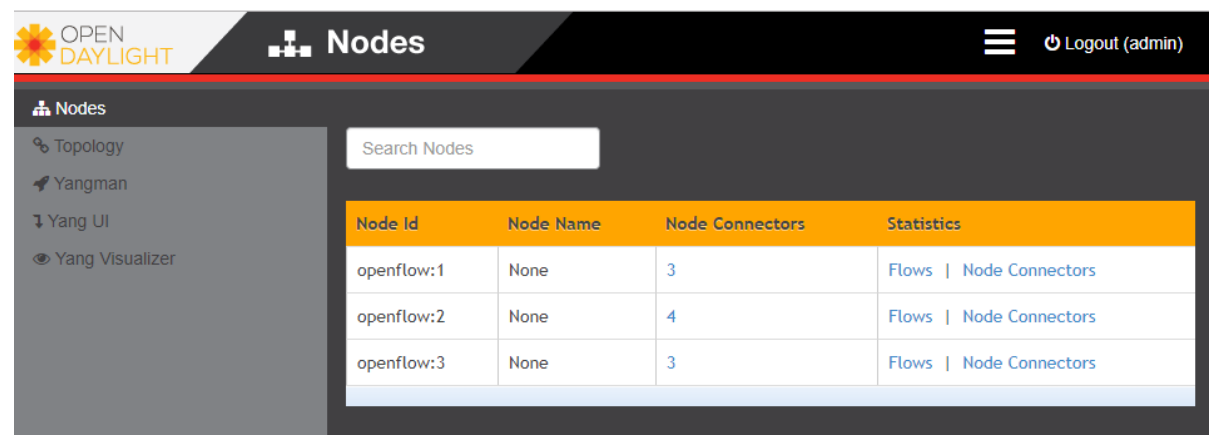
protocols: which is set to OpenFlow13 means that this switch is compatible with this protocol. The next figure shows the result of this command.

```
mininet@mininet-vm:~$ sudo mn --topo=linear,3 --mac --controller=remote,ip=192.168.56.101
--switch ovsk,protocols=OpenFlow13
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.56.101:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
```

```
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> 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> nodes
available nodes are:
c0 h1 h2 h3 s1 s2 s3
mininet>
```

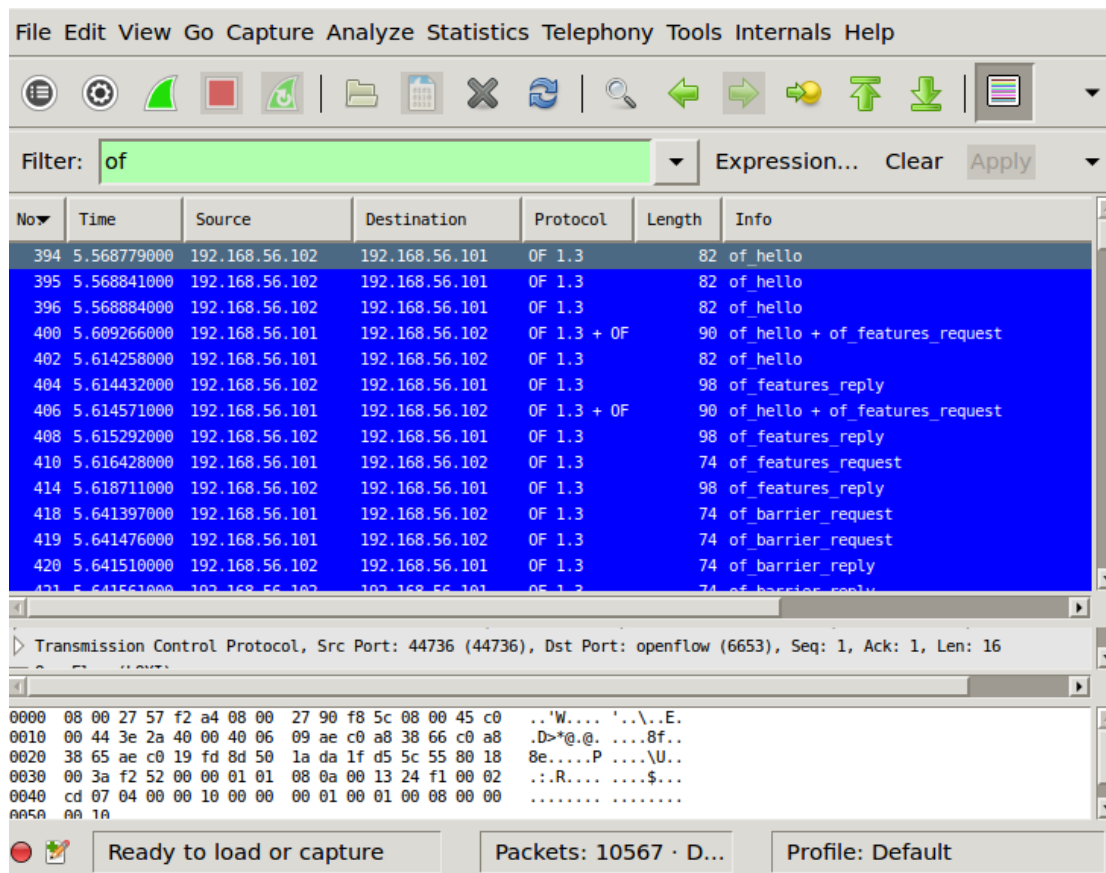
Using the web interface of the controller the created network topology is depicted in topology tab.





In addition, Wireshark tool will allow to analyze the packets of the network.

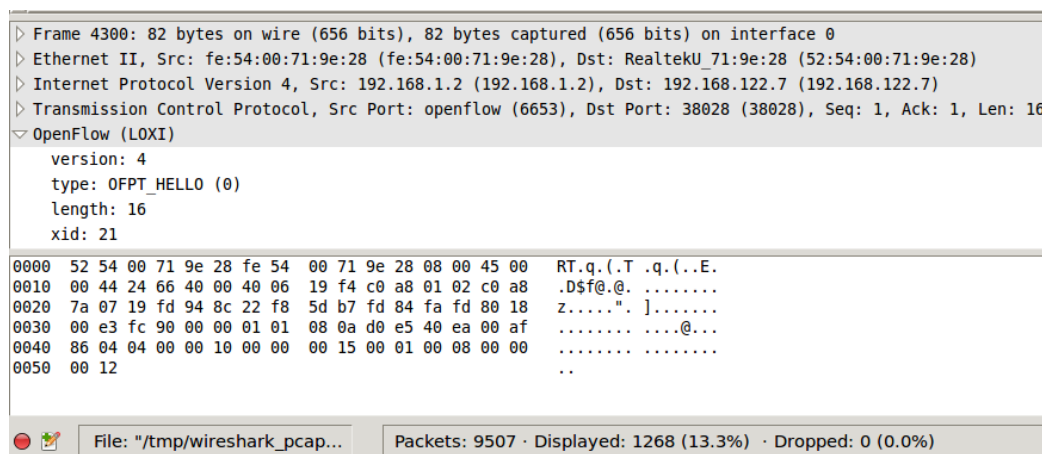
Setting as a filter the “of” key, which stands for OpenFlow protocol, then pressing the “apply” button will show only OpenFlow packets. The figure bellow presents filtered OpenFlow packets.



Wireshark verifies that there is packet exchange between the ODL controller 192.168.56.101 and virtual network topology from Mininet 192.168.56.102. These packets are called “of_hello” and are of type OFPT_HELLO.

The communication is initialized during the TCP handshake, the controller sends its version number to each switch through the of_hello packet, whereupon each switch responds with its supported version number through the of_hello packet. Finally, the controller requests to see the available ports through of_features_request. Since there are three switches three pair of OFPT_HELLO-OFTP_FEATURES_REQUEST generated.

The OF_HELLO packet contains, version, type, length , of_hello_elems, xid.



No.	Time	Source	Destination	Protoco	Lengt	Info
394	5.568779000	192.168.56.102	192.168.56.101	OF 1.3	82	of_hello
395	5.568841000	192.168.56.102	192.168.56.101	OF 1.3	82	of_hello

Transmission Control Protocol, Src Port: 44736 (44736), Dst Port: openflow (6653), Seq: 1, Ack:

OpenFlow (LOXI)

version: 4

type: OFPT_HELLO (0)

length: 16

xid: 1

of_hello_elem list

of_hello_elem_versionbitmap

type: 1

length: 8

of_uint32 list

of_uint32

value: 16

0000 08 00 27 57 f2 a4 08 00 27 90 f8 5c 08 00 45 c0 ..'W.... '..\...E.

0010 00 44 3e 2a 40 00 40 06 09 ae c0 a8 38 66 c0 a8 .D>*@.@.8f..

0020 38 65 ae c0 19 fd 8d 50 1a da 1f d5 5c 55 80 18 8e.....P\U..

0030 00 3a f2 52 00 00 01 01 08 0a 00 13 24 f1 00 02 ...R....\$...

0040 cd 07 04 00 00 10 00 00 00 01 00 01 00 08 00 00

0050 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00

Ready to load or capture

Packets: 10567 · D...

Profile: De

The OFTP_FEATURES_REQUEST packet contains also version, type length and xid.

The new one

No.	Time	Source	Destination	Protoco	Lengt	Info
410	5.616428000	192.168.56.101	192.168.56.102	OF 1.3	74	of_features_request
414	5.618711000	192.168.56.102	192.168.56.101	OF 1.3	98	of_features_reply
418	5.641397000	192.168.56.101	192.168.56.102	OF 1.3	74	of_barrier_request
419	5.641476000	192.168.56.101	192.168.56.102	OF 1.3	74	of_barrier_request
420	5.641510000	192.168.56.102	192.168.56.101	OF 1.3	74	of_barrier_reply

Frame 410: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0

Ethernet II, Src: CadmusCo_57:f2:a4 (08:00:27:57:f2:a4), Dst: CadmusCo_90:f8:5c (08:00:27:90:f8:5c)

Internet Protocol Version 4, Src: 192.168.56.101 (192.168.56.101), Dst: 192.168.56.102 (192.168.56.102)

Transmission Control Protocol, Src Port: openflow (6653), Dst Port: 44736 (44736), Seq: 17, Ack: 17, Len: 8

OpenFlow (LOXI)

version: 4

type: OFPT_FEATURES_REQUEST (5)

length: 8

0000 08 00 27 90 f8 5c 08 00 27 57 f2 a4 08 00 45 00 ..'\... 'W....E.

0010 00 3c 91 54 40 00 40 06 b7 4b c0 a8 38 65 c0 a8 .<.T@.@. .K..8e..

0020 38 66 19 fd ae c0 1f d5 5c 65 8d 50 1a ea 80 18 8f.....\e.P....

0030 00 e3 a0 46 00 00 01 01 08 0a 00 02 cd 12 00 13 ...F.....

0040 24 fd 04 05 00 08 00 00 00 03 \$.

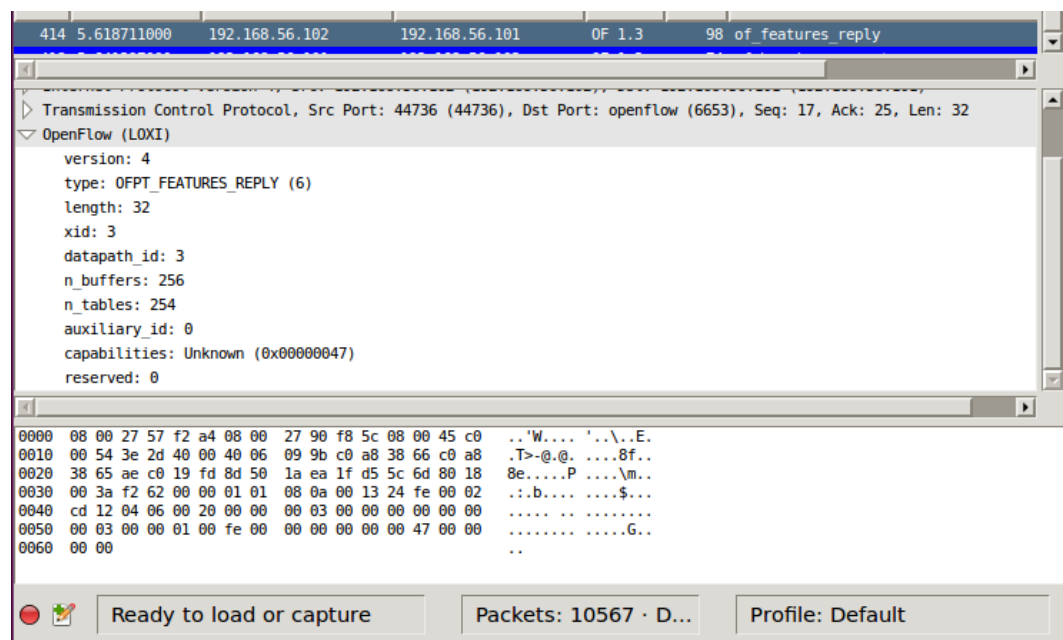
Ready to load or capture

Packets: 10567 · D...

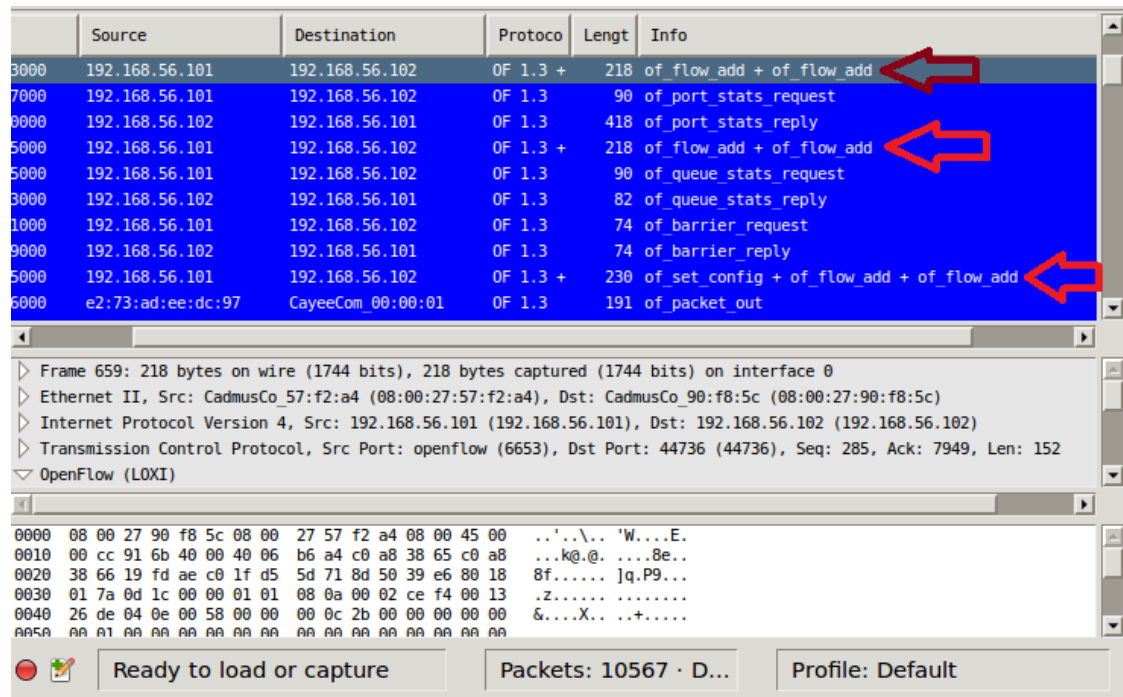
Profile: Default

Features reply

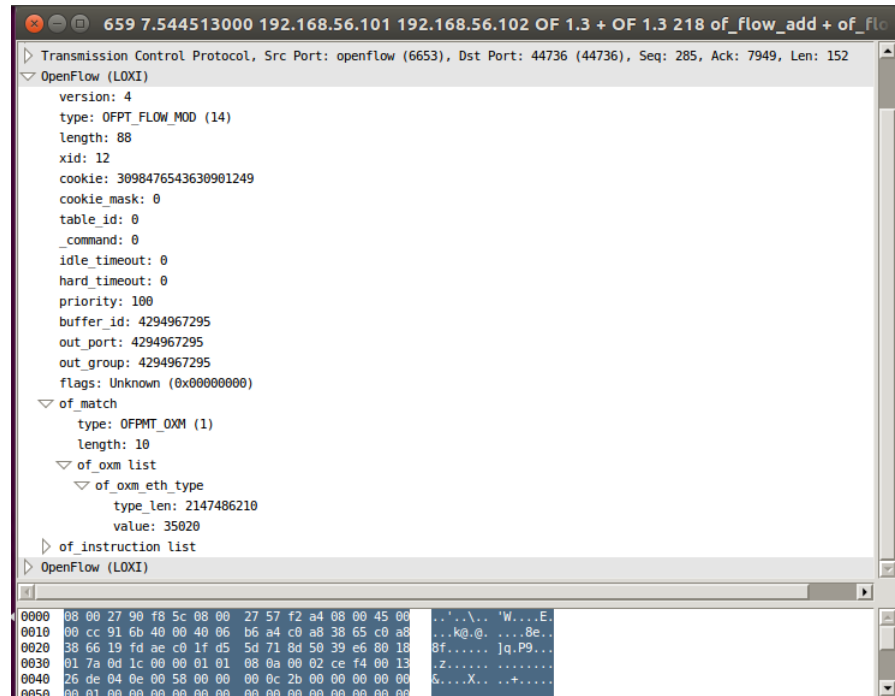
39 of 84

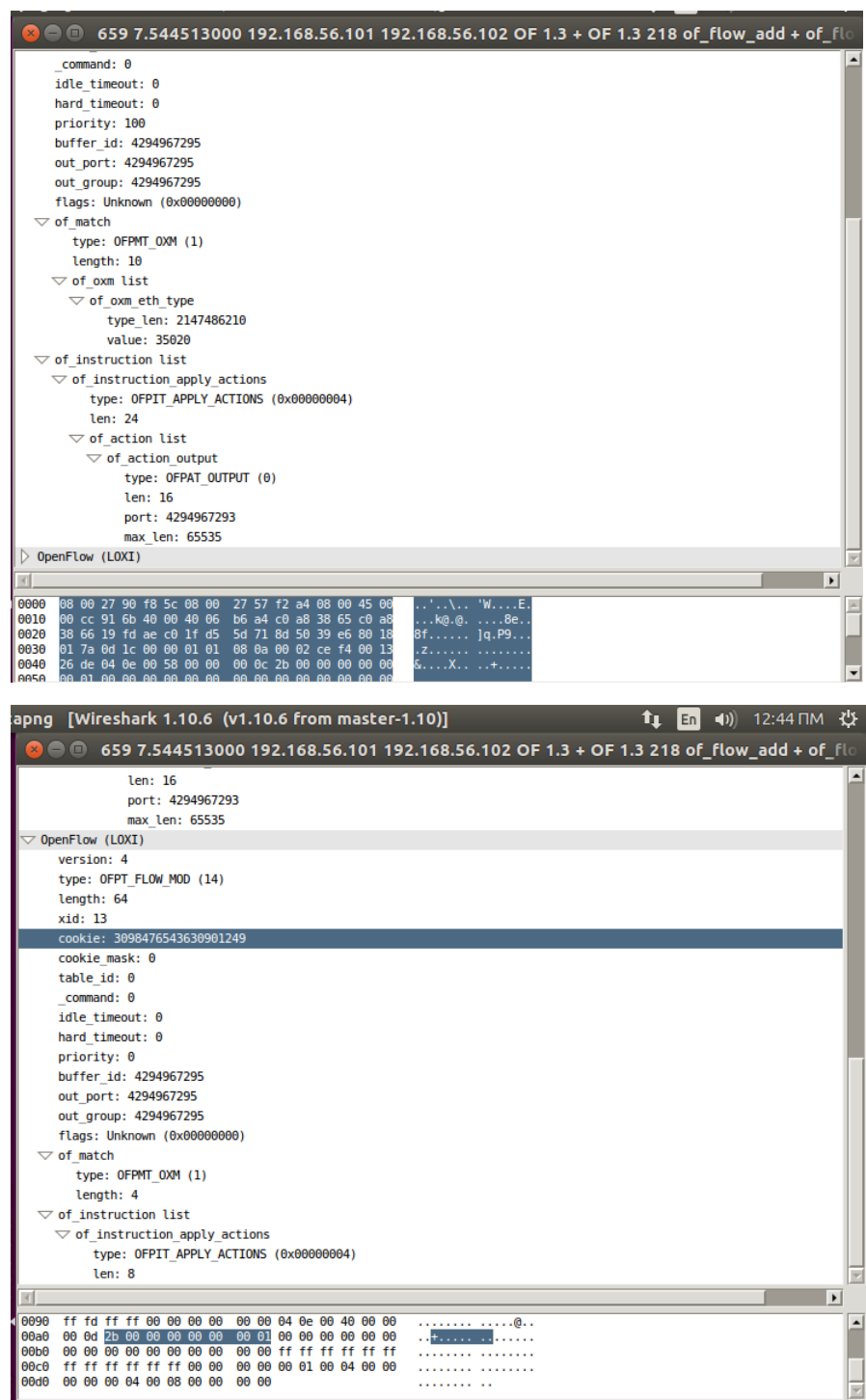


Two pair of add flow mod exist in packet analyzation.



New flow mod message





In the aspect of this research navigate in ODL data store let see what happens using the Restconf API we retrieve the all nodes.

<http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/>

```
{
  "nodes": {
    "node": [
      {
        "id": "openflow:1",
        "node-connector": [
          { },
          { },
          { }
        ],
        "flow-node-inventory:port-number": 59600,
        "flow-node-inventory:serial-number": "None",
        "flow-node-inventory:table": [ ],
        "flow-node-inventory:hardware": "Open vSwitch",
        "flow-node-inventory:description": "None",
        "flow-node-inventory:software": "2.0.2",
        "flow-node-inventory:switch-features": { },
        "flow-node-inventory:manufacturer": "Nicira, Inc.",
        "flow-node-inventory:ip-address": "192.168.56.102",
        "flow-node-inventory:snapshot-gathering-status-start": { },
        "flow-node-inventory:snapshot-gathering-status-end": { }
      },
      {
        "id": "openflow:2",
        "node-connector": [
          { },
          { },
          { },
          { }
        ],
        "flow-node-inventory:port-number": 59598,
        "flow-node-inventory:serial-number": "None",
        "flow-node-inventory:table": [ ],
        "flow-node-inventory:hardware": "Open vSwitch",
        "flow-node-inventory:description": "None",
        "flow-node-inventory:software": "2.0.2",
        "flow-node-inventory:switch-features": { },
        "flow-node-inventory:manufacturer": "Nicira, Inc.",
        "flow-node-inventory:ip-address": "192.168.56.102",
        "flow-node-inventory:snapshot-gathering-status-start": { },
        "flow-node-inventory:snapshot-gathering-status-end": { }
      },
      {
        "id": "openflow:3",
        "node-connector": [
          { },
          { },
          { }
        ],
        "flow-node-inventory:port-number": 59596,
        "flow-node-inventory:serial-number": "None",
        "flow-node-inventory:table": [ ],
        "flow-node-inventory:hardware": "Open vSwitch",
        "flow-node-inventory:description": "None",
        "flow-node-inventory:software": "2.0.2",
        "flow-node-inventory:switch-features": { },
        "flow-node-inventory:manufacturer": "Nicira, Inc.",
        "flow-node-inventory:ip-address": "192.168.56.102",
        "flow-node-inventory:snapshot-gathering-status-start": { },
        "flow-node-inventory:snapshot-gathering-status-end": { }
      }
    ]
  }
}
```

```
{
  "nodes": {
    "node": [
      {
        "id": "openflow:1",
        "node-connector": [
          {
            "id": "openflow:1:1",
            "flow-node-inventory:supported": "",
            "flow-node-inventory:peer-features": "",
            "flow-node-inventory:port-number": 1,
            "flow-node-inventory:advertised-features": "",
            "flow-node-inventory:hardware-address": "a2:97:7d:ed:c7:1c",
            "flow-node-inventory:current-feature": "ten-gb-fd copper",
            "flow-node-inventory:current-speed": 10000000,
            "flow-node-inventory:configuration": "",
            "flow-node-inventory:maximum-speed": 0,
            "flow-node-inventory:name": "s1-eth1",
            "flow-node-inventory:state": {
            },
            "opendaylight-port-statistics:flow-capable-node-connector-statistics": {
            }
          }
        ]
      },
      {
        "id": "openflow:1:2",
        "flow-node-inventory:supported": "",
        "flow-node-inventory:peer-features": "",
        "flow-node-inventory:port-number": 2,
        "flow-node-inventory:advertised-features": "",
        "flow-node-inventory:hardware-address": "6a:72:60:a6:5a:6e",
        "flow-node-inventory:current-feature": "ten-gb-fd copper",
        "flow-node-inventory:current-speed": 10000000,
        "flow-node-inventory:configuration": "",
        "flow-node-inventory:maximum-speed": 0,
        "flow-node-inventory:name": "s1-eth2",
        "flow-node-inventory:state": {
        },
        "opendaylight-port-statistics:flow-capable-node-connector-statistics": {
        }
      },
      {
        "id": "openflow:1:LOCAL",
        "flow-node-inventory:supported": "",
        "flow-node-inventory:peer-features": "",
        "flow-node-inventory:port-number": 4294967294,
        "flow-node-inventory:advertised-features": "",
        "flow-node-inventory:hardware-address": "b6:8c:30:e9:94:42",
        "flow-node-inventory:current-feature": "",
        "flow-node-inventory:current-speed": 0,
        "flow-node-inventory:configuration": "",
        "flow-node-inventory:maximum-speed": 0,
        "flow-node-inventory:name": "s1",
        "flow-node-inventory:state": {
        },
        "opendaylight-port-statistics:flow-capable-node-connector-statistics": {
        }
      }
    ]
  },
  "flow-node-inventory:port-number": 59600,
  "flow-node-inventory:serial-number": "None",
  "flow-node-inventory:table": [
  ],
  "flow-node-inventory:hardware": "Open vSwitch",
  "flow-node-inventory:description": "None",
  "flow-node-inventory:software": "2.0.2",
  "flow-node-inventory:switch-features": {
  },
  "flow-node-inventory:manufacturer": "Nicira, Inc.",
  "flow-node-inventory:ip-address": "192.168.56.102",
  "flow-node-inventory:snapshot-gathering-status-start": {
  },
  "flow-node-inventory:snapshot-gathering-status-end": {
  }
}
```

```
{
  "id": "openflow:1:1",
  "flow-node-inventory:supported": "",
  "flow-node-inventory:peer-features": "",
  "flow-node-inventory:port-number": 1,
  "flow-node-inventory:advertised-features": "",
  "flow-node-inventory:hardware-address": "a2:97:7d:ed:c7:1c",
  "flow-node-inventory:current-feature": "ten-gb-fd copper",
  "flow-node-inventory:current-speed": 10000000,
  "flow-node-inventory:configuration": "",
  "flow-node-inventory:maximum-speed": 0,
  "flow-node-inventory:name": "s1-eth1",
  "flow-node-inventory:state": {
    "blocked": false,
    "link-down": false,
    "live": false
  },
  "opendaylight-port-statistics:flow-capable-node-connector-statistics": {
    "receive-frame-error": 0,
    "packets": {
      "received": 0,
      "transmitted": 326
    },
    "collision-count": 0,
    "transmit-errors": 0,
    "bytes": {
      "received": 0,
      "transmitted": 27710
    },
    "duration": {
      "nanosecond": 219000000,
      "second": 1624
    },
    "receive-crc-error": 0,
    "receive-drops": 0,
    "receive-errors": 0,
    "receive-over-run-error": 0,
    "transmit-drops": 0
  }
}
```

Analyzing a random node from Table 0 matching rules action sets of each flow follows the OpenFlow protocol rules

<http://192.168.56.101:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/table/0/>

```
{
  "flow-node-inventory:table": [
    {
      "id": 0,
      "opendaylight-flow-table-statistics:flow-table-statistics": {
        "active-flows": 4,
        "packets-looked-up": 9144,
        "packets-matched": 9144
      },
      "flow": [
        {
          "id": "#UF$TABLE*0-3",
          "priority": 100,
          "opendaylight-flow-statistics:flow-statistics": {
            "packet-count": 606,
            "byte-count": 51510,
            "duration": {
              "nanosecond": 712000000,
              "second": 3025
            }
          }
        }
      ]
    }
  ]
}
```

```
    }
  },
  "table_id": 0,
  "cookie_mask": 0,
  "hard-timeout": 0,
  "match": {
    "ethernet-match": {
      "ethernet-type": {
        "type": 35020
      }
    }
  },
  "cookie": 3098476543630901000,
  "flags": "",
  "instructions": {
    "instruction": [
      {
        "order": 0,
        "apply-actions": {
          "action": [
            {
              "order": 0,
              "output-action": {
                "max-length": 65535,
                "output-node-connector": "CONTROLLER"
              }
            }
          ]
        }
      }
    ]
  },
  "idle-timeout": 0
},
{
  "id": "#UF$TABLE*0-4",
  "priority": 2,
  "opendaylight-flow-statistics:flow-statistics": {
    "packet-count": 0,
    "byte-count": 0,
    "duration": {
      "nanosecond": 804000000,
      "second": 3021
    }
  },
  "table_id": 0,
  "cookie_mask": 0,
  "hard-timeout": 0,
  "match": {
    "in-port": "1"
  },
  "cookie": 3098476543630901000,
  "flags": "",
  "instructions": {
    "instruction": [
      {
        "order": 0,
        "apply-actions": {
          "action": [
            {
              "order": 0,
              "output-action": {
                "max-length": 65535,
```

```
        "output-node-connector": "2"
      }
    },
    {
      "order": 1,
      "output-action": {
        "max-length": 65535,
        "output-node-connector": "CONTROLLER"
      }
    }
  ]
}
]
},
"idle-timeout": 0
},
{
  "id": "L2switch-0",
  "priority": 0,
  "opendaylight-flow-statistics:flow-statistics": {
    "packet-count": 0,
    "byte-count": 0,
    "duration": {
      "nanosecond": 715000000,
      "second": 3025
    }
  },
  "table_id": 0,
  "cookie_mask": 0,
  "hard-timeout": 0,
  "match": {},
  "cookie": 3098476543630901000,
  "flags": "",
  "idle-timeout": 0
},
{
  "id": "L2switch-1",
  "priority": 2,
  "opendaylight-flow-statistics:flow-statistics": {
    "packet-count": 0,
    "byte-count": 0,
    "duration": {
      "nanosecond": 804000000,
      "second": 3021
    }
  },
  "table_id": 0,
  "cookie_mask": 0,
  "hard-timeout": 0,
  "match": {
    "in-port": "2"
  },
  "cookie": 3098476543630901000,
  "flags": "",
  "instructions": {
    "instruction": [
      {
        "order": 0,
        "apply-actions": {
          "action": [
            {
              "order": 0,
```

```
        "output-action": {
          "max-length": 65535,
          "output-node-connector": "1"
        },
        {
          "order": 1,
          "output-action": {
            "max-length": 65535,
            "output-node-connector": "CONTROLLER"
          }
        }
      ]
    },
    "idle-timeout": 0
  }
]
}
```

The next link navigates to network topology defining the switches and the links (connection between the switches).

<http://localhost:8181/restconf/operational/network-topology:network-topology>

```
{
  "network-topology": {
    "topology": [
      {
        "topology-id": "flow:1",
        "node": [
          {
            "node-id": "openflow:1",
            "opendaylight-topology-inventory:inventory-node-ref": "/opendaylight-inventory:nodes/opendaylight-inventory:node[opendaylight-inventory:id='openflow:1']",
            "termination-point": [
              {
                "tp-id": "openflow:1:2",
                "opendaylight-topology-inventory:inventory-node-connector-ref": "/opendaylight-inventory:nodes/opendaylight-inventory:node[opendaylight-inventory:id='openflow:1']/opendaylight-inventory:node-connector[opendaylight-inventory:id='openflow:1:2']"
              }
            ]
          }
        ]
      },
      {
        "link-id": "openflow:3:2",
        "source": {
          "source-tp": "openflow:3:2",
          "source-node": "openflow:3"
        },
        "destination": {
          "dest-tp": "openflow:2:3",
          "dest-node": "openflow:2"
        }
      },
      {
        "link-id": "openflow:2:3",
        "source": {
          "source-tp": "openflow:2:3",
          "source-node": "openflow:2"
        },
        "destination": {
          "dest-tp": "openflow:3:2",
          "dest-node": "openflow:3"
        }
      }
    ]
  }
}
```

```
s1      Link encap:Ethernet  HWaddr 6a:d6:e9:04:5a:41
        UP BROADCAST RUNNING MTU:1500 Metric:1
        RX packets:1115 errors:0 dropped:1115 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:107040 (107.0 KB)  TX bytes:0 (0.0 B)

s2      Link encap:Ethernet  HWaddr b6:b2:7e:64:39:49
        UP BROADCAST RUNNING MTU:1500 Metric:1
        RX packets:1115 errors:0 dropped:1115 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:107040 (107.0 KB)  TX bytes:0 (0.0 B)

s3      Link encap:Ethernet  HWaddr 2a:30:94:1e:c4:47
        UP BROADCAST RUNNING MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

```
s1-eth2  Link encap:Ethernet  HWaddr 5a:40:d5:c0:4b:19
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:1115 errors:0 dropped:0 overruns:0 frame:0
         TX packets:1115 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:94775 (94.7 KB)  TX bytes:94775 (94.7 KB)

s2-eth2  Link encap:Ethernet  HWaddr ca:fc:56:b9:c3:00
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:1115 errors:0 dropped:0 overruns:0 frame:0
         TX packets:1115 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:94775 (94.7 KB)  TX bytes:94775 (94.7 KB)

s2-eth3  Link encap:Ethernet  HWaddr 12:b4:93:71:43:0d
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:1115 errors:0 dropped:0 overruns:0 frame:0
         TX packets:1115 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:94775 (94.7 KB)  TX bytes:94775 (94.7 KB)

s3-eth2  Link encap:Ethernet  HWaddr 1a:90:4c:73:58:5b
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:1115 errors:0 dropped:0 overruns:0 frame:0
         TX packets:1115 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:94775 (94.7 KB)  TX bytes:94775 (94.7 KB)

mininet@mininet-vm:~$
```

Before to continue to the next step of this use case, let’s see what more the OpenDaylight does for the created network topology relying on the installed features.

Mininet allows you to use the “**dpctl**” command to communicate with the virtual switch and get the status of the flows. The next figure shows this command output and verifies the existence of flows for flow table 0.

```
s2-eth2<->s1-eth2 (OK OK)
s3-eth2<->s2-eth3 (OK OK)
mininet> nodes
available nodes are:
c0 h1 h2 h3 s1 s2 s3
mininet> dpctl dump-flows -O OpenFlow13
*** s1 ***
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x2b00000000000000, duration=1330.951s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=1 actions=output:2,CONTROLLER:65535
 cookie=0x2b00000000000001, duration=1330.951s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=2 actions=output:1,CONTROLLER:65535
 cookie=0x2b00000000000000, duration=1334.859s, table=0, n_packets=268, n_bytes=22780, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000000, duration=1334.862s, table=0, n_packets=0, n_bytes=0, priority=0 actions=drop
*** s2 ***
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x2b00000000000004, duration=1330.95s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=3 actions=output:1,output:2,CONTROLLER:65535
 cookie=0x2b00000000000002, duration=1330.953s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=1 actions=output:2,output:3,CONTROLLER:65535
 cookie=0x2b00000000000003, duration=1330.952s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=2 actions=output:1,output:3,CONTROLLER:65535
 cookie=0x2b00000000000002, duration=1334.789s, table=0, n_packets=534, n_bytes=45390, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000002, duration=1334.797s, table=0, n_packets=0, n_bytes=0, priority=0 actions=drop
*** s3 ***
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x2b00000000000005, duration=1330.947s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=1 actions=output:2,CONTROLLER:65535
 cookie=0x2b00000000000006, duration=1330.946s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=2 actions=output:1,CONTROLLER:65535
 cookie=0x2b00000000000001, duration=1334.864s, table=0, n_packets=268, n_bytes=22780, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000001, duration=1334.864s, table=0, n_packets=0, n_bytes=0, priority=0 actions=drop
mininet>
```

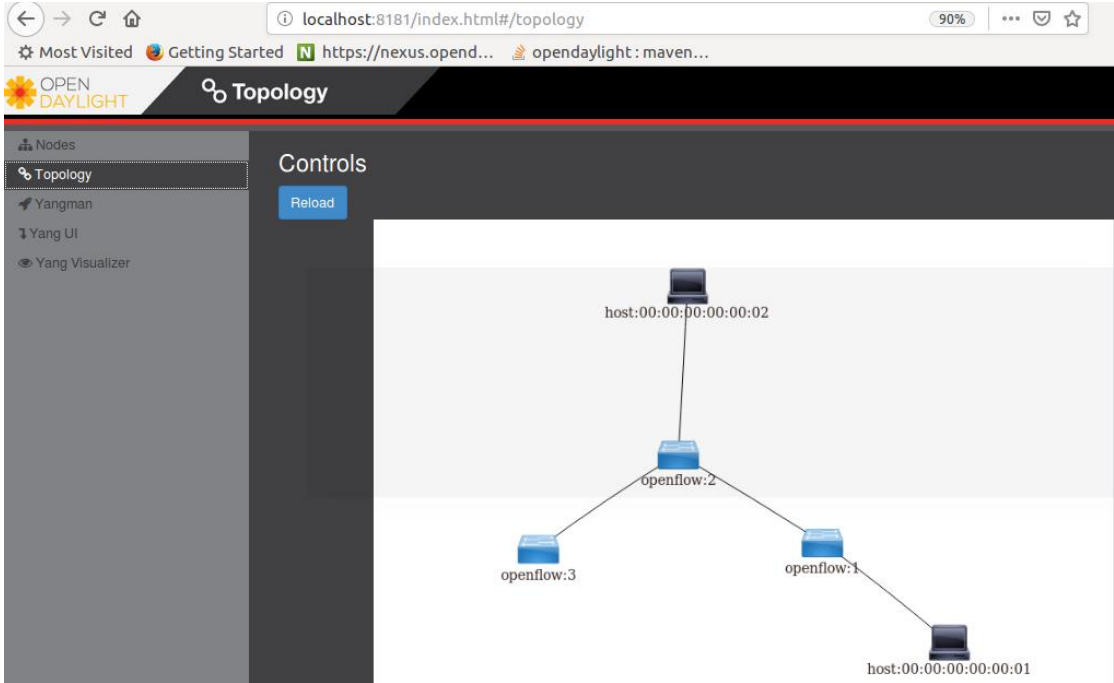
As yet, the process that has been presented registers the OpenFlow-enabled switches with the ODL and are stores related data to inventory and network topology. The registration process is accomplished via an OpenFlow HELLO packet coming from the OpenFlow switch to the ODL controller. Then ODL controller accepts the request and check whether the switch is allowed to be part of ODL's SDN domain.

Moving to the next step, the handling of host connection will be presented. For this scenario the “**ping**” command of the Mininet tool will be used. The next figure shows how a h1 from switch 1 (s1) pings the h2 of switch (s2).

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=598 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=96.3 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=3.83 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=1.11 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=7.62 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=1.22 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=1.15 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=2.06 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=1.16 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=2.41 ms
```

The first figure of the DLUX UI included only the switches, but after the connection between the hosts h1 and h2 the hosts of the switches s1 and s2 have been generated.

The explanation for hosts appearance after ping is related to how the ODL handles the connection between the hosts. There are several steps in order to retrieve the response from h2 host. ODL starts to identify the existence of the hosts only when there is a request for connection between them. Next the process of the host connection will be analyzed in detail.



The SDN controller knows the exiting switches and not for hosts, but the switches do not know how to handle the received frame that contains for destination a specific MAC address or the broadcast MAC.

All starts, when the h1 notices that it does not know the MAC address the h2 then, it sends ARP (Address Resolution Protocol) packet (frame layer 2) to switch s1 to find the IP address of the h2. This packet received from the switch does not has IP source and destination is just a broadcast frame.

However, when the switch s1 receives the ARP packet it checks in its flow tables for this broadcast frame if exists any matching flow entry, in this case it does not find any and then it encapsulates the packet in an OpenFlow **packet_in** packet and sends it to the ODL controller. The ODL decides what to do with this packet utilizing the Arp Handler feature.

Once the ARP packet is forwarded to all the switch ports and the ARP reply is forwarded back to host 1, which was the main ARP queried, host 1 starts sending layer 3 ICMP packets to host 2. The packets have a source and destination IP address as well as a MAC address. Again, the OVS switch does not know how to forward the packet as it does not have any flow entry for host 1 and host 2 MAC addresses yet.

The explanation for this step is that the controller sends the ARP packet to every switches port. Each switch knows about the connected hosts to its ports, as a result h2 responses to the h1.

Then the response packet that comes contains for source the MAC address of the h2 and destination the h1.

However, the controller has no prior knowledge of these MAC addresses. As a result, a **flow_mod** packet is sent from the controller to the switch, in order to add a flow to its flow table. Then, the ICMP packets are send from h1 to h2. Since the flow tables are not empty, the ping continues without controller corporation. The next figure shows the steps of the host h1 and h2 connection.

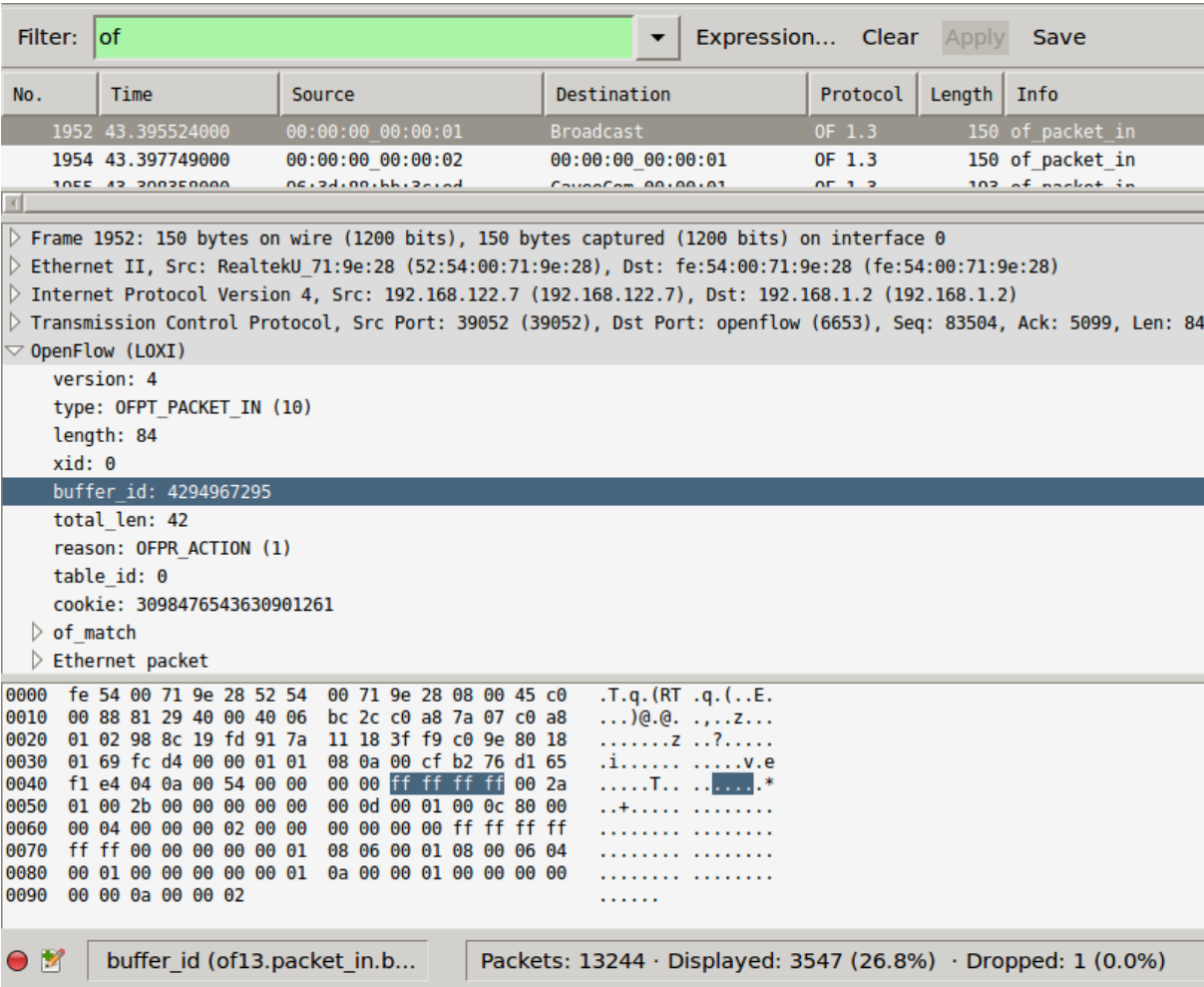
The packets in lines 734, 736, 740 describe how the packets are broadcasted, next, in line the 742 verifies that the response comes from source address 00:00:00:00:00:02. After that the flow add packets is forwarded from ODL to switches. Finally, we see that the final packets exchanging is completed from IP address 10.0.0.1 to 10.0.0.2 and backward.

The I2-switch module that is install in ODL is responsible to handle the received packets from switches. As a result, when the controller receives the packets it forwards the packets to all ports.

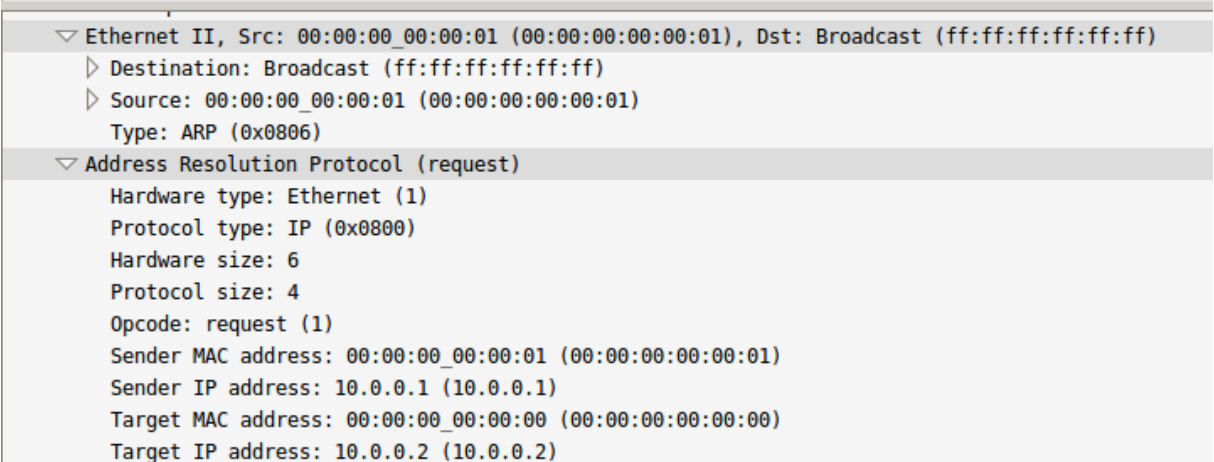
Filter: of Expression... Clear Apply Save						
No.	Time	Source	Destination	Protocol	Length	Info
649	11.662399000	192.168.122.7	192.168.1.4	OF 1.3	74	of_barrier_reply
734	15.590323000	00:00:00_00:00:01	Broadcast	OF 1.3	150	of_packet_in
736	15.600970000	00:00:00_00:00:01	Broadcast	OF 1.3	150	of_packet_in
740	15.693105000	00:00:00_00:00:01	Broadcast	OF 1.3	150	of_packet_in
742	15.716496000	00:00:00_00:00:02	00:00:00_00:00:01	OF 1.3	150	of_packet_in
744	15.738164000	192.168.1.4	192.168.122.7	OF 1.3	162	of_flow_add
745	15.738457000	192.168.1.4	192.168.122.7	OF 1.3	162	of_flow_add
746	15.762656000	00:00:00_00:00:02	00:00:00_00:00:01	OF 1.3	150	of_packet_in
748	15.773659000	192.168.1.4	192.168.122.7	OF 1.3 +	258	of_flow_add + of_flow_add
750	15.791850000	192.168.1.4	192.168.122.7	OF 1.3	162	[TCP Retransmission] of_flow_add
752	15.869751000	00:00:00_00:00:02	00:00:00_00:00:01	OF 1.3	150	of_packet_in
754	15.904464000	10.0.0.1	10.0.0.2	OF 1.3	206	of_packet_in
755	15.905398000	10.0.0.1	10.0.0.2	OF 1.3	206	of_packet_in
759	16.003714000	10.0.0.2	10.0.0.1	OF 1.3	206	of_packet_in
761	16.020763000	10.0.0.2	10.0.0.1	OF 1.3	206	of_packet_in
763	16.092735000	10.0.0.1	10.0.0.2	OF 1.3	206	of_packet_in
765	16.099626000	10.0.0.2	10.0.0.1	OF 1.3	206	of_packet_in
767	16.111621000	16:02:51:e3:6f:68	CayeeCom_00:00:01	LLDP + OF	577	Chassis Id = 00:00:00:00:00:02 Port
Frame /69: 452 bytes on wire (3616 bits), 452 bytes captured (3616 bits) on interface 0						
Ethernet II, Src: fe:54:00:71:9e:28 (fe:54:00:71:9e:28), Dst: RealtekU_71:9e:28 (52:54:00:71:9e:28)						
Internet Protocol Version 4, Src: 192.168.1.4 (192.168.1.4), Dst: 192.168.122.7 (192.168.122.7)						
Transmission Control Protocol, Src Port: openflow (6653), Dst Port: 39340 (39340), Seq: 1183, Ack: 854, Len: 386						
OpenFlow (LOXI)						
0000	52 54 00 71 9e 28 fe 54	00 71 9e 28 08 00 45 00	RT.q.(.T .q.(.E.			
0010	01 b6 ec 10 40 00 40 06	50 d5 c0 a8 01 04 c0 a8	@.@. P.....			
0020	7a 07 19 fd 99 ac bb 4e	a8 2c 2a 6a be a2 80 18	Z.....N .,*j....			
0030	05 ac fe 04 00 00 01 01	08 0a 19 ba 92 0c 00 3e	>			
0040	05 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00				
File: "/tmp/wireshark_pcap..." Packets: 6643 · Displayed: 512 (7.7%) · Dropped: 0 (0.0%)						

Filter: of							Expression...	Clear	Apply	Save
No.	Time	Source	Destination	Protocol	Length	Info				
1952	43.395524000	00:00:00_00:00:01	Broadcast	OF 1.3	150	of_packet_in				
1954	43.397749000	00:00:00_00:00:02	00:00:00_00:00:01	OF 1.3	150	of_packet_in				
1955	43.398358000	96:3d:88:bb:3c:ed	CayeeCom_00:00:01	OF 1.3	193	of_packet_in				
1956	43.398892000	00:00:00_00:00:01	Broadcast	OF 1.3	150	of_packet_in				
1957	43.399224000	00:00:00_00:00:02	00:00:00_00:00:01	OF 1.3	150	of_packet_in				
1958	43.399627000	00:00:00_00:00:01	Broadcast	OF 1.3	150	of_packet_in				
1959	43.399901000	00:00:00_00:00:02	00:00:00_00:00:01	OF 1.3	150	of_packet_in				
1967	43.444651000	192.168.1.2	192.168.122.7	OF 1.3	74	of_barrier_request				
1968	43.446352000	192.168.1.2	192.168.122.7	OF 1.3	74	of_barrier_request				
1970	43.448007000	192.168.1.2	192.168.122.7	OF 1.3	74	of_barrier_request				
1973	43.488078000	10.0.0.1	10.0.0.2	OF 1.3	206	of_packet_in				
1977	43.595424000	192.168.122.7	192.168.1.2	OF 1.3	74	of_barrier_reply				
1981	43.748543000	10.0.0.1	10.0.0.2	OF 1.3	206	of_packet_in				
1985	43.792061000	10.0.0.2	10.0.0.1	OF 1.3	206	of_packet_in				
1986	43.793805000	192.168.122.7	192.168.1.2	OF 1.3	74	of_barrier_reply				
1987	43.796409000	10.0.0.2	10.0.0.1	OF 1.3	206	of_packet_in				
1991	43.811850000	192.168.122.7	192.168.1.2	OF 1.3	74	of_barrier_reply				
1995	43.862642000	10.0.0.1	10.0.0.2	OF 1.3	206	of_packet_in				
1997	43.868227000	10.0.0.2	10.0.0.1	OF 1.3	206	of_packet_in				
2002	44.023270000	192.168.1.2	192.168.122.7	OF 1.3	82	of_table_stats_request				
Internet Protocol Version 4, Src: 192.168.122.7 (192.168.122.7), Dst: 192.168.1.2 (192.168.1.2)										
Transmission Control Protocol, Src Port: 39052 (39052), Dst Port: openflow (6653), Seq: 83504, Ack: 5099, Len: 84										
OpenFlow (LOXI)										
0000	fe 54 00 71 9e 28 52 54	00 71 9e 28 08 00 45 c0	.T.q.(RT .q.(.E.							
0010	00 88 81 29 40 00 40 06	bc 2c c0 a8 7a 07 c0 a8	...)@.@. ...z...							
0020	01 02 98 8c 19 fd 91 7a	11 18 3f f9 c0 9e 80 18	z .?.....							
0030	01 69 fc d4 00 00 01 01	08 0a 00 cf b2 76 d1 65	.i.....v.e							
0040	f1 e4 04 0a 00 54 00 00	00 00 ff ff ff 00 2a	T.....*							
Ready to load or capture							Packets: 13244 · Displayed: 3547 (26.8%) · Dropped: 1 (0.0%)			

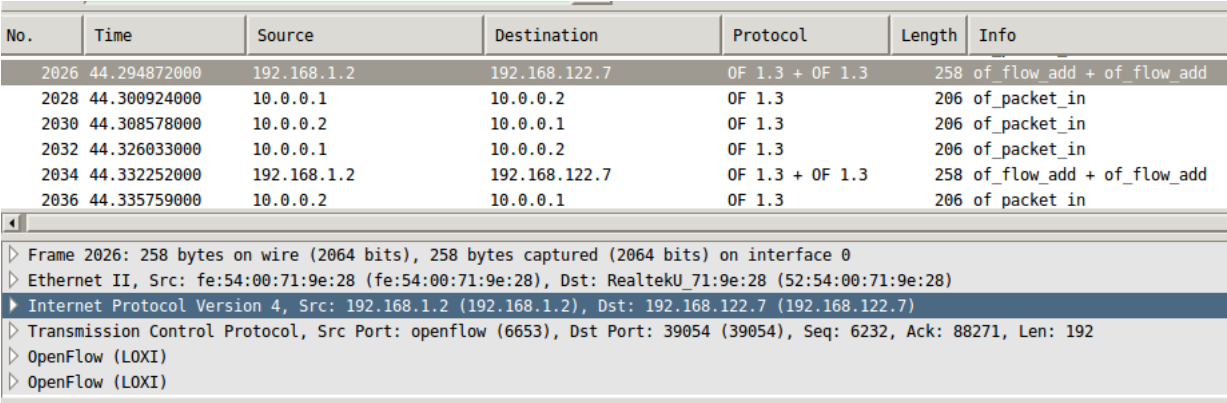
The **of_packet_in** contains: version, type, xid, buffer_id, total_len, reason, table_id, of_match, Ethernet packet. The most important field of the packet that is the Ethernet packet.



The figure bellow verifies that the of_packet_in packet Source MAC address is 00:00:00:00:00:01 and destination is the broadcast frame (ff:ff:ff:ff:ff:ff). In ARP part the Source IP address 10.0.0.1 and Destination IP address is 10.0.0.2.



The of_packet_in packet is broadcasted to OpenDaylight controller and since the l2switch module is installed it starts to take control of the of_packet_in packet. L2switch module also does not know where the target host h2 is located. The only information that is retrieved from l2switch is the source and target ip address along with MAC address when it reads the packet. So, it uses the ofpt_flow_mod packet of table with id 0 in order to clarify to switches how to broadcast the packet to all active ports.

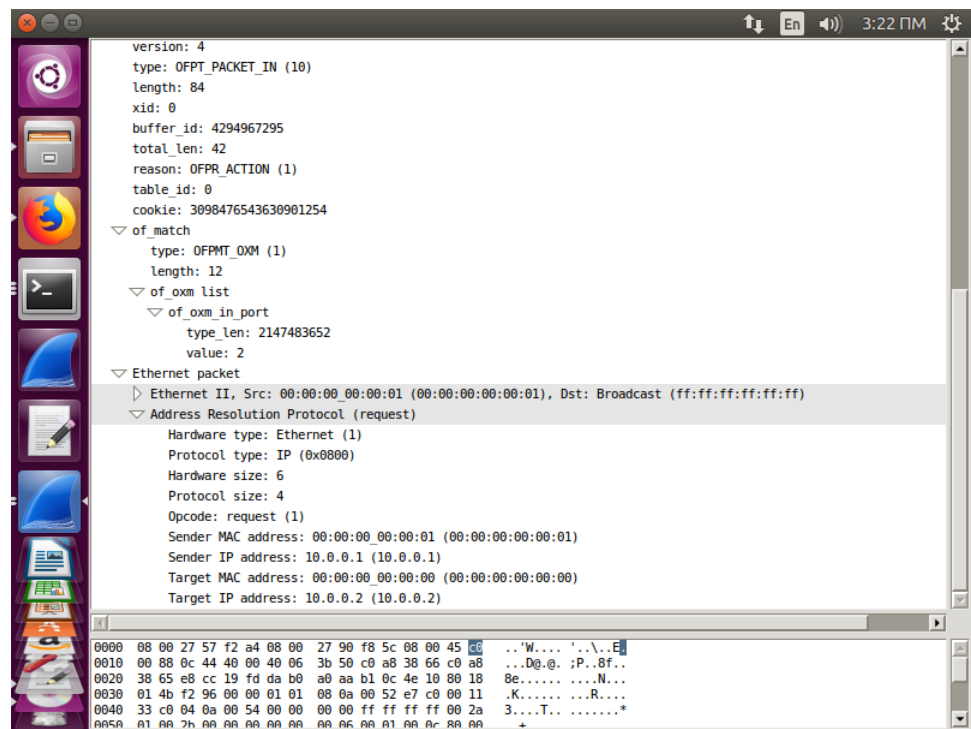
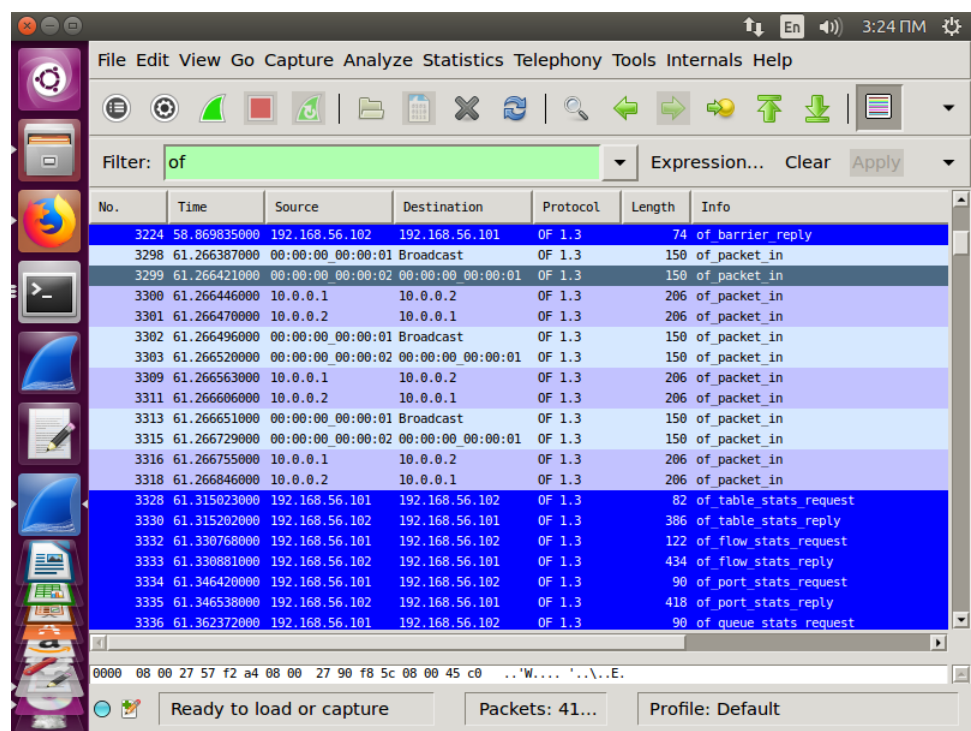


The content of the OpenFlow ofpt_flow_mod packet is presented in the figured.

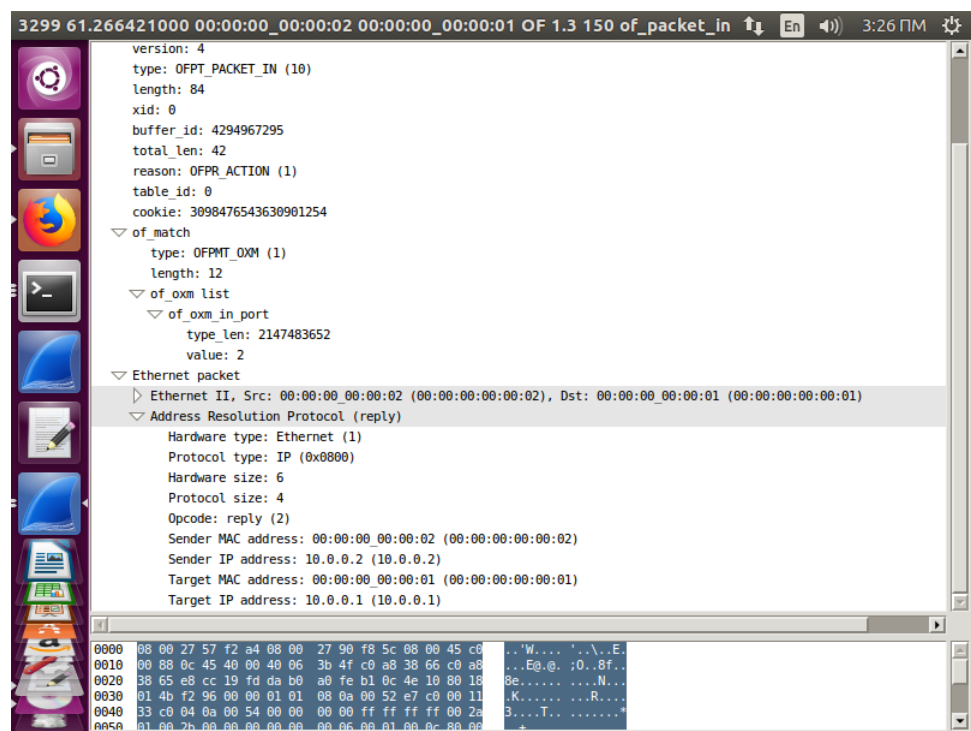
OpenFlow (LOXI) version: 4 type: OFPT_FLOW_MOD (14) length: 96 xid: 8200 cookie: 3026418949592973312 cookie_mask: 0 table_id: 0 _command: 0 idle_timeout: 600 hard_timeout: 300 priority: 10 buffer_id: 4294967295 out_port: 4294967295 out_group: 4294967295 flags: Unknown (0x00000000) of_match type: OFPMT_OXM (1) length: 24 of_oxm_list of_oxm_eth_dst type_len: 2147485190 value: 00:00:00_00:00:02 (00:00:00:00:00:02) of_oxm_eth_src type_len: 2147485702 value: 00:00:00_00:00:01 (00:00:00:00:00:01) of_instruction_list of_instruction_apply_actions type: OFPIT_APPLY_ACTIONS (0x00000004) len: 24 of_action_list of_action_output type: OFPAT_OUTPUT (0) len: 16 port: 1 max_len: 65535	OpenFlow (LOXI) version: 4 type: OFPT_FLOW_MOD (14) length: 96 xid: 8201 cookie: 3026418949592973313 cookie_mask: 0 table_id: 0 _command: 0 idle_timeout: 600 hard_timeout: 300 priority: 10 buffer_id: 4294967295 out_port: 4294967295 out_group: 4294967295 flags: Unknown (0x00000000) of_match type: OFPMT_OXM (1) length: 24 of_oxm_list of_oxm_eth_dst type_len: 2147485190 value: 00:00:00_00:00:01 (00:00:00:00:00:01) of_oxm_eth_src type_len: 2147485702 value: 00:00:00_00:00:02 (00:00:00:00:00:02) of_instruction_list of_instruction_apply_actions type: OFPIT_APPLY_ACTIONS (0x00000004) len: 24 of_action_list of_action_output type: OFPAT_OUTPUT (0) len: 16 port: 2 max_len: 65535
--	--

The flows above verify that there are two active ports, port 1 and port 2.

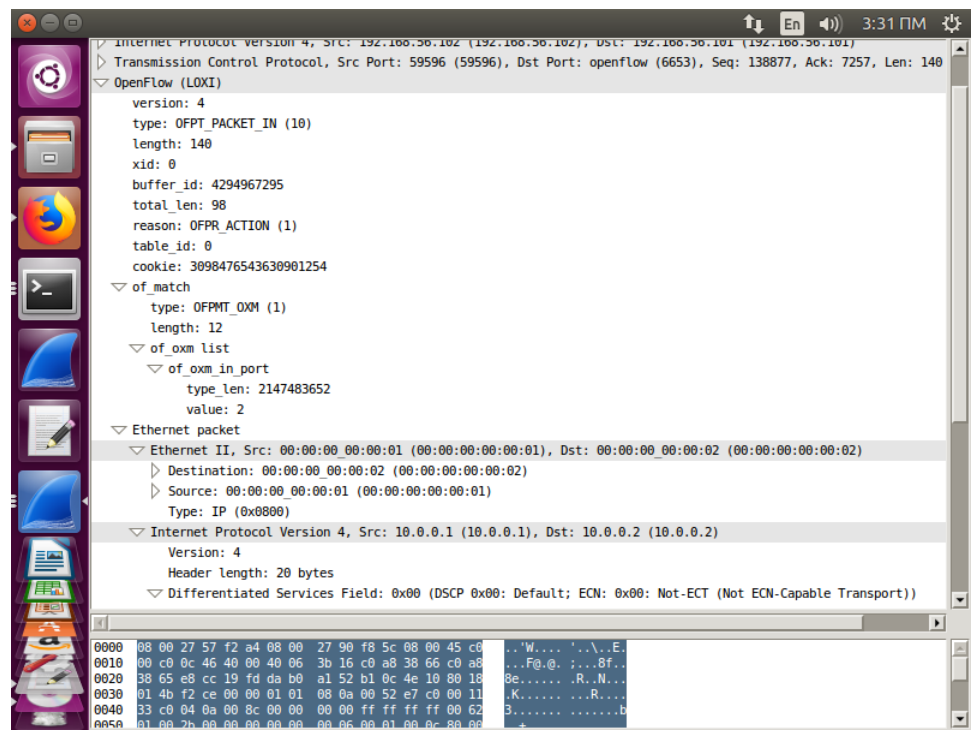
OpenFlow (LOXI) version: 4 type: OFPT_FLOW_MOD (14) length: 96 xid: 7550 cookie: 3026418949592973314 cookie_mask: 0 table_id: 0 _command: 0 idle_timeout: 600 hard_timeout: 300 priority: 10 buffer_id: 4294967295 out_port: 4294967295 out_group: 4294967295 flags: Unknown (0x00000000) of_match type: OFPMT_OXM (1) length: 24 of_oxm_list of_oxm_eth_dst type_len: 2147485190 value: 00:00:00_00:00:02 (00:00:00:00:00:02) of_oxm_eth_src type_len: 2147485702 value: 00:00:00_00:00:01 (00:00:00:00:00:01) of_instruction_list of_instruction_apply_actions type: OFPIT_APPLY_ACTIONS (0x00000004) len: 24 of_action_list of_action_output type: OFPAT_OUTPUT (0) len: 16 port: 2 max_len: 65535	OpenFlow (LOXI) version: 4 type: OFPT_FLOW_MOD (14) length: 96 xid: 7551 cookie: 3026418949592973315 cookie_mask: 0 table_id: 0 _command: 0 idle_timeout: 600 hard_timeout: 300 priority: 10 buffer_id: 4294967295 out_port: 4294967295 out_group: 4294967295 flags: Unknown (0x00000000) of_match type: OFPMT_OXM (1) length: 24 of_oxm_list of_oxm_eth_dst type_len: 2147485190 value: 00:00:00_00:00:01 (00:00:00:00:00:01) of_oxm_eth_src type_len: 2147485702 value: 00:00:00_00:00:02 (00:00:00:00:00:02) of_instruction_list of_instruction_apply_actions type: OFPIT_APPLY_ACTIONS (0x00000004) len: 24 of_action_list of_action_output type: OFPAT_OUTPUT (0) len: 16 port: 1 max_len: 65535
--	--

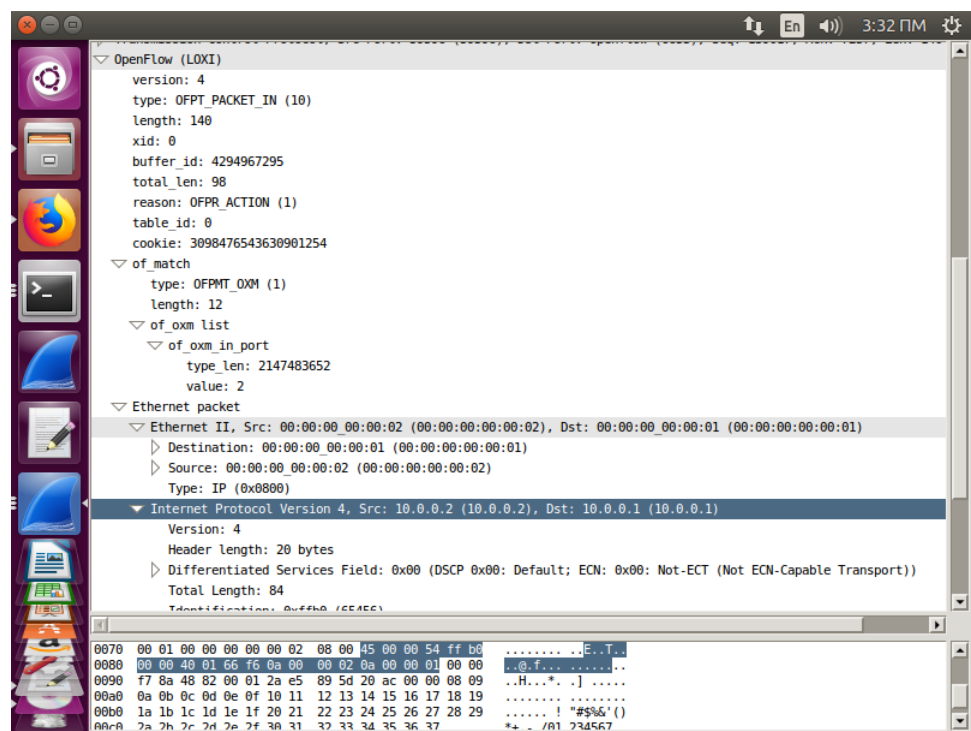


Analizing the packet with number #3299



Analizing the packet with number #3300





Verification through `dpctl dump-flows` command.

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=2277>
<Host h2: h2-eth0:10.0.0.2 pid=2279>
<Host h3: h3-eth0:10.0.0.3 pid=2281>
<OVSSwitch{'protocols': 'OpenFlow13'} s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=2286>
<OVSSwitch{'protocols': 'OpenFlow13'} s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=2289>
<OVSSwitch{'protocols': 'OpenFlow13'} s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=2292>
<RemoteController{'ip': '192.168.1.2'} c0: 192.168.1.2:6653 pid=2271>
mininet>
```

```
mininet> dpctl dump-flows -O OpenFlow13
*** s1 ***
OFPST FLOW reply (OF1.3) (xid=0x2):
  cookie=0x2b00000000000000, duration=5199.639s, table=0, n_packets=58, n_bytes=5404, priority=2,in_port
  =1 actions=output:2,CONTROLLER:65535
  cookie=0x2b00000000000001, duration=5199.639s, table=0, n_packets=58, n_bytes=5404, priority=2,in_port
  =2 actions=output:1,CONTROLLER:65535
  cookie=0x2b00000000000000, duration=5203.547s, table=0, n_packets=1042, n_bytes=88570, priority=100,d
  l_type=0x88cc actions=CONTROLLER:65535
  cookie=0x2a000000000000023, duration=235.317s, table=0, n_packets=247, n_bytes=23870, idle timeout=600,
  hard timeout=300, priority=10,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:02 actions=output:1
  cookie=0x2a000000000000022, duration=235.317s, table=0, n_packets=247, n_bytes=23870, idle timeout=600,
  hard timeout=300, priority=10,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:01 actions=output:1
  cookie=0x2b00000000000000, duration=5203.55s, table=0, n_packets=0, n_bytes=0, priority=0 actions=drop
*** s2 ***
OFPST FLOW reply (OF1.3) (xid=0x2):
  cookie=0x2b00000000000004, duration=5199.637s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=3 a
  ctions=output:1,output:2,CONTROLLER:65535
  cookie=0x2b00000000000002, duration=5199.64s, table=0, n_packets=58, n_bytes=5404, priority=2,in_port=
  1 actions=output:2,output:3,CONTROLLER:65535
  cookie=0x2b00000000000003, duration=5199.639s, table=0, n_packets=58, n_bytes=5404, priority=2,in_port
  =2 actions=output:1,output:3,CONTROLLER:65535
  cookie=0x2b00000000000002, duration=5203.476s, table=0, n_packets=2082, n_bytes=176970, priority=100,d
  l_type=0x88cc actions=CONTROLLER:65535
  cookie=0x2a00000000000001f, duration=235.333s, table=0, n_packets=247, n_bytes=23870, idle timeout=600,
  hard timeout=300, priority=10,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:02 actions=output:1
  cookie=0x2a00000000000001e, duration=235.333s, table=0, n_packets=247, n_bytes=23870, idle timeout=600,
  hard timeout=300, priority=10,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:01 actions=output:2
  cookie=0x2b00000000000002, duration=5203.484s, table=0, n_packets=0, n_bytes=0, priority=0 actions=dro
  p
*** s3 ***
OFPST FLOW reply (OF1.3) (xid=0x2):
  cookie=0x2b00000000000005, duration=5199.635s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=1 a
  ctions=output:2,CONTROLLER:65535
  cookie=0x2b00000000000006, duration=5199.634s, table=0, n_packets=108, n_bytes=10024, priority=2,in_po
  rt=2 actions=output:1,CONTROLLER:65535
  cookie=0x2b00000000000001, duration=5203.552s, table=0, n_packets=1042, n_bytes=88570, priority=100,d
  l_type=0x88cc actions=CONTROLLER:65535
  cookie=0x2b00000000000001, duration=5203.552s, table=0, n_packets=0, n_bytes=0, priority=0 actions=dro
```

Analyzing a random node from Table 0 matching rules action sets of each flow follows the OpenFlow protocol rules for **openflow:1** and **openflow:2** ,after ping command.

<http://192.168.56.101:8181/restconf/operational/.opendaylight-inventory:nodes/node/openflow:1/table/0/>

{

```
"flow-node-inventory:table": [  
  {  
    "id": 0,  
    "opendaylight-flow-table-statistics:flow-table-statistics": {  
      "active-flows": 6,  
      "packets-looked-up": 12086,  
      "packets-matched": 12086  
    },  
    "flow": [  
      {  
        "id": "#UF$TABLE*0-3",  
        "priority": 100,  
        "opendaylight-flow-statistics:flow-statistics": {  
          "packet-count": 842,  
          "byte-count": 71570,  
          "duration": {  
            "nanosecond": 700000000,  
            "second": 4201  
          }  
        },  
        "table_id": 0,  
        "cookie_mask": 0,  
        "hard-timeout": 0,  
        "match": {  
          "ethernet-match": {  
            "ethernet-type": {  
              "type": 35020  
            }  
          }  
        },  
        "cookie": 3098476543630901000,  
        "flags": "",  
        "instructions": {  
          "instruction": [  
            {  
              "order": 0,  
              "apply-actions": {  
                "action": [  
                  {  
                    "order": 0,  
                    "output-action": {  
                      "max-length": 65535,  
                      "output-node-connector": "CONTROLLER"  
                    }  
                  }  
                ]  
              }  
            }  
          ]  
        },  
        "idle-timeout": 0  
      },  
      {  
        "id": "#UF$TABLE*0-4",  
        "priority": 2,  
        "opendaylight-flow-statistics:flow-statistics": {  
          "packet-count": 18,  
          "byte-count": 1652,  
          "duration": {  
            "nanosecond": 792000000,  
            "second": 4197  
          }  
        },  
        "idle-timeout": 0  
      }  
    ]  
  }  
]
```

```
"table_id": 0,
"cookie_mask": 0,
"hard-timeout": 0,
"match": {
  "in-port": "1"
},
"cookie": 3098476543630901000,
"flags": "",
"instructions": {
  "instruction": [
    {
      "order": 0,
      "apply-actions": {
        "action": [
          {
            "order": 0,
            "output-action": {
              "max-length": 65535,
              "output-node-connector": "2"
            }
          },
          {
            "order": 1,
            "output-action": {
              "max-length": 65535,
              "output-node-connector": "CONTROLLER"
            }
          }
        ]
      }
    }
  ]
},
"idle-timeout": 0
},
{
  "id": "L2switch-0",
  "priority": 0,
  "opendaylight-flow-statistics:flow-statistics": {
    "packet-count": 0,
    "byte-count": 0,
    "duration": {
      "nanosecond": 703000000,
      "second": 4201
    }
  },
  "table_id": 0,
  "cookie_mask": 0,
  "hard-timeout": 0,
  "match": {},
  "cookie": 3098476543630901000,
  "flags": "",
  "idle-timeout": 0
},
{
  "id": "L2switch-11",
  "priority": 10,
  "opendaylight-flow-statistics:flow-statistics": {
    "packet-count": 174,
    "byte-count": 16772,
    "duration": {
      "nanosecond": 422000000,
      "second": 167
    }
  }
}
```

```
    }
  },
  "table_id": 0,
  "cookie_mask": 0,
  "hard-timeout": 300,
  "match": {
    "ethernet-match": {
      "ethernet-source": {
        "address": "00:00:00:00:00:01"
      },
      "ethernet-destination": {
        "address": "00:00:00:00:00:02"
      }
    }
  },
  "cookie": 3026418949592973300,
  "flags": "",
  "instructions": {
    "instruction": [
      {
        "order": 0,
        "apply-actions": {
          "action": [
            {
              "order": 0,
              "output-action": {
                "max-length": 65535,
                "output-node-connector": "2"
              }
            }
          ]
        }
      }
    ]
  },
  "idle-timeout": 600
},
{
  "id": "L2switch-1",
  "priority": 2,
  "opendaylight-flow-statistics:flow-statistics": {
    "packet-count": 18,
    "byte-count": 1652,
    "duration": {
      "nanosecond": 792000000,
      "second": 4197
    }
  },
  "table_id": 0,
  "cookie_mask": 0,
  "hard-timeout": 0,
  "match": {
    "in-port": "2"
  },
  "cookie": 3098476543630901000,
  "flags": "",
  "instructions": {
    "instruction": [
      {
        "order": 0,
        "apply-actions": {
          "action": [
            {
```

```
        "order": 0,
        "output-action": {
            "max-length": 65535,
            "output-node-connector": "1"
        }
    },
    {
        "order": 1,
        "output-action": {
            "max-length": 65535,
            "output-node-connector": "CONTROLLER"
        }
    }
]
}
}
],
},
"idle-timeout": 0
},
{
    "id": "L2switch-10",
    "priority": 10,
    "opendaylight-flow-statistics:flow-statistics": {
        "packet-count": 174,
        "byte-count": 16772,
        "duration": {
            "nanosecond": 422000000,
            "second": 167
        }
    },
    "table_id": 0,
    "cookie_mask": 0,
    "hard-timeout": 300,
    "match": {
        "ethernet-match": {
            "ethernet-source": {
                "address": "00:00:00:00:00:02"
            },
            "ethernet-destination": {
                "address": "00:00:00:00:00:01"
            }
        }
    },
    "cookie": 3026418949592973300,
    "flags": "",
    "instructions": {
        "instruction": [
            {
                "order": 0,
                "apply-actions": {
                    "action": [
                        {
                            "order": 0,
                            "output-action": {
                                "max-length": 65535,
                                "output-node-connector": "1"
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

```
    },
    "idle-timeout": 600
  }
]
}
]
```

<http://192.168.56.101:8181/restconf/operational/.opendaylight-inventory:nodes/node/openflow:2/table/0/>

```
{
  "flow-node-inventory:table": [
    {
      "id": 0,
      "opendaylight-flow-table-statistics:flow-table-statistics": {
        "active-flows": 7,
        "packets-looked-up": 12598,
        "packets-matched": 12596
      },
      "flow": [
        {
          "id": "#UF$TABLE*0-5",
          "priority": 2,
          "opendaylight-flow-statistics:flow-statistics": {
            "packet-count": 18,
            "byte-count": 1652,
            "duration": {
              "nanosecond": 641000000,
              "second": 4315
            }
          },
          "table_id": 0,
          "cookie_mask": 0,
          "hard-timeout": 0,
          "match": {
            "in-port": "1"
          },
          "cookie": 3098476543630901000,
          "flags": "",
          "instructions": {
            "instruction": [
              {
                "order": 0,
                "apply-actions": {
                  "action": [
                    {
                      "order": 0,
                      "output-action": {
                        "max-length": 65535,
                        "output-node-connector": "2"
                      }
                    },
                    {
                      "order": 1,
                      "output-action": {
                        "max-length": 65535,
                        "output-node-connector": "3"
                      }
                    }
                  ],
                  "order": 2,
```

```
        "output-action": {
          "max-length": 65535,
          "output-node-connector": "CONTROLLER"
        }
      }
    ]
  }
},
"idle-timeout": 0
},
{
  "id": "L2switch-6",
  "priority": 10,
  "opendaylight-flow-statistics:flow-statistics": {
    "packet-count": 296,
    "byte-count": 28504,
    "duration": {
      "nanosecond": 271000000,
      "second": 285
    }
  },
  "table_id": 0,
  "cookie_mask": 0,
  "hard-timeout": 300,
  "match": {
    "ethernet-match": {
      "ethernet-source": {
        "address": "00:00:00:00:00:02"
      },
      "ethernet-destination": {
        "address": "00:00:00:00:00:01"
      }
    }
  },
  "cookie": 3026418949592973300,
  "flags": "",
  "instructions": {
    "instruction": [
      {
        "order": 0,
        "apply-actions": {
          "action": [
            {
              "order": 0,
              "output-action": {
                "max-length": 65535,
                "output-node-connector": "2"
              }
            }
          ]
        }
      }
    ]
  },
  "idle-timeout": 600
},
{
  "id": "L2switch-7",
  "priority": 10,
  "opendaylight-flow-statistics:flow-statistics": {
    "packet-count": 296,
```

```
"byte-count": 28504,
"duration": {
  "nanosecond": 271000000,
  "second": 285
}
},
"table_id": 0,
"cookie_mask": 0,
"hard-timeout": 300,
"match": {
  "ethernet-match": {
    "ethernet-source": {
      "address": "00:00:00:00:00:01"
    },
    "ethernet-destination": {
      "address": "00:00:00:00:00:02"
    }
  }
},
"cookie": 3026418949592973300,
"flags": "",
"instructions": {
  "instruction": [
    {
      "order": 0,
      "apply-actions": {
        "action": [
          {
            "order": 0,
            "output-action": {
              "max-length": 65535,
              "output-node-connector": "1"
            }
          }
        ]
      }
    }
  ]
},
"idle-timeout": 600
},
{
  "id": "#UF$TABLE*0-2",
  "priority": 100,
  "opendaylight-flow-statistics:flow-statistics": {
    "packet-count": 1728,
    "byte-count": 146880,
    "duration": {
      "nanosecond": 477000000,
      "second": 4319
    }
  },
  "table_id": 0,
  "cookie_mask": 0,
  "hard-timeout": 0,
  "match": {
    "ethernet-match": {
      "ethernet-type": {
        "type": 35020
      }
    }
  }
},
"cookie": 3098476543630901000,
```

```
"flags": "",
"instructions": {
  "instruction": [
    {
      "order": 0,
      "apply-actions": {
        "action": [
          {
            "order": 0,
            "output-action": {
              "max-length": 65535,
              "output-node-connector": "CONTROLLER"
            }
          }
        ]
      }
    }
  ]
},
"idle-timeout": 0
},
{
  "id": "L2switch-2",
  "priority": 0,
  "opendaylight-flow-statistics:flow-statistics": {
    "packet-count": 0,
    "byte-count": 0,
    "duration": {
      "nanosecond": 485000000,
      "second": 4319
    }
  },
  "table_id": 0,
  "cookie_mask": 0,
  "hard-timeout": 0,
  "match": {},
  "cookie": 3098476543630901000,
  "flags": "",
  "idle-timeout": 0
},
{
  "id": "L2switch-3",
  "priority": 2,
  "opendaylight-flow-statistics:flow-statistics": {
    "packet-count": 18,
    "byte-count": 1652,
    "duration": {
      "nanosecond": 640000000,
      "second": 4315
    }
  },
  "table_id": 0,
  "cookie_mask": 0,
  "hard-timeout": 0,
  "match": {
    "in-port": "2"
  },
  "cookie": 3098476543630901000,
  "flags": "",
  "instructions": {
    "instruction": [
      {
        "order": 0,
```

```
"apply-actions": {
  "action": [
    {
      "order": 0,
      "output-action": {
        "max-length": 65535,
        "output-node-connector": "1"
      }
    },
    {
      "order": 1,
      "output-action": {
        "max-length": 65535,
        "output-node-connector": "3"
      }
    },
    {
      "order": 2,
      "output-action": {
        "max-length": 65535,
        "output-node-connector": "CONTROLLER"
      }
    }
  ]
}
],
"idle-timeout": 0
},
{
  "id": "L2switch-4",
  "priority": 2,
  "opendaylight-flow-statistics:flow-statistics": {
    "packet-count": 0,
    "byte-count": 0,
    "duration": {
      "nanosecond": 638000000,
      "second": 4315
    }
  },
  "table_id": 0,
  "cookie_mask": 0,
  "hard-timeout": 0,
  "match": {
    "in-port": "3"
  },
  "cookie": 3098476543630901000,
  "flags": "",
  "instructions": {
    "instruction": [
      {
        "order": 0,
        "apply-actions": {
          "action": [
            {
              "order": 0,
              "output-action": {
                "max-length": 65535,
                "output-node-connector": "1"
              }
            }
          ]
        }
      },
      {

```

```
        "order": 1,
        "output-action": {
            "max-length": 65535,
            "output-node-connector": "2"
        }
    },
    {
        "order": 2,
        "output-action": {
            "max-length": 65535,
            "output-node-connector": "CONTROLLER"
        }
    }
]
},
"idle-timeout": 0
}
]
}
]
```

8 Host isolation with Virtual Tenant Network (VTN)

Virtual Tenant Manager (VTN) will be used along with a custom network topology to create VLANs to set a lab for isolating host traffic between different VLANs. VTN is one of the key modules of ODL. It has many features, such as virtual routers and bridges. An OpenDaylight Plugin that interacts with other modules to implement the components of the VTN model. It also provides a REST interface to configure VTN components in OpenDaylight. VTN Manager is implemented as one plugin to the OpenDaylight. This provides a REST interface to create/update/delete VTN components [22].

Features Overview:

- **odl-vtn-manager** provides VTN Manager’s JAVA API. For creation of virtual bridges
- **odl-vtn-manager-rest** provides VTN Manager’s REST API.

```
opendaylight-user@root>feature:install odl-vtn-manager odl-vtn-manager-rest
opendaylight-user@root>feature:list | grep vtn
odl-vtn-manager-rest          | 0.7.2          | x      | Started | odl-vtn-
manager-rest                  | OpenDaylight :: VTN Manager :: REST API
odl-vtn-manager               | 0.7.2          | x      | Started | vtn-mana
ger-0.7.2                     | OpenDaylight :: VTN Manager :: Java API
features-vtn-manager          | 0.7.2          |        | Uninstalled | features
-vtn-manager                  | features-vtn-manager
odl-vtn-manager-neutron       | 0.7.2          |        | Uninstalled | odl-vtn-
manager-neutron               | OpenDaylight :: VTN Manager :: Neutron Interface
opendaylight-user@root>
```

Setting up the environment in Mininet

```

sdn@sdn-opendaylight:~$ ifconfig
enp0s3  Link encap:Ethernet  HWaddr 08:00:27:f3:2a:7b
        inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
        inet6 addr: fe80::3982:5697:585c:259a/64  Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:1122450 errors:0 dropped:0 overruns:0 frame:0
        TX packets:403262 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:1090508082 (1.0 GB)  TX bytes:25336006 (25.3 MB)

enp0s8  Link encap:Ethernet  HWaddr 08:00:27:d1:47:dd
        inet addr:192.168.56.101  Bcast:192.168.56.255  Mask:255.255.255.0
        inet6 addr: fe80::8862:802d:5ece:411/64  Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:14570 errors:0 dropped:0 overruns:0 frame:0
        TX packets:3412 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:5906832 (5.9 MB)  TX bytes:483264 (483.2 KB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128  Scope:Host
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:4336 errors:0 dropped:0 overruns:0 frame:0
        TX packets:4336 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:6665958 (6.6 MB)  TX bytes:6665958 (6.6 MB)

sdn@sdn-opendaylight:~$

```

```

-rw-rw-r-- 1 mininet mininet 21504 Mar 21 2017 util.py
-rw-r--r-- 1 root root 21872 Mar 21 2017 util.pyc
-rw-rw-r-- 1 mininet mininet 2356 Sep 19 00:48 vlan.py
mininet@mininet-vm:~/mininet/mininet$
mininet@mininet-vm:~/mininet/mininet$
mininet@mininet-vm:~/mininet/mininet$
mininet@mininet-vm:~/mininet/mininet$
mininet@mininet-vm:~/mininet/mininet$ cd
mininet@mininet-vm:~$ ifocnfig
No command 'ifocnfig' found, did you mean:
  Command 'ifconfig' from package 'net-tools' (main)
ifocnfig: command not found
mininet@mininet-vm:~$ ifconfig
eth0    Link encap:Ethernet  HWaddr 08:00:27:53:75:d9
        inet addr:192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:9468 errors:0 dropped:0 overruns:0 frame:0
        TX packets:4480 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:2267336 (2.2 MB)  TX bytes:4957528 (4.9 MB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:4645 errors:0 dropped:0 overruns:0 frame:0
        TX packets:4645 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:2733100 (2.7 MB)  TX bytes:2733100 (2.7 MB)

mininet@mininet-vm:~$

```

VLANS are configured between the h1,h2,h2 with VLAN id =100 and VLAN id = 200 for h4,h5,h6.

```
#!/usr/bin/python
```

```
from mininet.node import Host, RemoteController
```

```
from mininet.topo import Topo
```

```
import apt
```

```
#package check Start
```

```
cache = apt.Cache()
```

```
if cache['vlan'].is_installed:
```

```
print "Vlan installed"
```

```
else:
```

```
print "ERROR:VLAN package not installed please run sudo apt-get install vlan"
```

```
exit(1)

#package check End

class VLANHost( Host ):
    def config( self, vlan=1, **params ):
        """Configure VLANHost according to (optional) parameters:
        vlan: VLAN ID for default interface"""
        r = super( Host, self ).config( **params )
        intf = self.defaultIntf()
        # remove IP from default, "physical" interface
        self.cmd( 'ifconfig %s inet 0' % intf )
        # create VLAN interface
        self.cmd( 'vconfig add %s %d' % ( intf, vlan ) )
        # assign the host's IP to the VLAN interface
        self.cmd( 'ifconfig %s.%d inet %s' % ( intf, vlan, params['ip'] ) )
        # update the intf name and host's intf map
        newName = '%s.%d' % ( intf, vlan )
        # update the (Mininet) interface to refer to VLAN interface name
        intf.name = newName
        # add VLAN interface to host's name to intf map
        self.nameToIntf[ newName ] = intf
        return r

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

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

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        host1=self.addHost( 'h1', cls=VLANHost, vlan=100)
        host2=self.addHost( 'h2', cls=VLANHost, vlan=200)
        host3=self.addHost( 'h3', cls=VLANHost, vlan=100)
        host4=self.addHost( 'h4', cls=VLANHost, vlan=200)
        host5=self.addHost( 'h5', cls=VLANHost, vlan=100)
        host6=self.addHost( 'h6', cls=VLANHost, vlan=200)
```

```
s1 = self.addSwitch( 's1' )
s2 = self.addSwitch( 's2' )
s3 = self.addSwitch( 's3' )
```

```
self.addLink(s1,host1)
```

```
self.addLink(s1,host2)
```

```
self.addLink(s1,s2)
```

```
self.addLink(s2,host3)
```

```
self.addLink(s2,host4)
```

```
self.addLink(s2,s3)
```

```
self.addLink(s3,host5)
```

```
self.addLink(s3,host6)
```

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

```
sdn@sdn-opendaylight:~$ scp vlan.py mininet@192.168.56.102:~/mininet/mininet/
mininet@192.168.56.102's password:
vlan.py                                100% 2229      2.2KB/s   00:00
sdn@sdn-opendaylight:~$
```

```
total 420
drwxrwxr-x  3 mininet mininet  4096 Sep 19 00:48 ./
drwxrwxr-x 13 mininet mininet  4096 Mar 21 2017 ../
-rwxrwxr-x  1 mininet mininet  4349 Mar 21 2017 clean.py*
-rw-rw-r--  1 mininet mininet 15900 Mar 21 2017 cli.py
-rw-r--r--  1 root    root    16898 Mar 21 2017 cli.pyc
lrwxrwxrwx  1 mininet mininet   11 Mar 21 2017 examples -> ../examples/
-rw-rw-r--  1 mininet mininet   71 Mar 21 2017 __init__.py
-rw-r--r--  1 root    root     219 Mar 21 2017 __init__.pyc
-rw-rw-r--  1 mininet mininet 22037 Mar 21 2017 link.py
-rw-r--r--  1 root    root    19619 Mar 21 2017 link.pyc
-rw-rw-r--  1 mininet mininet  6078 Mar 21 2017 log.py
-rw-r--r--  1 root    root     5951 Mar 21 2017 log.pyc
-rw-rw-r--  1 mininet mininet  2469 Mar 21 2017 moduledeps.py
-rw-r--r--  1 root    root     2900 Mar 21 2017 moduledeps.pyc
-rwxrwxr-x  1 mininet mininet 36864 Mar 21 2017 net.py*
-rw-r--r--  1 root    root    33427 Mar 21 2017 net.pyc
-rw-rw-r--  1 mininet mininet  5539 Mar 21 2017 nodelib.py
-rw-r--r--  1 root    root     5832 Mar 21 2017 nodelib.pyc
-rw-rw-r--  1 mininet mininet 60242 Mar 21 2017 node.py
-rw-r--r--  1 root    root    58065 Mar 21 2017 node.pyc
-rw-rw-r--  1 mininet mininet  2776 Mar 21 2017 term.py
-rw-r--r--  1 root    root     3140 Mar 21 2017 term.pyc
drwxrwxr-x  2 mininet mininet  4096 Mar 21 2017 test/
-rw-rw-r--  1 mininet mininet  2912 Mar 21 2017 toplib.py
-rw-rw-r--  1 mininet mininet 12536 Mar 21 2017 topo.py
-rw-rw-r--  1 mininet mininet 21504 Mar 21 2017 util.py
-rw-r--r--  1 root    root    21872 Mar 21 2017 util.pyc
-rw-rw-r--  1 mininet mininet  2356 Sep 19 00:48 vlan.py
mininet@mininet-vm:~/mininet/mininet$
```

Connection from ODL_VM to MININET VM to forward the X server

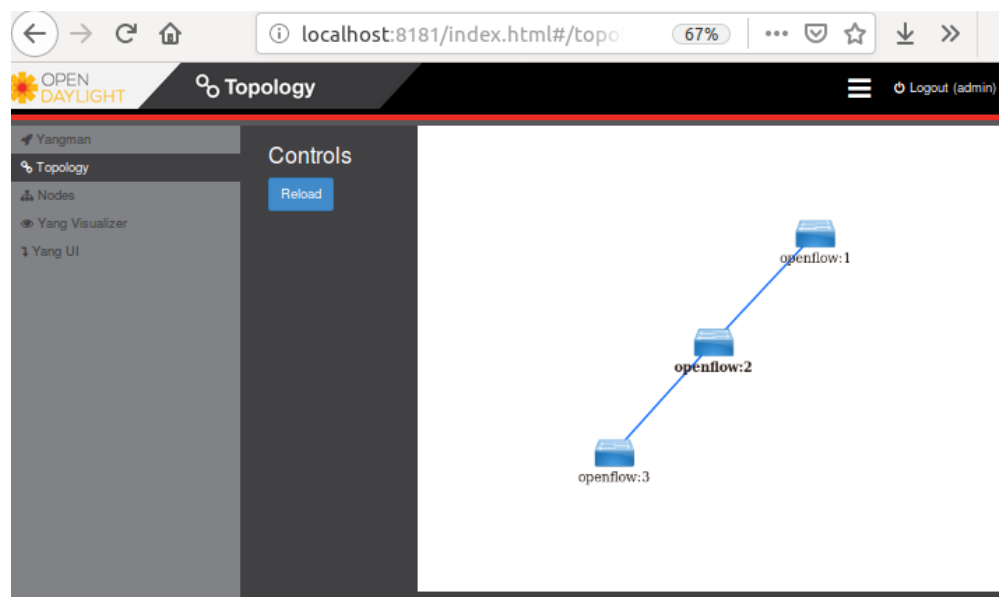
```
sdn@sdn-opendaylight:~$ ssh -X mininet@192.168.56.102
mininet@192.168.56.102's password:
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic i686)

 * Documentation:  https://help.ubuntu.com/
Last login: Fri Sep 20 22:56:47 2019
/usr/bin/xauth:  file /home/mininet/.Xauthority does not exist
mininet@mininet-vm:~$ ll
total 72
drwxr-xr-x 10 mininet mininet 4096 Sep 21 00:00 ./
drwxr-xr-x  3 root    root    4096 Aug  3 2016 ../
-rw-r----- 1 mininet mininet  56 Sep 21 00:00 .Xauthority
-rw-r----- 1 mininet mininet  408 Mar 21 2017 .bash_history
-rw-r--r--  1 mininet mininet  220 Aug  3 2016 .bash_logout
-rw-r--r--  1 mininet mininet 3655 Mar 21 2017 .bashrc
drwx----- 2 mininet mininet 4096 Mar 21 2017 .cache/
-rw-rw-r--  1 mininet mininet  85 Mar 21 2017 .gitconfig
-rw-r--r--  1 mininet mininet  675 Aug  3 2016 .profile
-rw-r----- 1 root    root    1024 Mar 21 2017 .rnd
drwxrwxr-x  2 mininet mininet 4096 Mar 21 2017 .wireshark/
-rw-rw-r--  1 mininet mininet 1630 Mar 21 2017 install-mininet-vm.sh
drwxrwxr-x 17 mininet mininet 4096 Mar 21 2017 loxigen/
drwxrwxr-x 13 mininet mininet 4096 Mar 21 2017 mininet/
drwxrwxr-x 14 mininet mininet 4096 Mar 21 2017 oflops/
drwxrwxr-x 11 mininet mininet 4096 Mar 21 2017 oftest/
drwxrwxr-x 19 mininet mininet 4096 Mar 21 2017 openflow/
drwxrwxr-x  7 mininet mininet 4096 Mar 21 2017 pox/
mininet@mininet-vm:~$ cd
```

```
sudo mn --controller=remote,ip=192.168.56.101 --custom ~/mininet/mininet/vlan.py --topo
simplevlan --mac --switch ovsk,protocols=Openflow13
```

```
mininet@mininet-vm:~$ sudo mn --controller=remote,ip=192.168.56.101 --custom ~/mininet/min
inet/vlan.py --topo simplevlan --mac --switch ovsk,protocols=OpenFlow13
Vlan installed
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.56.101:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, h1) (s1, h2) (s2, h3) (s2, h4) (s2, s3) (s3, h5) (s3, h6)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
```

```
mininet> net
h1 h1-eth0.100:s1-eth1
h2 h2-eth0.200:s1-eth2
h3 h3-eth0.100:s2-eth2
h4 h4-eth0.200:s2-eth3
h5 h5-eth0.100:s3-eth2
h6 h6-eth0.200:s3-eth3
s1 lo: s1-eth1:h1-eth0.100 s1-eth2:h2-eth0.200 s1-eth3:s2-eth1
s2 lo: s2-eth1:s1-eth3 s2-eth2:h3-eth0.100 s2-eth3:h4-eth0.200 s2-eth4:s3-eth1
s3 lo: s3-eth1:s2-eth4 s3-eth2:h5-eth0.100 s3-eth3:h6-eth0.200
c0
mininet> links
s1-eth1<->h1-eth0.100 (OK OK)
s1-eth2<->h2-eth0.200 (OK OK)
s1-eth3<->s2-eth1 (OK OK)
s2-eth2<->h3-eth0.100 (OK OK)
s2-eth3<->h4-eth0.200 (OK OK)
s2-eth4<->s3-eth1 (OK OK)
s3-eth2<->h5-eth0.100 (OK OK)
s3-eth3<->h6-eth0.200 (OK OK)
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 h6 s1 s2 s3
mininet>
```



The screenshot shows the OpenDaylight web interface at localhost:8181/index.html#/node/index. The 'Nodes' section is active, displaying a table of node details. The table has columns for Node Id, Node Name, Node Connectors, and Statistics. The data is as follows:

Node Id	Node Name	Node Connectors	Statistics
openflow:2	None	5	Flows Node Connectors
openflow:3	None	4	Flows Node Connectors
openflow:1	None	4	Flows Node Connectors

```
mininet> dpctl dump-flows -O OpenFlow13
*** s1 ***
OFPST FLOW reply (OF1.3) (xid=0x2):
 cookie=0x2b00000000000007, duration=2220.909s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=3 actions=output:2,output:1
 cookie=0x2b00000000000009, duration=2220.908s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=1 actions=output:3,output:2,CONTROLLER:65535
 cookie=0x2b00000000000008, duration=2220.909s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=2 actions=output:3,output:1,CONTROLLER:65535
 cookie=0x2b00000000000001, duration=2224.871s, table=0, n_packets=446, n_bytes=37910, priority=100,d_l_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000000, duration=2224.871s, table=0, n_packets=0, n_bytes=0, priority=0 actions=drop
*** s2 ***
OFPST FLOW reply (OF1.3) (xid=0x2):
 cookie=0x2b00000000000000, duration=2220.925s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=3 actions=output:2,output:1,output:4,CONTROLLER:65535
 cookie=0x2b00000000000002, duration=2220.918s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=1 actions=output:3,output:2,output:4
 cookie=0x2b00000000000003, duration=2220.916s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=4 actions=output:3,output:2,output:1
 cookie=0x2b00000000000001, duration=2220.923s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=2 actions=output:3,output:1,output:4,CONTROLLER:65535
 cookie=0x2b00000000000002, duration=2224.874s, table=0, n_packets=892, n_bytes=75820, priority=100,d_l_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000002, duration=2224.874s, table=0, n_packets=0, n_bytes=0, priority=0 actions=drop
*** s3 ***
OFPST FLOW reply (OF1.3) (xid=0x2):
 cookie=0x2b00000000000006, duration=2220.914s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=3 actions=output:2,output:1,CONTROLLER:65535
 cookie=0x2b00000000000005, duration=2220.916s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=1 actions=output:2,output:3
 cookie=0x2b00000000000004, duration=2220.917s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=2 actions=output:1,output:3,CONTROLLER:65535
 cookie=0x2b00000000000000, duration=2224.874s, table=0, n_packets=446, n_bytes=37910, priority=100,d_l_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000001, duration=2224.874s, table=0, n_packets=0, n_bytes=0, priority=0 actions=drop
mininet>
```

Setting up the VTN modules in OpenDaylight

opendaylight-user@root>feature:install odl-vtn-manager odl-vtn-manager-rest

VTN features loaded to ODL restconf API.

vtn(2015-03-28)	Show/Hide	List Operations	Expand Operations	Raw
vtn-config(2015-02-09)	Show/Hide	List Operations	Expand Operations	Raw
vtn-flow(2015-04-10)	Show/Hide	List Operations	Expand Operations	Raw
vtn-flow-condition(2015-03-13)	Show/Hide	List Operations	Expand Operations	Raw
vtn-flow-filter(2015-09-07)	Show/Hide	List Operations	Expand Operations	Raw
vtn-flow-impl(2015-03-13)	Show/Hide	List Operations	Expand Operations	Raw
vtn-inventory(2015-02-09)	Show/Hide	List Operations	Expand Operations	Raw
vtn-mac-map(2015-09-07)	Show/Hide	List Operations	Expand Operations	Raw
vtn-mac-table(2015-09-07)	Show/Hide	List Operations	Expand Operations	Raw
vtn-mapping(2015-10-01)	Show/Hide	List Operations	Expand Operations	Raw
vtn-path-map(2015-03-28)	Show/Hide	List Operations	Expand Operations	Raw
vtn-path-policy(2015-02-09)	Show/Hide	List Operations	Expand Operations	Raw
vtn-port-map(2015-09-07)	Show/Hide	List Operations	Expand Operations	Raw
vtn-static-topology(2015-08-01)	Show/Hide	List Operations	Expand Operations	Raw
vtn-topology(2015-02-09)	Show/Hide	List Operations	Expand Operations	Raw
vtn-vbridge(2015-09-07)	Show/Hide	List Operations	Expand Operations	Raw
vtn-version(2015-09-01)	Show/Hide	List Operations	Expand Operations	Raw
vtn-vinterface(2015-09-07)	Show/Hide	List Operations	Expand Operations	Raw
vtn-vlan-map(2015-09-07)	Show/Hide	List Operations	Expand Operations	Raw
vtn-vterminal(2015-09-07)	Show/Hide	List Operations	Expand Operations	Raw

<http://192.168.56.101:8181/restconf/operational/vtn-inventory:vtn-nodes>

```
{
  "vtn-nodes":{
    "vtn-node":[
      {
        "id": "openflow:3",
        "openflow-version": "OF13",
        "vtn-port": [
          {
            "id": "openflow:3:3",
            "cost": 1000,
            "enabled": true,
            "name": "s3-eth3"
          },
          {
            "id": "openflow:3:2",
            "cost": 1000,
            "enabled": true,
            "name": "s3-eth2"
          },
          {
            "id": "openflow:3:1",
            "cost": 1000,
            "enabled": true,
            "name": "s3-eth1",
            "port-link": [
              {
                "link-id": "openflow:2:4",
                "peer": "openflow:2:4"
              },
              {
                "link-id": "openflow:3:1",
                "peer": "openflow:2:4"
              }
            ]
          }
        ]
      }
    ]
  }
}
```

```
    ]
  }
]
},
{
  "id": "openflow:2",
  "openflow-version": "OF13",
  "vtn-port": [
    {
      "id": "openflow:2:4",
      "cost": 1000,
      "enabled": true,
      "name": "s2-eth4",
      "port-link": [
        {
          "link-id": "openflow:2:4",
          "peer": "openflow:3:1"
        },
        {
          "link-id": "openflow:3:1",
          "peer": "openflow:3:1"
        }
      ]
    },
  ],
  {
    "id": "openflow:2:3",
    "cost": 1000,
    "enabled": true,
    "name": "s2-eth3"
  },
  {
    "id": "openflow:2:2",
    "cost": 1000,
    "enabled": true,
    "name": "s2-eth2"
  },
  {
    "id": "openflow:2:1",
    "cost": 1000,
    "enabled": true,
    "name": "s2-eth1",
    "port-link": [
      {
        "link-id": "openflow:2:1",
        "peer": "openflow:1:3"
      },
      {
        "link-id": "openflow:1:3",
        "peer": "openflow:1:3"
      }
    ]
  }
]
}
```

```
},
{
  "id":"openflow:1",
  "openflow-version":"OF13",
  "vtn-port":[
    {
      "id":"openflow:1:3",
      "cost":1000,
      "enabled":true,
      "name":"s1-eth3",
      "port-link":[
        {
          "link-id":"openflow:2:1",
          "peer":"openflow:2:1"
        },
        {
          "link-id":"openflow:1:3",
          "peer":"openflow:2:1"
        }
      ]
    }
  ],
},
{
  "id":"openflow:1:2",
  "cost":1000,
  "enabled":true,
  "name":"s1-eth2"
},
{
  "id":"openflow:1:1",
  "cost":1000,
  "enabled":true,
  "name":"s1-eth1"
}
]
}
]
```

Using the Rest API to create a virtual tenant, virtual bridge and vlan mapping

vtn(2015-03-28)

Show/Hide | List Operations | Expand Operations | Raw

GET /operational/vtn:vtns

Implementation Notes

The root container of all VTNs (Virtual Tenant Network). Note that the VTN configuration must be modified by RPC. Do not edit this container directly.

Response Class

Model | Model Schema

(operational)vtns

Response Content Type

application/json

Try it out! | Hide Response

Request URL

http://192.168.56.101:8181/restconf/operational/vtn:vtns

Response Body

```
{
  "vtns": {}
}
```

Response Code

200

Response Headers

```
{
  "Expires": "Thu, 01 Jan 1970 00:00:00 GMT",
  "Content-Encoding": "gzip",
  "Transfer-Encoding": "chunked",
  "Vary": "Accept-Encoding, User-Agent",
  "Content-Type": "application/json"
}
```

POST /operations/vtn:remove-vtn

POST /operations/vtn:update-vtn

Implementation Notes

Create or modify the VTN specified by the name. Upon successful completion, the result of this operation will be set to the `status` field in the RPC output. `CREATED` indicates that the specified VTN has been newly created. `CHANGED` indicates that the configuration for the specified VTN has been changed. Null indicates that the configuration for the specified VTN was not changed. On failure, one of vtn-error-tag value which indicates the cause of error is set into application tag in RPC error. `BADREQUEST` is set if the RPC input contains invalid data. `CONFLICT` is set if `update-mode` field in the RPC input is `CREATE` and the specified VTN is present. `NOTFOUND` is set if `update-mode` field in the RPC input is `MODIFY` and the specified VTN is not present. `INTERNALERROR` is set if the operation failed due to internal error.

Response Class

Model | Model Schema

(update-vtn)output-TOP {
 vtn:output (object[(update-vtn)output], optional)
}
(update-vtn)output {
 vtn:status (CREATED, optional) = ["CREATED" or "REMOVED" or "CHANGED"]: Describes the status of the RPC result.
}

Response Content Type

application/json

Payload retrieved from Model Schema. Define your own values for the attributes. Not all attributes are mandatory.

Parameters

Parameter	Value	Description	Parameter Type	Data Type
(update-vtn)input-TOP	<pre>{ "vtn:input": { "vtn:tenant-name": "Some tenant-name", "vtn:update-mode": "UPDATE", "vtn:operation": "ADD", "vtn:description": "Some description", "vtn:idle-timeout": "0", "vtn:hard-timeout": "0" } }</pre>		body	Model Model Schema

Parameter content type:

application/json

Try it out!

Model Schema

```
{
  "vtn:input": {
    "vtn:tenant-name": "Some tenant-name",
    "vtn:update-mode": "UPDATE",
    "vtn:operation": "ADD",
    "vtn:description": "Some description",
    "vtn:idle-timeout": "0",
    "vtn:hard-timeout": "0"
  }
}
```

Click to set as parameter value

```
{
  "vtn:input": {
    "vtn:tenant-name": "tenant_sdn",
    "vtn:description": "Virtual tenant"
  }
}
```

75 of 84

Parameter

Value

(update-vtn)input-TOP

```
{
  "vtn:input": {
    "vtn:tenant-name": "tenant_sdn",
    "vtn:description": "Virtual tenant"
  }
}
```

Parameter content type:

application/json

Try it out!

[Hide Response](#)

Request URL

http://localhost:8181/restconf/operations/vtn:update-vtn

Response Body

```
{
  "output": {
    "status": "CREATED"
  }
}
```

Response Code

200

Request URL

http://localhost:8181/restconf/operational/vtn:vtns

Response Body

```
{
  "vtns": {
    "vtn": [
      {
        "name": "tenant_sdn",
        "vtenant-config": {
          "hard-timeout": 0,
          "description": "Virtual tenant",
          "idle-timeout": 300
        }
      }
    ]
  }
}
```

Response Code

200

Creating virtuall bridges bridge_sdn1 bridge_sdn2

```
{
  "vtn-vbridge:input": {
    "vtn-vbridge:tenant-name": "tenant_sdn",
    "vtn-vbridge:bridge-name": "bridge_sdn1"
  }
}
```

(update-vbridge)input-TOP

```
{
  "xtn-vbridge:input": {
    "xtn-vbridge:tenant-name": "tenant_sdn",
    "xtn-vbridge:bridge-name": "bridge_sdn1"
  }
}
```

Parameter content type: application/json

Try it out! Hide Response

Request URL

http://localhost:8181/restconf/operations/vtn-vbridge:update-vbridge

Response Body

```
{
  "output": {
    "status": "CREATED"
  }
}
```

(update-vbridge)input-TOP

```
{
  "vtn-vbridge:input": {
    "vtn-vbridge:tenant-name": "tenant_sdn",
    "vtn-vbridge:bridge-name": "bridge_sdn2"
  }
}
```

Parameter content type: application/json

Try it out! Hide Response

Request URL

http://localhost:8181/restconf/operations/vtn-vbridge:update-vbridge

Response Body

```
{
  "output": {
    "status": "CREATED"
  }
}
```

Request URL

http://localhost:8181/restconf/operational/vtn:vtns

Response Body

```
{
  "vtns": {
    "vtn": [
      {
        "name": "tenant_sdn",
        "vtenant-config": {
          "hard-timeout": 0,
          "description": "Virtual tenant",
          "idle-timeout": 300
        },
        "vbridge": [
          {
            "name": "bridge_sdn1",
            "vbridge-config": {
              "age-interval": 600
            },
            "bridge-status": {
              "path-faults": 0,
              "state": "UNKNOWN"
            }
          }
        ]
      }
    ]
  }
}
```

Response Code

200

```
    "bridge-status": {
      "path-faults": 0,
      "state": "UNKNOWN"
    }
  },
  {
    "name": "bridge_sdn2",
    "vbridge-config": {
      "age-interval": 600
    },
    "bridge-status": {
      "path-faults": 0,
      "state": "UNKNOWN"
    }
  }
]
}
```

Response Code

200

Mapping vlan-id 100 to bridge_sdn1 and vlan- id 200 to bridge_sdn2 respectively.

```
{
  "vtn-vlan-map:input": {
    "vtn-vlan-map:tenant-name": "tenant_sdn",
    "vtn-vlan-map:bridge-name": "bridge_sdn1",
    "vtn-vlan-map:vlan-id": "100"
  }
}
```

(add-vlan-map)input-TOP

```
{
  "xtn-vlan-map:input": {
    "xtn-vlan-map:tenant-name": "tenant_sdn",
    "xtn-vlan-map:bridge-name": "bridge_sdn1",
    "xtn-vlan-map:vlan-id": "100"
  }
}
```

Parameter content type: application/json

Try it out! Hide Response

Request URL

http://localhost:8181/restconf/operations/vtn-vlan-map:add-vlan-map

Response Body

```
{
  "output": {
    "active": true,
    "map-id": "ANY.100"
  }
}
```

(add-vlan-map)input-TOP

```
{
  "xtn-vlan-map:input": {
    "xtn-vlan-map:tenant-name": "tenant_sdn",
    "xtn-vlan-map:bridge-name": "bridge_sdn2",
    "xtn-vlan-map:vlan-id": "200"
  }
}
```

Parameter content type: application/json

Try it out! Hide Response

Request URL

http://localhost:8181/restconf/operations/vtn-vlan-map:add-vlan-map

Response Body

```
{
  "output": {
    "active": true,
    "map-id": "ANY.200"
  }
}
```

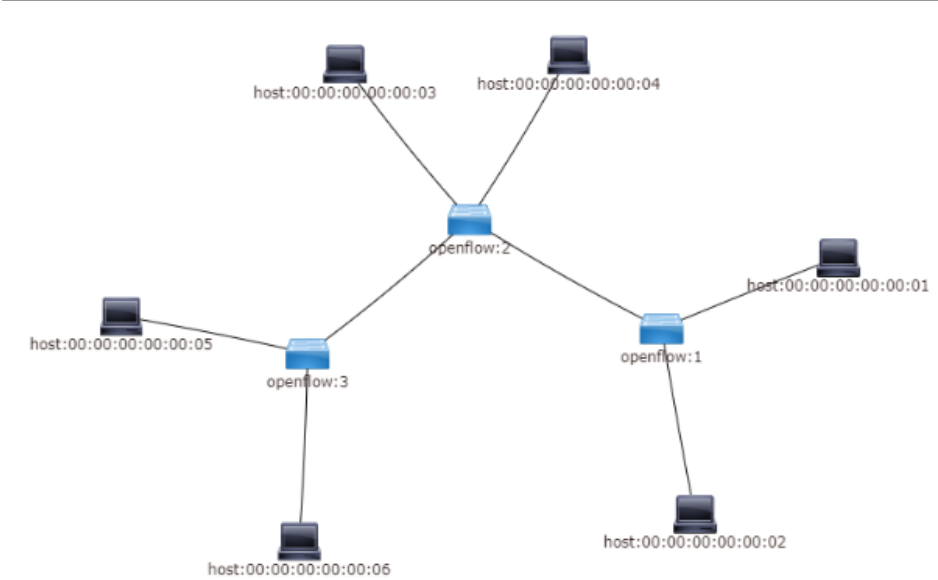
77 of 84

After mapping

Request URL	Request URL
http://localhost:8181/restconf/operational/vtn:vtns	http://localhost:8181/restconf/operational/vtn:vtns
Response Body	Response Body
<pre>{ "name": "bridge_sdn1", "vbridge-config": { "age-interval": 600 }, "bridge-status": { "path-faults": 0, "state": "UP" }, "vlan-map": [{ "map-id": "ANY.100", "vlan-map-config": { "vlan-id": 100 }, "vlan-map-status": { "active": true } }] }</pre>	<pre> "name": "bridge_sdn2", "vbridge-config": { "age-interval": 600 }, "bridge-status": { "path-faults": 0, "state": "UP" }, "vlan-map": [{ "map-id": "ANY.200", "vlan-map-config": { "vlan-id": 200 }, "vlan-map-status": { "active": true } }] } }</pre>

Testing via pingall commad.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 X h5 X
h2 -> X X h4 X h6
h3 -> h1 X X h5 X
h4 -> X h2 X X h6
h5 -> h1 X h3 X X
h6 -> X h2 X h4 X
*** Results: 60% dropped (12/30 received)
mininet>
```



```
*** s1 -----
OFPST FLOW reply (OF1.3) (xid=0x2):
  cookie=0x7f56000000000004, duration=144.839s, table=0, n_packets=2, n_bytes=148, idle tim
eout=300, send_flow_rem priority=10,in_port=1,d_l_vlan=100,d_l_src=00:00:00:00:00:01,d_l_dst=
00:00:00:00:00:05 actions=output:3
  cookie=0x7f56000000000006, duration=135.857s, table=0, n_packets=2, n_bytes=148, idle tim
eout=300, send_flow_rem priority=10,in_port=2,d_l_vlan=200,d_l_src=00:00:00:00:00:02,d_l_dst=
00:00:00:00:00:04 actions=output:3
  cookie=0x7f56000000000005, duration=135.883s, table=0, n_packets=4, n_bytes=296, send flo
w_rem priority=10,in_port=3,d_l_vlan=200,d_l_src=00:00:00:00:00:04,d_l_dst=00:00:00:00:00:02
actions=output:2
  cookie=0x7f56000000000008, duration=132.84s, table=0, n_packets=2, n_bytes=148, idle time
out=300, send_flow_rem priority=10,in_port=2,d_l_vlan=200,d_l_src=00:00:00:00:00:02,d_l_dst=0
0:00:00:00:00:06 actions=output:3
  cookie=0x7f56000000000001, duration=147.607s, table=0, n_packets=4, n_bytes=296, send flo
w_rem priority=10,in_port=3,d_l_vlan=100,d_l_src=00:00:00:00:00:03,d_l_dst=00:00:00:00:00:01
actions=output:1
  cookie=0x7f56000000000002, duration=147.603s, table=0, n_packets=2, n_bytes=148, idle tim
eout=300, send_flow_rem priority=10,in_port=1,d_l_vlan=100,d_l_src=00:00:00:00:00:01,d_l_dst=
00:00:00:00:00:03 actions=output:3
  cookie=0x7f56000000000003, duration=144.873s, table=0, n_packets=4, n_bytes=296, send flo
w_rem priority=10,in_port=3,d_l_vlan=100,d_l_src=00:00:00:00:00:05,d_l_dst=00:00:00:00:00:01
actions=output:1
  cookie=0x7f56000000000007, duration=132.872s, table=0, n_packets=4, n_bytes=296, send flo
w_rem priority=10,in_port=3,d_l_vlan=200,d_l_src=00:00:00:00:00:06,d_l_dst=00:00:00:00:00:02
actions=output:2
  cookie=0x2b0000000000001b, duration=702.487s, table=0, n_packets=52, n_bytes=2840, priori
ty=2,in_port=3 actions=output:2,output:1
  cookie=0x2b0000000000001d, duration=702.485s, table=0, n_packets=17, n_bytes=894, priorit
y=2,in_port=1 actions=output:3,output:2,CONTROLLER:65535
  cookie=0x2b0000000000001c, duration=702.486s, table=0, n_packets=13, n_bytes=710, priorit
y=2,in_port=2 actions=output:3,output:1,CONTROLLER:65535
  cookie=0x2b00000000000007, duration=708.358s, table=0, n_packets=946, n_bytes=80410, prio
rity=100,d_l_type=0x88cc actions=CONTROLLER:65535
  cookie=0x7f57ffffffffffff, duration=708.354s, table=0, n_packets=0, n_bytes=0, send_flow_
rem priority=0 actions=CONTROLLER:65535

OFPST FLOW reply (OF1.3) (xid=0x2):
  cookie=0x7f56000000000004, duration=144.842s, table=0, n_packets=2, n_bytes=148, send_flow_rem priorit
y=10,in_port=1,d_l_vlan=100,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:05 actions=output:4
  cookie=0x7f56000000000006, duration=135.86s, table=0, n_packets=2, n_bytes=148, send_flow_rem priority
=10,in_port=1,d_l_vlan=200,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:04 actions=output:3
  cookie=0x7f56000000000003, duration=144.875s, table=0, n_packets=4, n_bytes=296, send_flow_rem priorit
y=10,in_port=4,d_l_vlan=100,d_l_src=00:00:00:00:00:05,d_l_dst=00:00:00:00:00:01 actions=output:1
  cookie=0x7f5600000000000a, duration=126.843s, table=0, n_packets=2, n_bytes=148, idle timeout=300, sen
d_flow_rem priority=10,in_port=2,d_l_vlan=100,d_l_src=00:00:00:00:00:03,d_l_dst=00:00:00:00:00:05 actions=
output:4
  cookie=0x7f56000000000001, duration=147.608s, table=0, n_packets=4, n_bytes=296, idle timeout=300, sen
d_flow_rem priority=10,in_port=2,d_l_vlan=100,d_l_src=00:00:00:00:00:03,d_l_dst=00:00:00:00:00:01 actions=
output:1
  cookie=0x7f56000000000005, duration=135.884s, table=0, n_packets=4, n_bytes=296, idle timeout=300, sen
d_flow_rem priority=10,in_port=3,d_l_vlan=200,d_l_src=00:00:00:00:00:04,d_l_dst=00:00:00:00:00:02 actions=
output:1
  cookie=0x7f56000000000007, duration=132.874s, table=0, n_packets=4, n_bytes=296, send_flow_rem priorit
y=10,in_port=4,d_l_vlan=200,d_l_src=00:00:00:00:00:06,d_l_dst=00:00:00:00:00:02 actions=output:1
  cookie=0x7f5600000000000b, duration=114.889s, table=0, n_packets=4, n_bytes=296, send_flow_rem priorit
y=10,in_port=4,d_l_vlan=200,d_l_src=00:00:00:00:00:06,d_l_dst=00:00:00:00:00:04 actions=output:3
  cookie=0x7f56000000000009, duration=126.881s, table=0, n_packets=4, n_bytes=296, send_flow_rem priorit
y=10,in_port=4,d_l_vlan=100,d_l_src=00:00:00:00:00:05,d_l_dst=00:00:00:00:00:03 actions=output:2
  cookie=0x7f56000000000008, duration=132.842s, table=0, n_packets=2, n_bytes=148, send_flow_rem priorit
y=10,in_port=1,d_l_vlan=200,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:06 actions=output:4
  cookie=0x7f56000000000002, duration=147.607s, table=0, n_packets=2, n_bytes=148, send_flow_rem priorit
y=10,in_port=1,d_l_vlan=100,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:03 actions=output:2
  cookie=0x7f5600000000000c, duration=114.841s, table=0, n_packets=2, n_bytes=148, idle timeout=300, sen
d_flow_rem priority=10,in_port=3,d_l_vlan=200,d_l_src=00:00:00:00:00:04,d_l_dst=00:00:00:00:00:06 actions=
output:4
  cookie=0x2b00000000000014, duration=702.5s, table=0, n_packets=13, n_bytes=710, priority=2,in_port=3 a
ctions=output:2,output:1,output:4,CONTROLLER:65535
  cookie=0x2b00000000000016, duration=702.499s, table=0, n_packets=26, n_bytes=1420, priority=2,in_port=
1 actions=output:3,output:2,output:4
  cookie=0x2b00000000000017, duration=702.494s, table=0, n_packets=26, n_bytes=1420, priority=2,in_port=
4 actions=output:3,output:2,output:1
  cookie=0x2b00000000000015, duration=702.5s, table=0, n_packets=15, n_bytes=858, priority=2,in_port=2 a
ctions=output:3,output:1,output:4,CONTROLLER:65535
  cookie=0x2b00000000000008, duration=708.284s, table=0, n_packets=1892, n_bytes=160820, priority=100,d_l
_type=0x88cc actions=CONTROLLER:65535
  cookie=0x7f57ffffffffffff, duration=708.259s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=
0 actions=CONTROLLER:65535

*** s3 -----
OFPST FLOW reply (OF1.3) (xid=0x2):
  cookie=0x7f56000000000004, duration=144.844s, table=0, n_packets=2, n_bytes=148, send flo
w_rem priority=10,in_port=1,d_l_vlan=100,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:05
actions=output:2
  cookie=0x7f56000000000009, duration=126.882s, table=0, n_packets=4, n_bytes=296, idle tim
eout=300, send_flow_rem priority=10,in_port=2,d_l_vlan=100,d_l_src=00:00:00:00:00:05,d_l_dst=
00:00:00:00:00:03 actions=output:1
  cookie=0x7f5600000000000b, duration=114.891s, table=0, n_packets=4, n_bytes=296, idle tim
eout=300, send_flow_rem priority=10,in_port=3,d_l_vlan=200,d_l_src=00:00:00:00:00:06,d_l_dst=
00:00:00:00:00:04 actions=output:1
  cookie=0x7f56000000000003, duration=144.876s, table=0, n_packets=4, n_bytes=296, idle tim
eout=300, send_flow_rem priority=10,in_port=2,d_l_vlan=100,d_l_src=00:00:00:00:00:05,d_l_dst=
00:00:00:00:00:01 actions=output:1
  cookie=0x7f56000000000008, duration=132.844s, table=0, n_packets=2, n_bytes=148, send flo
w_rem priority=10,in_port=1,d_l_vlan=200,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:06
actions=output:3
  cookie=0x7f5600000000000a, duration=126.846s, table=0, n_packets=2, n_bytes=148, send flo
w_rem priority=10,in_port=1,d_l_vlan=100,d_l_src=00:00:00:00:00:03,d_l_dst=00:00:00:00:00:05
actions=output:2
  cookie=0x7f5600000000000c, duration=114.843s, table=0, n_packets=2, n_bytes=148, send flo
w_rem priority=10,in_port=1,d_l_vlan=200,d_l_src=00:00:00:00:00:04,d_l_dst=00:00:00:00:00:06
actions=output:3
  cookie=0x7f56000000000007, duration=132.871s, table=0, n_packets=4, n_bytes=296, idle tim
eout=300, send_flow_rem priority=10,in_port=3,d_l_vlan=200,d_l_src=00:00:00:00:00:06,d_l_dst=
00:00:00:00:00:02 actions=output:1
  cookie=0x2b0000000000001a, duration=702.493s, table=0, n_packets=13, n_bytes=710, priorit
y=2,in_port=3 actions=output:2,output:1,CONTROLLER:65535
  cookie=0x2b00000000000019, duration=702.495s, table=0, n_packets=54, n_bytes=2988, priori
ty=2,in_port=1 actions=output:2,output:3
  cookie=0x2b00000000000018, duration=702.496s, table=0, n_packets=13, n_bytes=710, priorit
y=2,in_port=2 actions=output:1,output:3,CONTROLLER:65535
  cookie=0x2b00000000000006, duration=708.519s, table=0, n_packets=946, n_bytes=80410, prio
rity=100,d_l_type=0x88cc actions=CONTROLLER:65535
  cookie=0x2b00000000000006, duration=708.489s, table=0, n_packets=0, n_bytes=0, priority=0
actions=drop
mininet> █
```

9 AAA (Authentication-Authorization, Accounting)

The AAA project provides authentication, authorization and accounting. This service is based on the **Apache Shiro** Java Security Framework. AAA plugin utilizes the **Shiro Realms** to support this service. Authentication verifies users who are granted access to the system resources. This function is achieved while providing valid credentials (user name and password). ODL controller’s default user is the administrator. However, this service allows to create other users who also will have access the system. Authorization comes exactly after the authentication and specifies what an authenticated user can do in the system, in other words set user’s permissions. Accounting is the process that keep records of the authenticated user in system.

The AAA service is easily configured by manipulating the realms. There are two methods to achieve this:

- Utilizing the idmtool configuration tool used in order to perform basic user management operations, allows to list, add, delete, change password, delete roles and add roles, and assigns to the users.
- Utilizing the odl-aaa-shiro feature from REST API
- Utilizing the odl-aaa-cli (command line interface) of karaf console

If the second option is selected in order to make any configuration, the odl-aa-shiro feature must be installed before *restconf* API. The main configuration file for AAA is located at “*etc/shiro.ini*” relative to the ODL Karaf home directory.

ODL provides the “**admin**” user, who is permitted to do any operation in the ODL controller. Chapter [x] will present how to create a user with specific permissions.

The next figure verifies the existing “**admin**” user in ODL using the **curl** command.

```
sdn@sdn-openshift:~$ curl -u admin:admin http://localhost:8181/auth/v1/users
{"users":[{"userid":"admin@sdn","name":"admin","description":"admin user","enabled":1,"email":"","password":"*****","salt":"*****","domainid":"sdn"}]}sdn@sdn-openshift:~$
```

add this to the references

[<https://docs.opendaylight.org/en/stable-oxygen/user-guide/authentication-and-authorization-services.html>]

In this chapter user configuration will be presented. In order to proceed CRUD operation on ODL users some additional tools will be used:

- **Firefox** web browser
- **Curl** command

The above tools are not mandatory, there are also alternatives for example, Chrome web browser could be used with Postman REST API client or a curl command instead. Also, the DLUX module will be installed for testing the applied configuration on users.

openshift-user@root>feature:install features-aaa

```
openshift-user@root>feature:install features-aaa

openshift-user@root>feature:list | grep aaa
odl-aaa-netconf-plugin-no-cluster | 1.4.4 | Uninstalled | odl-aaa-
netconf-plugin-no-cluster | 0.7.4 | Started | odl-aaa-
odl-aaa-cert | 0.7.4 | x | Started | features-
features-aaa | 0.7.4 | x | Started | features-
-aaa | 0.7.4 | x | Started | features-
odl-aaa-jersey-1 | 0.7.4 | Started | odl-aaa-
jersey-1 | 0.7.4 | Started | odl-aaa-
odl-aaa-netconf-plugin | 1.4.4 | Uninstalled | odl-aaa-
netconf-plugin | 0.7.4 | Started | odl-aaa-
odl-aaa-encryption-service | 0.7.4 | Started | odl-aaa-
0.7.4 | 0.7.4 | Started | odl-aaa-
odl-aaa-api | 0.7.4 | Started | odl-aaa-
api | 0.7.4 | Started | odl-aaa-
odl-aaa-cli | 0.7.4 | Started | odl-aaa-
cli | 0.7.4 | Started | odl-aaa-
odl-aaa-shiro | 0.7.4 | Started | odl-aaa-
0.7.4 | 0.7.4 | Started | odl-aaa-
```

Before starting, the existence of the only user (admin) is verified by the next figure.

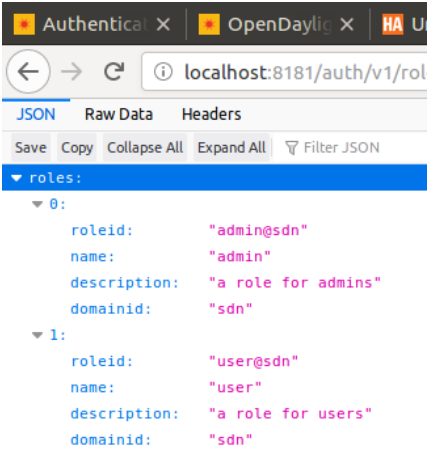
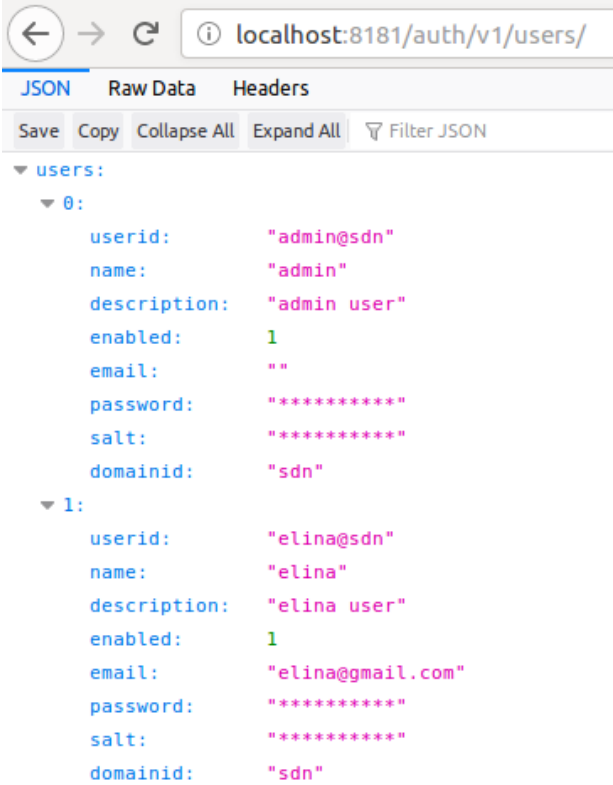
```
sdn@sdn-opendaylight:~$ curl -u admin:admin http://localhost:8181/auth/v1/users
{"users":[{"userid":"admin@sdn","name":"admin","description":"admin user","enabled":1,"email":"","password":"*****","salt":"*****","domainid":"sdn"}]}sdn@sdn-opendaylight:~$
```

Now the creation of a new user “elina” will take place using the curl command.

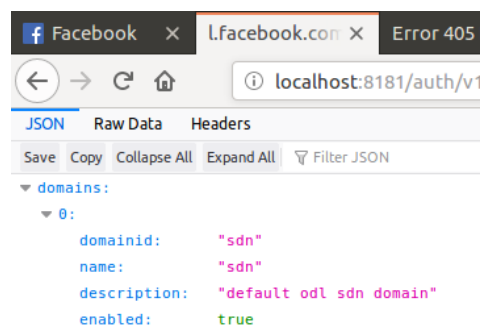
```
elina.json
{
  "name": "elina",
  "description": "elina user",
  "email": "elina@gmail.com",
  "password": "elina",
  "salt": "elina",
  "domainid": "sdn"
}
```

```
sdn@sdn-opendaylight:~$ curl -u admin:admin -X POST -H "Content-Type: application/json" --data-binary @./elina.json http://localhost:8181/auth/v1/users
{"userid":"elina@sdn","name":"elina","description":"elina user","enabled":1,"email":"elina@gmail.com","password":"*****","salt":"*****","domainid":"sdn"}sdn@sdn-opendaylight:~$
```

The endpoint URL that configures the users is <http://localhost:8181/auth/v1/users>

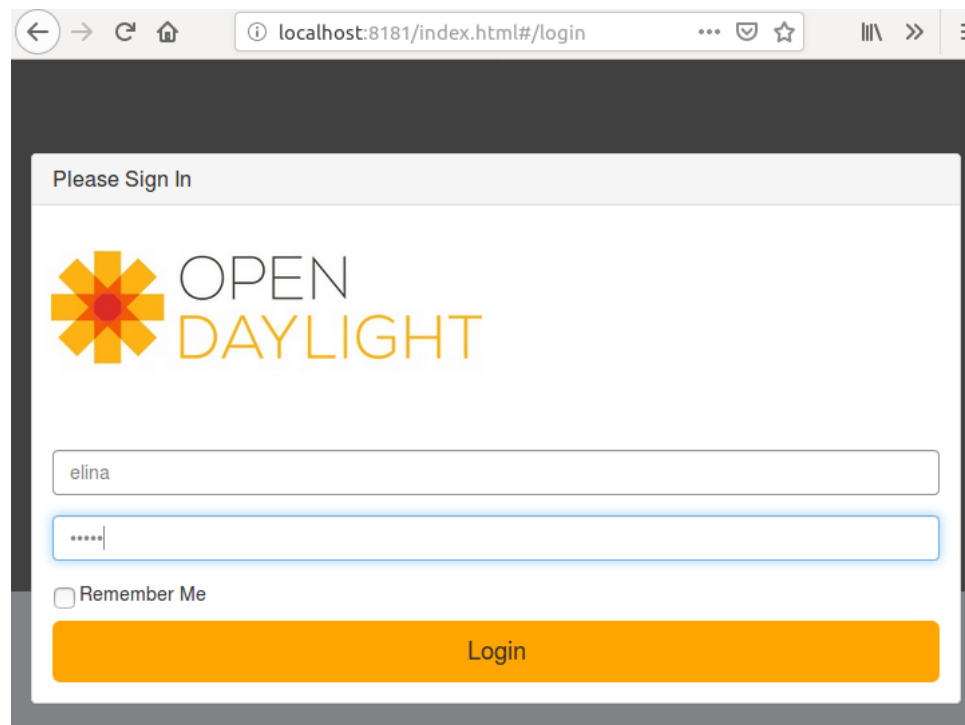
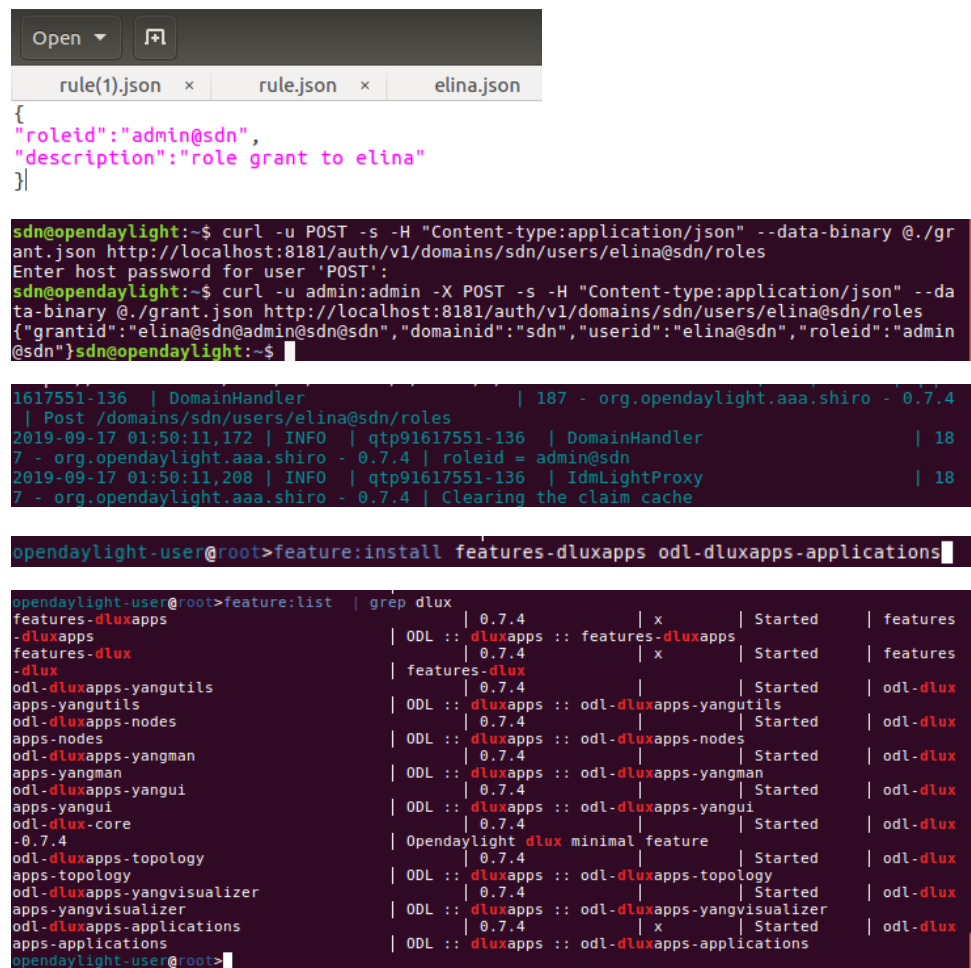


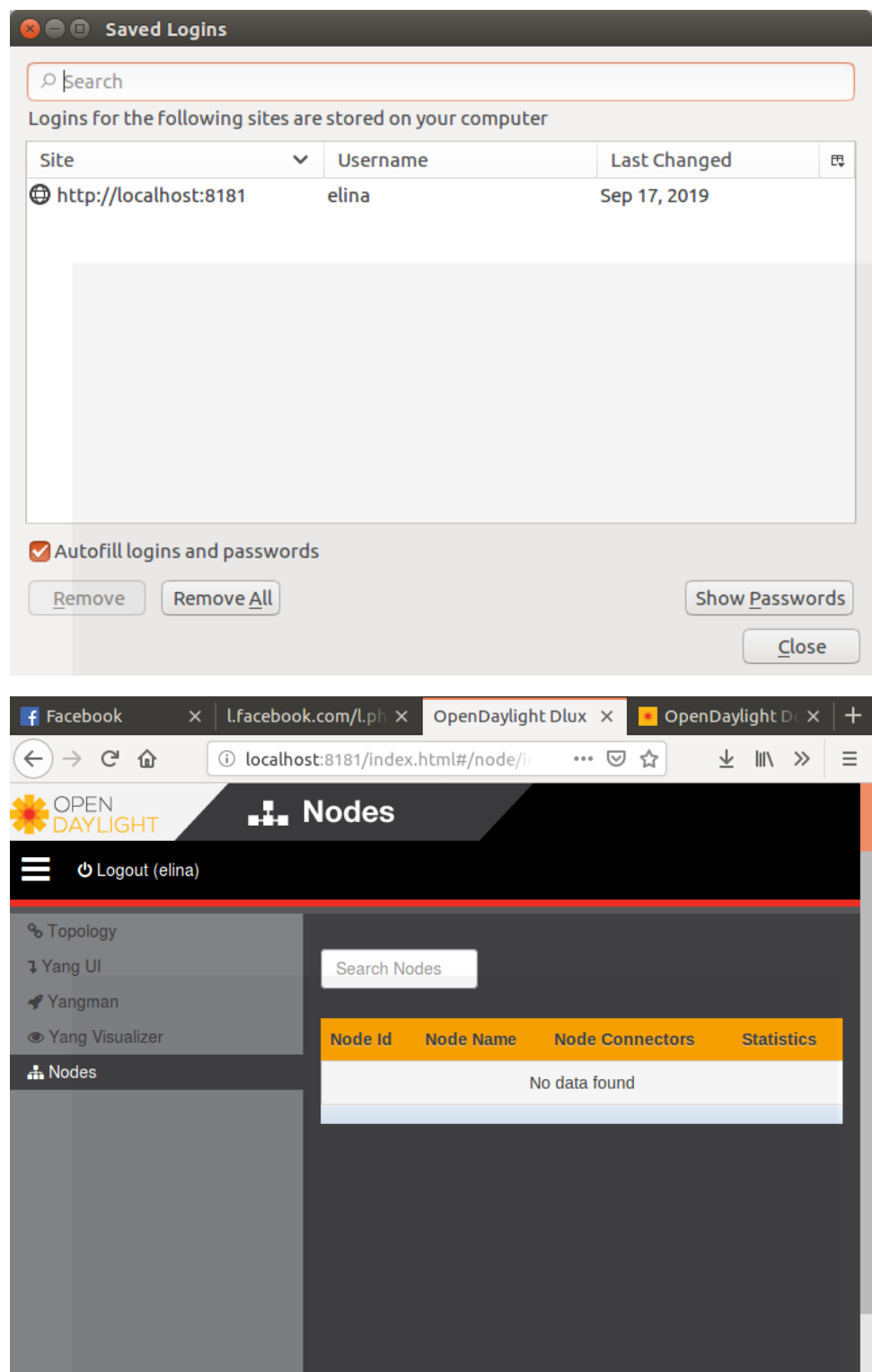
There is only one domain the default “sdn”



In order to enable the user “elina” to have access grand a role will be needed.

opendaylight-user@root>feature:install features-dluxapps odl-dluxapps-applications





References

- [1] <https://www.networkworld.com/article/3209131/what-sdn-is-and-where-its-going.html>
- [2] https://www.researchgate.net/figure/OpenFlow-switch-architecture-An-OpenFlow-Switch-consists-of-one-or-more-flow-tables-and-a_fig4_320346909
- [3] https://www.slideshare.net/bdnog/introduction-to-software-defined-networking-sdn?from_action=save
- [4] P. Heise, F. Geyer, and R. Obermaisser. Deterministic OpenFlow: Performance evaluation of SDN hardware for avionic networks. In Network and Service Management (CNSM), 2015 11th International Conference on, pages 372–377. IEEE, 2015.
- [5] Open Networking Foundation, OpenFlow Switch Specification Version 1.3.1 (Wire Protocol 0x04) September 6, 2012 ONF TS-007
- [6] <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/7995427/How+to+Work+with+Fast-Failover+OpenFlow+Groups#HowtoWorkwithFast-FailoverOpenFlowGroups-OpenFlowGroups>
- [7] <http://confignetworks.com/inside-openflow/>
- [8] <https://docs.opendaylight.org/en/stable-neon/user-guide/opendaylight-controller-overview.htm>
- [9] <https://www.javaworld.com/article/2077837/java-se-hello-osgi-part-1-bundles-for-beginners.html>
- [10] <https://www.baeldung.com/osgi>
- [11] <http://sdntutorials.com/opendaylight-and-osgi/>
- [12] <https://www.howtoforge.com/tutorial/software-defined-networking-sdn-architecture-and-role-of-openflow/>
- [13] <https://docs.opendaylight.org/en/stable-nitrogen/getting-started-guide/common-features/dlux.html>
- [14] <https://docs.opendaylight.org/en/stable-fluorine/user-guide/l2switch-user-guide.html>
- [15] <https://docs.opendaylight.org/en/stable-neon/user-guide/openflow-plugin-project-user-guide.html>
- [16] <https://docs.opendaylight.org/en/latest/downloads.html>
- [17] <https://docs.opendaylight.org/en/latest/downloads.html>
- [18] <https://docs.opendaylight.org/en/stable-oxygen/getting-started-guide/clustering.html>
- [19] <http://mininet.org/overview/>
- [20] <https://github.com/CPqD/ofsoftswitch13/wiki/Dpctl-Documentation> , <http://ranosgrant.cocolog-nifty.com/openflow/dpctl.8.html>
- [21] <https://www.openvswitch.org/>
- [22] [https://docs.opendaylight.org/en/stable-fluorine/user-guide/virtual-tenant-network-\(vtn\).html](https://docs.opendaylight.org/en/stable-fluorine/user-guide/virtual-tenant-network-(vtn).html)
- [23] <https://docs.opendaylight.org/en/stable-oxygen/user-guide/authentication-and-authorization-services.html>