**NETCONF Configuration Protocol with ONOS controller**

ESME Sudria &&

Beijing Institute of Technology

School of Computer Science and Technology

Computer Science

Bachelor’s Thesis

Student: LiLu Lin

Supervisor: Prof. Abdulhalim Dandoush

Prof. Chunxiao Gao

**Abstract**

With the rapid development of the Internet and its related area, Internet service providers and device vendors are facing the gradual increased demand on the provision and deployment and of Internet services. With the more traditional device specific CLI or the SNMP approach, the high cost of manual device configuration and service deployment is hindering the development of Internet service providers. Thus the ISPs and device vendors are working together to seek the future solution of network configuration problem.

With the of IETF RFC 4741 standard, NETCONF came into being. Together with YANG, NETCONF can create an automatic, high performance, cost efficiency approach to solve the network configuration problem. At the beginning of this paper, we mainly focus on the analysis of NETCONF protocol with ONOS SDN controller and will also cover a part of OpenFlow protocol as a comparison. Later we will perform a set of experiments on ONOS about NETCONF. Then we will discuss the Testing of NETCONF with TestON, a system test tool provided by ONOS. Finally we will find out how the NETCONF work in the scenario of optical network by using LINC-switch.

Key words: NETCONF, OpenFlow, Mininet, Of-config, LINC-switch, TestON, YANG, Optical Network, ONOS, SDN

# Table of Contents

[Introduction 5](#_Toc514893720)

[1 Chapter 1: SDN architectures and protocols 6](#_Toc514893721)

[1.1 SDN and NFV 6](#_Toc514893722)

[1.1.1 Network function virtualization 6](#_Toc514893723)

[1.1.2 Software Defined Networks 7](#_Toc514893724)

[1.1.3 Comparison of SDN and NFV 9](#_Toc514893725)

[1.2 OpenFlow 10](#_Toc514893726)

[1.2.1 Introduction 10](#_Toc514893727)

[1.2.2 When to use 11](#_Toc514893728)

[1.3 NETCONF 12](#_Toc514893729)

[1.3.1 Introduction 12](#_Toc514893730)

[1.3.2 When to use 12](#_Toc514893731)

[1.4 YANG 13](#_Toc514893732)

[1.4.1 Introduction 13](#_Toc514893733)

[1.4.2 When to use 13](#_Toc514893734)

[1.5 ONOS 14](#_Toc514893735)

[1.5.1 Introduction 14](#_Toc514893736)

[1.5.2 Architecture 14](#_Toc514893737)

[2 Chapter2: OpenFlow 16](#_Toc514893738)

[2.1 OpenFlow architecture 16](#_Toc514893739)

[2.2 OpenFlow messages format 17](#_Toc514893740)

[2.3 OpenFlow packet analysis 30](#_Toc514893741)

[3 Chapter3: NETCONF 45](#_Toc514893742)

[3.1 NETCONF unfolded 45](#_Toc514893743)

[3.1.1 Important Terminology 45](#_Toc514893744)

[3.1.2 Architecture 46](#_Toc514893745)

[3.1.3 NETCONF operations 48](#_Toc514893746)

[3.2 NETCONF setup on ONOS 53](#_Toc514893747)

[3.2.1 Prerequisites 53](#_Toc514893748)

[3.2.2 Installation 57](#_Toc514893749)

[3.2.3 Result 58](#_Toc514893750)

[4 Chapter4: Comparison of OpenFlow and NETCONF 62](#_Toc514893751)

[5 Chapter5: System test with TestON 66](#_Toc514893752)

[5.1 Introducing TestON 66](#_Toc514893753)

[5.2 Exemplary TestON usage 67](#_Toc514893754)

[6 Chapter6: Optical network use-case with LINC 72](#_Toc514893755)

[6.1 Optical network and LINC 72](#_Toc514893756)

[6.2 Use-case showcase 73](#_Toc514893757)

[6.2.1 Installation of LINC switch 73](#_Toc514893758)

[6.2.2 Connect to LINC 75](#_Toc514893759)

[7 References 76](#_Toc514893760)

[8 Appendix 78](#_Toc514893761)

[Appendix A: Environment setup on ONOS 78](#_Toc514893762)

[Appendix B: OpenFlow Packets 80](#_Toc514893763)

[Appendix C: NETCONF messages and config files 93](#_Toc514893764)

[Appendix D: Files used in TestON 115](#_Toc514893765)

[Appendix E Files used in LINC-switch 138](#_Toc514893766)

# Introduction

Internet service providers are having a hard time expanding recently when the rapid increase on network devices and services took place. One of the biggest problems that the Internet Service providers were facing is the increased cost of service provisioning and deployment caused by the high cost of manual device configuration and updating. Traditional approach like device specific CLI and SNMP cannot meet the demand of the development of Internet service because of their inevitable nature of requiring human attention. As a matter of fact that the amount of devices and the types of its vendors are growing exponentially, approaches that require human involvement will very likely to fall short on both cost-efficiency and performance. Different vendors and complex network topology will require an enormous amount of time and effort to be put into the configuration of network devices, and the delay between the creation of a service and the deployment of a service will affect the income of Internet service providers greatly.

Also in big data centers, the big amount of network devices and complex network topology require an efficient way to manage and maintain. With traditional CLI or SNMP, it will be too challenging for network operators to handle such complicated network.

To find a new approach to solve the problem that troubling the industry, ISPs and network device vendors worked together to create a next generation approach to this network device configuration problem: NETCONF and YANG. NETCONF was created to manage and handle the configuration of network devices while YANG was focused on modeling different vendors of devices, providing NETCONF with information which will be used to configure the device.

The main purpose of this paper is to analyze and introduce the OpenFlow and NETCONF protocol as well as showcasing the usage of them under multiple scenarios by practical experiment from scratch.

Objectives: Study and understand SDN networks and in particular ONOS controller with all the related virtualization environment and protocols such as mininet, OVS, iperf, SouthBound Interfaces SBI, etc. Mainly, This paper aims for providing a practical manual guide for OpenFlow and Netconf SBI with the presentation of a full experiments of some useful use-cases.

# Chapter 1: SDN architectures and protocols

## SDN and NFV

Before introducing the NETCONF protocol, a good comprehension on the background of NETCONF will certainly help on the understanding it.

### Network function virtualization

Virtualization is focused on provisioning and can be useful in various ways. For example, when a network administrator trying to set up a system for a large amount of machines , he can adopt virtualization and create one working environment. Then he can duplicate this environment for all machines he need to deploy, saving a tremendous amount of time and effort. Virtualization can also help save the resource. A powerful server can use virtualization to be divided into multiple less powerful virtual machines. This process will save the idle resources of the server if the software running on it cannot make full use of its power.

Similarly, Network function virtualization is the concept used in SDN, it is also a process of providing virtualized network functions by building software upon the traditional switches, with the development of the Internet, this concept is widely accepted and the process of NFV is growing at a rapid speed. NFV can help share the bandwidth between multiple hosts and are especially useful on managing network devices.

### Software Defined Networks

Software Defined Network is a new approach to the configuration and management of network centers and a large amount of devices. What SDN differs from traditional network is that the topology of network were defined by software instead of the hardware, indicating the separation of control plane and data plane. In this way, the creation, modification, deletion of a network topology will become flexible, simple and cost-efficient as the network operations will no longer require reconnecting cables and configurating physical switches.

SDN can also be useful in other ways. It provided both north bound and south bound interfaces. The architecture of the SDN are as follows:

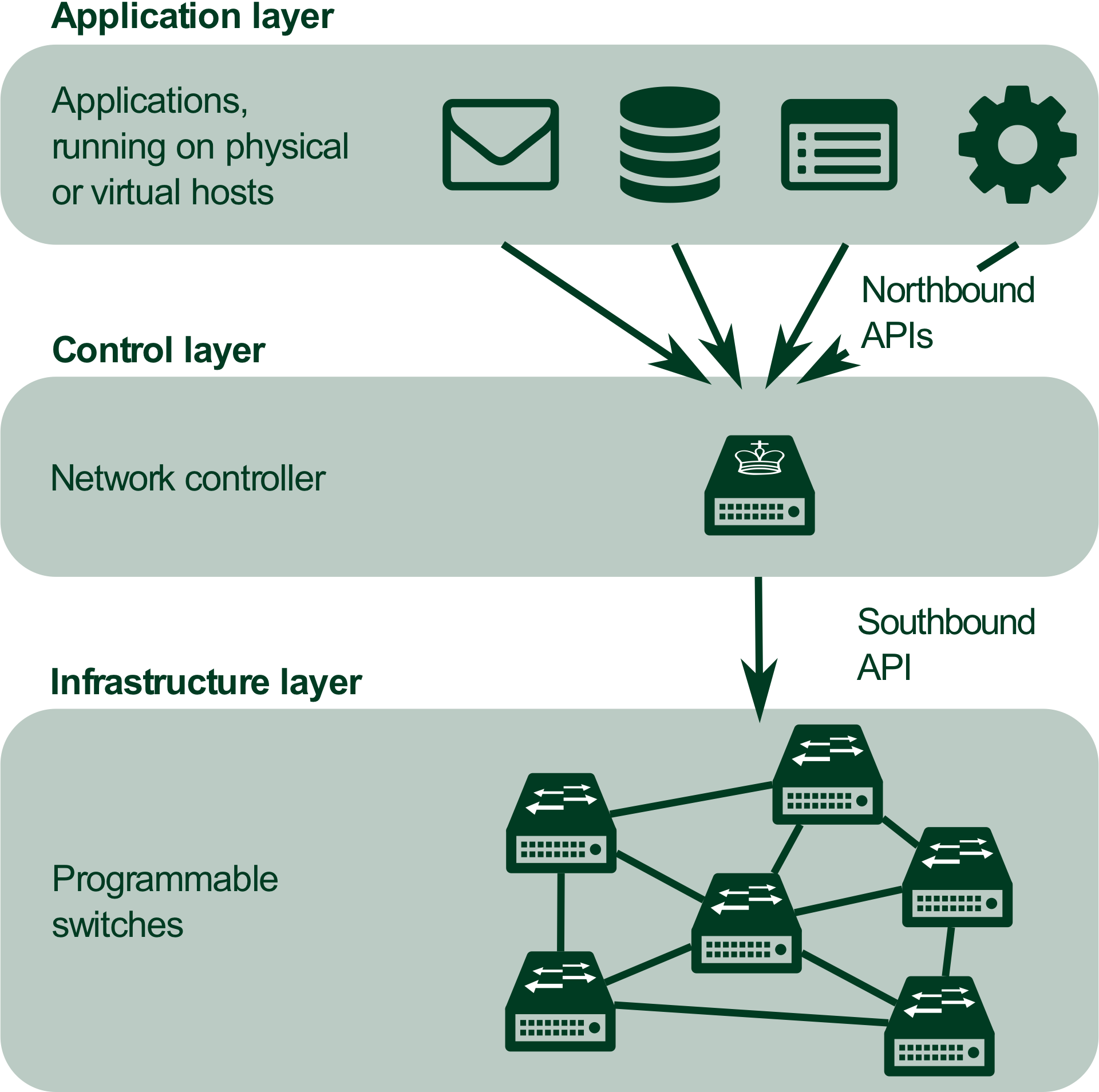


Fig 1-1. SDN architecture[18]

The north bound interfaces are usually SDN applications which provide services such as data statistics and performance evaluation, or ACL. With the flexible nature of software defined network, the applications can be deployed on many different environments regardless of the physical topology of the networks. The applications will also have an advantage over traditional ones as they can be updated more often since the SDN controller can issue OTA updates regularly whereas traditional ones were often hard-coded in a specific device and were rarely upgraded.

The south bound interfaces of SDN are the datapaths, which is a logical network device that capable of network message forwarding and data processing. It usually consist of one or more physical switches and routers. The SDN base its interface on these programmable network devices to create an abstraction that removes the protentional problem which might be brought forth by the different vendor of the devices. The network devices will support a certain kind of protocol such as OpenFlow or NETCONF in order to communicate with SDN controller. Working together, the SDN controller now serves as the control plane and the datapath now works as the data plane.

The SDN controller itself is the logical center of the entire SDN. However, its implementation can be distributed or decentralized. The controller is responsible for the implantation of control mechanism of the entire SDN. It will have an integral view on the entire network, which simplifies the control logic and enables many features that were not possible for traditional network. It controls the packet forwarding logic which defines the topology of the SDN.

### Comparison of SDN and NFV

To sum up, Fig 2 from sdxcentral[17] shows the difference between NFV and SDN.

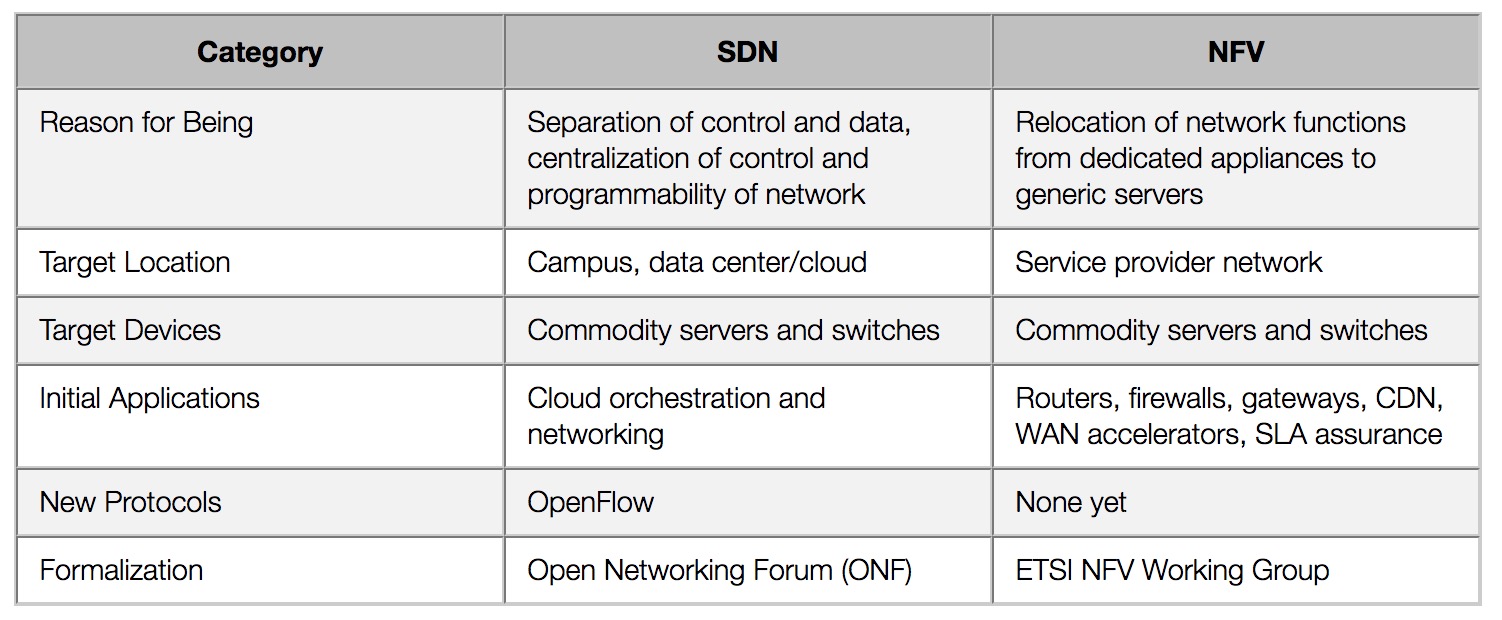


Fig 1-2. NFV compared to SDN[17]

Simply put it, NFV is more generic compared to SDN, it can be implemented in non-SDN environments such as switches, routers and gateways. It can be used to help realize the SDN in many cases. SDN is more targeted to the data center scenario whereas NFV is more targeted to service provider network. SDN is more about data traffics, which the data center cares a lot, the statistical use of SDN is undeniably valuable for network monitoring, and alternation of network topology is performed in data centers more frequently since the change of business logic in data center is often. ISPs are, on the other hand, only require device reconfigurations to provide new services or update existing ones.

## OpenFlow

### Introduction

OpenFlow is a network transmission protocol that change the way switches and routers work by making them exchange packets according to flow table and OpenFlow SDN controller instead of the hardware controller. It serves as a southbound interface in SDN network and centralize the controller on one machine and make network devices more focusing on forwarding. OpenFlow separates the controlling and the forwarding in order to create sophisticated network topology, it also modifies packet distributing rules to control the network. SDN controller can only modify the flow table.

OpenFlow transformed the switch/router controlled packet forwarding mechanics into a cooperation between OpenFlow switch and SDN controller. In this way it separates the control and forwarding so that the controller can set the OpenFlow flow table in the OpenFlow switch in order to establish control over the expected data forwarding.

It is more focused on traffic engineering meaning that it is more about controlling the traffic. The flow table enables it to control packet across different vendors of network devices. It cannot change the way device work fundamentally.

Here we use OpenFlow as a comparison protocol to NETCONF, to see the difference between them so we can have a grip on when and how do we use NETCONF.

### When to use

When we have multiple brands of network devices and in need of adding a new function to the SDN, if some of the devices do not have the CLI command to provide the function we need. We can change the flow table to easily achieve that function using OpenFlow.

## NETCONF

### Introduction

NETCONF also acts as a SDN southbound interface, it is a new way to manage the network devices. NETCONF uses the API or CLI provided by the traditional network devices to obtain and send the configuration data, state data and notification data. All NETCONF messages are in the form of XML, as standardized by IETF RFC 6241.

NETCONF keeps the original function of the network devices and does not require modifications like OpenFlow does. SDN controller can modify every section of the device, meaning that the NETCONF can make fundamental changes of the device, which enables it to serve as a network device configuration protocol.

### When to use

Under the circumstances when we must reconfigure a network device, we need to use NETCONF over OpenFlow since OpenFlow is more focused on traffic engineering and mostly work on flow table, which contains packet distributing rules. So in order to change the device fundamentally, we need NETCONF.

Also when we need some configuration to be permanent, we should use NETCONF because the flow table in network devices are temporary, it will be lost after reboot.

## YANG

### Introduction

YANG is a data modeling language defined by IETF RFC6020 and mostly used by NETCONF. As there need to be a standard of data travels between the NETCONF SDN controller and network devices, Yang came into being. It can be used to model configuration data and state data as well as be converted to XML JSON format for later use.

NETCONF can obtain information of the device through the YANG model profiles, each device has a YANG model. In the cases where the device does not provide a model, NETCONF can use a generic one.

Here we use YANG to work together with NETCONF to better run our experiments as YANG have an indispensable place in the use of NETCONF.

### When to use

In order to configure the network devices automatically, NETCONF need to know the basic structure of a type of device. If we create a Yang model for this kind of device. NETCONF will know what kind of info it need to get from the device API. Therefore, it will work correctly after it gets the information provided by device API.

## ONOS

### Introduction

Open Network Operating System, known as ONOS, is a leading SDN controller created by ONF(Open Network Foundation). It is open-sourced and aims for SDN/NFV solutions.

ONOS was designed to meet the need of ISPs and data centers, providing simple programmable interfaces. ONOS supports various way of controlling its southbound network. With ONOS control plane, the end-users can use network applications without the need to face the problem of configuring the switches and routers themselves.

Here we use ONOS as SDN controller in order to learn and study about NETCONF protocol as well as other protocols, then we can also perform experiments on it to enhance what we have learned.

### Architecture

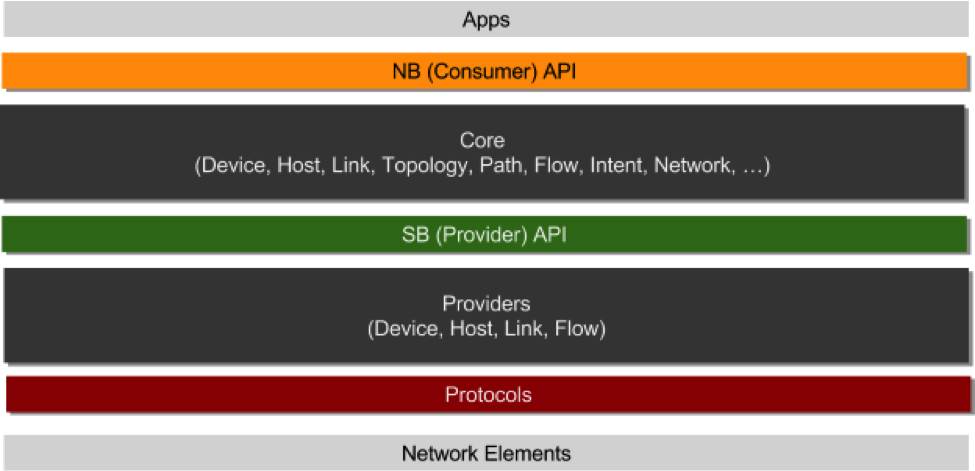


Fig 1-3. ONOS Architecture[16]

ONOS is written in Java based on Apache Karaf OSGI container, it is designed to be running as several nodes and can be managed by cell cluster.

ONOS is not bounded with any type of specific network configuration protocol such as OpenFlow or NETCONF. It adopted high-level abstraction model which can in turn be adopted to a specific type of standard type of protocol, providing ONOS with high flexibility.

ONOS application can be loaded and unloaded individually using its north bound interfaces. The modular design helped to bring the flexibility and compatibility to the next level. A number of applications are provided by ONOS by default, ranging from device management to traffic processing.

ONOS have four main northbound interfaces, the GUI, the REST API, the gRPC and CLI. GUI is the most convenient way for normal end users to get involved with ONOS, as it provide a well-built WEB GUI. Users can explore from the browser and check the information of the devices by clicking on tabs. Most importantly, the user can view the topology from GUI with ease. The REST API are useful for communicating with ONOS using program, it can also be used by putting URL in browser. gRPC is another way to get information from ONOS using programs. CLI is an easy way for experienced user to perform certain operations on ONOS where they could not find in GUI.

Overall, ONOS is a well-built SDN controller that can act as a control plane for the SDN network.

# Chapter2: OpenFlow

OpenFlow originates from the Clean Slate Project in Stanford University. OpenFlow has a simple and elegant design by being an API-like network device configure protocol. It maintain one or more flow tables, the most common way OpenFlow are using to handle the packets, whose generation, maintenance, distribution process was controlled by SDN controller.

## OpenFlow architecture

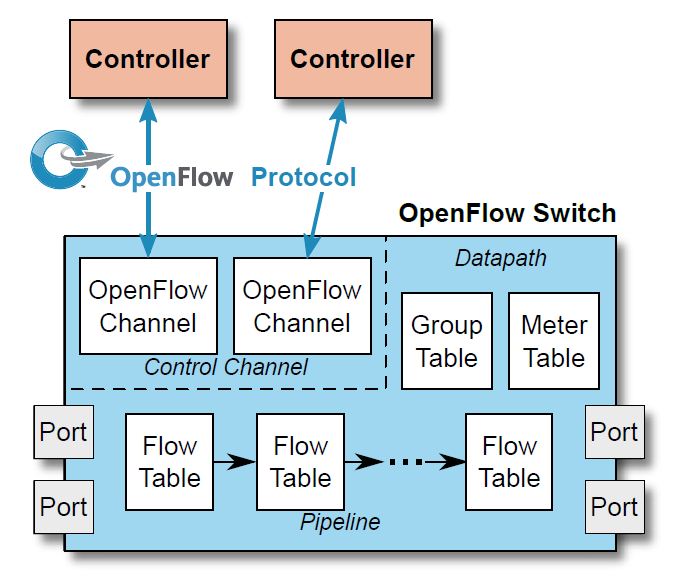


Fig 2-1. OpenFlow Architecture[9]

OpenFlow protocol are based on one or more flow tables and one group table. The tables contains the rule of packet forwarding and packet lookups.

OpenFlow do not have control plane, it created OpenFlow channels to communicate with external controllers to work. The external controller can add/update/delete flow entries in flow tables, alternating the packet forwarding/lookup rules

Each flow entry have match fields and some instructions to match packets, the packets will go through pipeline and eventually forward to a port, which can be physical or logical.

The group table, on the other hand, can have addition processing for the packets such as flooding, multipath, fast reroute, and link aggregation. It contains group entries and have more complex forwarding semantics.

The meter table consists of meter entries, which can be used to implement Qos operations such as limiting bandwidth or rate. The SDN controller can use statistics data for traffic optimization.

## OpenFlow messages format

OpenFlow has a big set of messages, Here we take a look at the functionality and format of the most of them to have a better understanding on how OpenFlow works.

#### OFPT\_HELLO

After creating the socket, switch will exchange OpenFlow hello packet with the SDN controller.

##### Objective：to negotiate the protocol version.

##### Content：the highest protocol version supported by this end.

##### Result：select the protocol version supported by both ends.

##### On Success：Establish connection.

##### On Fail：OFPT\_ERROR (TYPE:OFPT\_HELLO\_FAILED,CODE =0), connection ended.

#### OFPT\_ERROR

Types of OpenFlow errors:

##### OFPET\_HELLO\_FAILED

##### OFPET\_BAD\_REQUEST

##### OFPET\_BAD\_ACTION

##### OFPET\_FLOW\_MOD\_FAILED

##### OFPET\_PORT\_MOD\_FAILED

##### OFPET\_QUEUE\_OP\_FAILED

#### OFPT\_ECHO

##### Including OFPT\_ECHO\_REQUEST, OFPT\_ECHO\_REPLY

##### Function：check the connection status to make sure the connection is still working.

when there are no other kind of packets travelling between the switch and controller, the controller will send OFPT\_ECHO\_REQUEST message to switch once in a while.

#### OFPT\_FEATURES

when the connection between switch and controller were established , controller will send OFPT\_FEATYRES\_REQUEST packet to the switch in order to obtain supported features from the switch.

##### sending time：after the connection is established

##### packet from controller to switch：OFPT\_FEATURES\_REQUEST

##### packet from the switch to controller：OFPT\_FEATURES\_REPLY

##### Objective：to obtain supported features from the switch

#### OFPT\_FEATURES\_REQUEST

##### TYPE=5

##### Without data

#### OFPT\_FEATURES\_REPLY

##### TYPE =6

##### [0:8] is header

##### [8:32] are features of switch in length of 24 bytes.

##### [32:] info about ports, proportional to the number of ports in length of 48 bytes.

Possible switch features:

##### NOT DEFINED

##### OFPC\_ARP\_MATCH\_IP   #Match IP address in ARP packets

##### OFPC\_QUEUE\_STATS   #Queue statistics

##### OFPC\_IP\_STREAM     #Can reassemble IP fragments

##### OFPC\_RESERVED     #Reserved, must be zero

##### OFPC\_STP           #802.1d spanning tree

##### OFPC\_PORT\_STATS   #Port statistics

##### OFPC\_TABLE\_STATS   #Table statistics

##### OFPC\_FLOW\_STATS   #Flow statistics

Possible port information:

##### port\_no

##### hw\_addr

##### port\_name

##### OFPPC\_NO\_PACKET\_IN

##### OFPPC\_NO\_FWD

##### OFPPC\_NO\_FLOOD

##### OFPPC\_NO\_RECV\_STP

##### OFPPC\_NO\_RECV

##### OFPPC\_NO\_STP

##### OFPPC\_PORT\_DOWN

##### # state else

##### OFPPS\_LINK\_DOWN

##### ​

##### # Current features

##### OFPPF\_PAUSE\_ASYM

##### OFPPF\_PAUSE

##### OFPPF\_AUTONEG

##### OFPPF\_FIBER

##### OFPPF\_COPPER

##### OFPPF\_10GB\_FD

##### OFPPF\_1GB\_FD

##### OFPPF\_1GB\_HD

##### OFPPF\_100MB\_FD

##### OFPPF\_100MB\_HD

##### OFPPF\_10MB\_FD

##### OFPPF\_10MB\_HD

##### ​

##### #32 bit each below

##### advertised # features being advised by the port

##### supported # features supported by the port

##### peer # features advertised by peer

This part of information is key to OpenFlow. Every action performed by controller will be based on information on features supported by the switch obtained from here. Every time the switch connected to the switch, it will receive a features\_request message from the controller ,when the switch send the features back to the controller, the controller will know about the switch so as to get information to assume control later.

#### OFPT\_PACKET\_IN

After the controller got the features of the switch, the switch will start processing packets.

When there is an packet received by the switch which the switch have no idea about what action to perform on, the switch will encapsulate it in packet\_in packet and forward it to the controller. eg arp packets and icmp packets.

reason for packet\_in packet generation:

##### **OFPR\_NO\_MATCH**

##### **OFPR\_ACTION**

packets that are not matched will generate packet\_in, an action can also generate packet\_in which means we can use action to send the data packet we need to controller.

After packet\_in was generated, either one of these will be performed：

##### **Packet\_out**

##### **Flow\_mod**

If the packet is meant for broadcasting, controller will encapsulate it as packet\_out packet and send it back to switch and ordering it to perform flood action, meaning sending this packet to all ports except the receiving one.

#### OFPT\_PACKET\_OUT

##### function：for controller to send data to the switch

##### Eg：when using arp in ofsw, it shouldn't be directly broadcasted. Instead, it should be encapsulated in the packet\_out packet and then broadcast the packet\_out packet.

#### OFPT\_FLOW\_MOD

OFPT\_FLOW\_MOD is the most important data in OpenFlow.

OFPT\_FLOW\_MOD consists of header & match & flow\_mod & action.

Packet content:

##### Wildcards

##### OFPFW\_NW\_TOS=1

##### OFPFW\_DL\_VLAN\_PCP=1

##### OFPFW\_NW\_DST\_MASK=0

##### OFPFW\_NW\_SRC\_MASK=0

##### OFPFW\_TP\_DST=1

##### OFPFW\_TP\_SRC=1

##### OFPFW\_NW\_PROTO=1

##### OFPFW\_DL\_TYPE=1

##### OFPFW\_DL\_VLAN=1

##### OFPFW\_IN\_PORT=1

##### OFPFW\_DL\_DST=1

##### OFPFW\_DL\_SRC=1

##### Match

##### in\_port=msg.payload.payload.payload.in\_port

##### dl\_src=pkt\_parsed.src

##### dl\_dst=pkt\_parsed.dst

##### dl\_type=pkt\_parsed.type

##### dl\_vlan=pkt\_parsed.payload.vlan

##### nw\_tos=pkt\_parsed.payload.tos

##### nw\_proto=pkt\_parsed.payload.proto

##### nw\_src=pkt\_parsed.payload.src

##### nw\_dst=pkt\_parsed.payload.dst

##### tp\_src = 0

##### tp\_dst = 0

##### Flow\_mod

##### cookie=0

##### command=0

##### idle\_timeout=10

##### hard\_timeout=30

##### out\_port=msg.payload.payload.payload.payload.port

##### buffer\_id=buffer\_id

##### flags=1

#### **OFP\_HEADER**

header is the head of every packet which have 3 parameters

##### type

##### length

##### xid: packet id

Note that basic length is 72, action have a length of 8 so the length have to be able to be divided evenly by 8.

#### WILDCARDS

First 32 bit of match on OF1.3 OFPFW\_NW\_DST\_MASK=63 means matched IP address, 0 means not. With OpenFlow 1.0 is the opposite.

#### MATCH

Storing the control information here.

if there was a packet in before the FLOW\_MOD, match will be filled with corresponding data, so that the FLOW\_MOD will be working correctly.

Something important to be noted：

##### not every packet will have a vlan\_tag.

##### not every packet will have 4 layer ports.

#### FLOW\_MOD

Flow mod packet structure:

##### BitField("cookie", 0, 64) #Opaque controller-issued identifier

##### ShortEnumField("command", 0, ofp\_flow\_mod\_command)

##### ShortField("idle\_timeout", 60)

##### ShortField("hard\_timeout", 0)

##### ShortField("priority", 0)

##### IntField("buffer\_id", 0)

##### ShortField("out\_port", 0) #flags are important, the 1<<0 bit is OFPFF\_SEND\_FLOW\_REM, send OFPT\_FLOW\_REMOVED

##### #1<<1 bit is OFPFF\_CHECK\_OVERLAP, checking if the entries' field overlaps(among same priority)

##### #1<<2 bit is OFPFF\_EMERG, used only switch disconnected with controller)

##### ShortField("flags", 0)

command field determined the action of flow\_mod:

##### ofp\_flow\_mod\_command =   0: OFPFC\_ADD           # New flow

##### 1: OFPFC\_MODIFY         # Modify all matching flows

##### 2: OFPFC\_MODIFY\_STRICT # Modify entry strictly matching wildcards

##### 3: OFPFC\_DELETE         # Delete all matching flows

##### 4: OFPFC\_DELETE\_STRICT # Strictly match wildcards and priority

#### Time parameter: idle\_timeout & idle\_timeout

##### idle\_timeout: x, if a flow packet was not matched in x seconds, it will be deleted.

##### hard\_timeout: y, this flow packet will be deleted after y seconds, even it's still active,even matched.

#### priority

priority field indicating the priority of flow packets.

the larger the number, the higher the priority, and the smaller number of table in which it will be stored.

#### buffer\_id

usually set by the switch automatically. If the packet was sent manually buffer\_id should be -1(0xffff), informing the switch that this packet is not in the buffer queue.

#### out\_port

Assigned output port, with some speical ones like flood and local

##### 0xff00: OFPP\_MAX

##### 0xfff8: OFPP\_IN\_PORT

##### 0xfff9: OFPP\_TABLE

##### 0xfffa: OFPP\_NORMAL

##### 0xfffb: OFPP\_FLOOD

##### 0xfffc: OFPP\_ALL

##### 0xfffd: OFPP\_CONTROLLER

##### 0xfffe: OFPP\_LOCAL

##### 0xffff: OFPP\_NONE

if you don't know the port number, you should fill in flood: 0xfffb.

#### flags

They should be set as 1 because it will let the switch to send a flow\_removed message to controller about deleting a flow packet.

#### ACTION

Without an action, the switch will drop the packet, which is the default action when handling a FLOW\_MOD packet. So every flow\_mod packet should have an action. There are two types of action：

##### **primary action: Forward and Drop**

##### **optional action:FLOOD,NALMAL**

For example, output is a primary action. every action will have an action\_header(), and then a body:

##### ofp\_action\_header(type=0)/ofp\_action\_output(type =0, port =oxfffb,len =8)

all actions:

##### 0: OFPAT\_OUTPUT

##### 1: OFPAT\_SET\_VLAN\_VID

##### 2: OFPAT\_SET\_VLAN\_PCP

##### 3: OFPAT\_STRIP\_VLAN

##### 4: OFPAT\_SET\_DL\_SRC

##### 5: OFPAT\_SET\_DL\_DST

##### 6: OFPAT\_SET\_NW\_SRC

##### 7: OFPAT\_SET\_NW\_DST

##### 8: OFPAT\_SET\_NW\_TOS

##### 9: OFPAT\_SET\_TP\_SRC

##### 10: OFPAT\_SET\_TP\_DST

##### 11: OFPAT\_ENQUEU

action will not be limited in flow\_mod, it will also appear in stats\_reply and other OpenFlow packets.

#### OFPT\_BARRIER\_REQUEST && REPLY

When the switch receives OFPT\_BARRIER\_REQUEST message, it will send a OFPT\_BARRIER\_REPLY message to controller. As the packets were following the FIOF order, when the controller received this message it means the flow\_mod action was performed.

Possible errors when flow\_mod action was performed:

##### If there was a non-logic errorthe swtich will report this error when processing flow\_mod packet.

##### If there was a logic error, it will be written into the flow table, such error can't be dectected by openflow.

#### OFPT\_FLOW\_REMOVED

if flow\_mod's flags is set to 1, then after the flow expired, the switch will send a OFPT\_FLOW\_REMOVED message to controller.

##### Structure：header()/wildcards()/match()/flow\_removed()

##### Function：notify the controll after the flow expires with some stacitacs data.

Flow removed fields:

##### cookie 64

##### priority 16

##### reason 8

##### pad None

##### duration\_sec 32

##### duration\_nsec 32

##### idle\_timeout 16

##### pad 0

##### pad 0

##### packet\_count 64

##### byte\_count 6

duration\_sec field is the time the flow packet existed.

#### OFPT\_STATS\_REQUEST && REPLY

OFPT\_STATS\_REQUEST && REPLY is for statistics information collection.

#### OFPT\_STATS\_REQUEST

There are many types of OFPT\_STATS\_REQUEST and the OFPT\_STATS\_RPLY

Stats Request fields:

##### ShortEnumField("type", 0, ofp\_stats\_types)

##### ShortField("flag", 0)

#### **Type**

##### 0: Request for the switch vendor and version.

##### 1: flow stats

##### 2: aggregate stats request

##### 3: Type of OFPST\_TABLE (0)

##### 4: port status request

##### 5: queue request

##### 6:vendor request, might be undefined.

#### OFPT\_STATS\_REPLY

Every request will have a corresponding reply, the most import reply(low\_stats\_reply) is as below:

##### Structure：**header(type=17)/reply\_header()/flow\_stats/wildcards/match/ flow\_stats\_data**

##### Function：send statistical information about flow

ofp\_flow\_stats(body[4:8]) will have table\_id field to show which flow table it was in. flow\_stats\_data contains the packet\_count and byte\_count field, with these two fields we can calculate the throughput.

After these data, there should still be some actions.

#### ERROR

Open Flow Error structure:

Type

##### 0: "OFPET\_HELLO\_FAILED"

##### 1: "OFPET\_BAD\_REQUEST"

##### 2: "OFPET\_BAD\_ACTION"

##### 3: "OFPET\_FLOW\_MOD\_FAILED"

##### 4: "OFPET\_PORT\_MOD\_FAILED"

##### 5: "OFPET\_QUEUE\_OP\_FAILED"

Error Codes：

##### hello\_failed\_code

##### 0: "OFPHFC\_INCOMPATIBLE"

##### 1: "OFPHFC\_EPERM"

##### bad\_request\_code

##### 0: "OFPBRC\_BAD\_VERSION"

##### 1: "OFPBRC\_BAD\_TYPE"

##### 2: "OFPBRC\_BAD\_STAT"

##### 3: "OFPBRC\_BAD\_VENDOR"

##### 4: "OFPBRC\_BAD\_SUBTYPE"

##### 5: "OFPBRC\_EPERM"

##### 6: "OFPBRC\_BAD\_LEN"

##### 7: "OFPBRC\_BUFFER\_EMPTY"

##### 8: "OFPBRC\_BUFFER\_UNKNOWN"

##### bad\_action\_code

##### 0: "OFPBAC\_BAD\_TYPE"

##### 1: "OFPBAC\_BAD\_LEN"

##### 2: "OFPBAC\_BAD\_VENDOR"

##### 3: "OFPBAC\_BAD\_VENDOR\_TYPE"

##### 4: "OFPBAC\_BAD\_OUT\_PORT"

##### 6: "OFPBAC\_BAD\_ARGUMENT"

##### 7: "OFPBAC\_EPERM"         #permissions error

##### 8: "OFPBAC\_TOOMANY"

##### 9: "OFPBAC\_BAD\_QUEUE"

##### flow\_mod\_failed\_code

##### 0: "OFPFMFC\_ALL\_TABLES\_FULL"

##### 1: "OFPFMFC\_OVERLAP"

##### 2: "OFPFMFC\_EPERM"

##### 3: "OFPFMFC\_BAD\_EMERG\_TIMEOUT"

##### 4: "OFPFMFC\_BAD\_COMMAND"

##### 5: "OFPFMFC\_UNSUPPORT"

##### port\_mod\_failed\_code

##### 0: "OFPPMFC\_BAD\_PORT"

##### 1: "OFPPFMC\_BAD\_HW\_ADDR"

##### ​

##### queue\_op\_failed\_code

##### 0: "OFPQOFC\_BAD\_PORT"

##### 1: "OFPQOFC\_BAD\_QUEUE"

## OpenFlow packet analysis

First we create our own network topology using mininet.

mn –-topo single,2 –-mac –-switch ovsk,protocols=OpenFlow13 –-controller remote,ip=192.168.56.102,port=6633

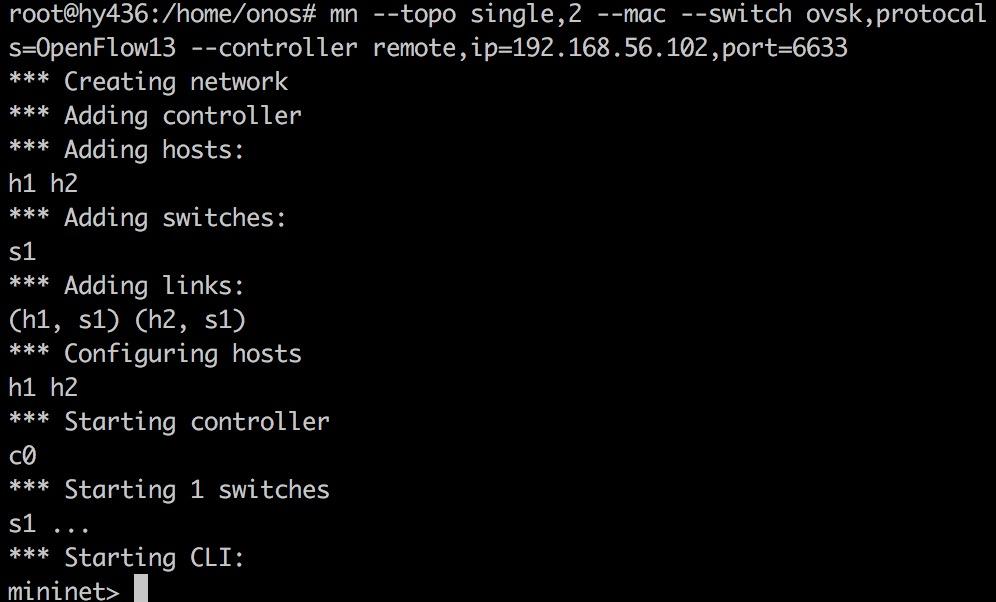


Fig 2-2. topology create result

We can see pingall results show that the topology was successfully created.

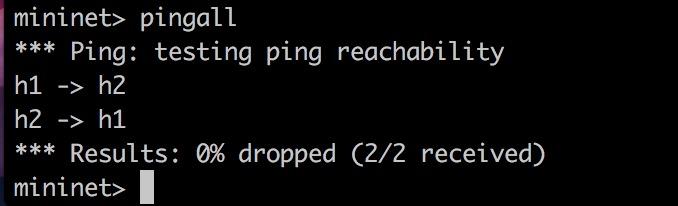


Fig 2-3. pingall result

Now take a look at theONOS WEB GUI at <http://192.168.56.102:8181/onos/ui/> we can see a topology like this:

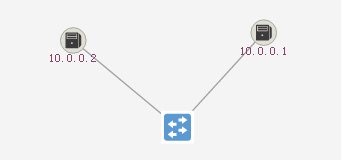


Fig 2-4. topology

Now we capture packets on the VM using command

###### tcpdump -i lo -w mycap.pcap

And then we fetch it using scp

###### scp [onos@192.168.56.102:~/mycap.pcap ~/Desktop/mycap.pcap](mailto:onos@192.168.56.102:~/mycap.pcap%20~/Desktop/mycap.pcap)

Open it with wireshark with OpenFlow v4 protocol filter , we get:

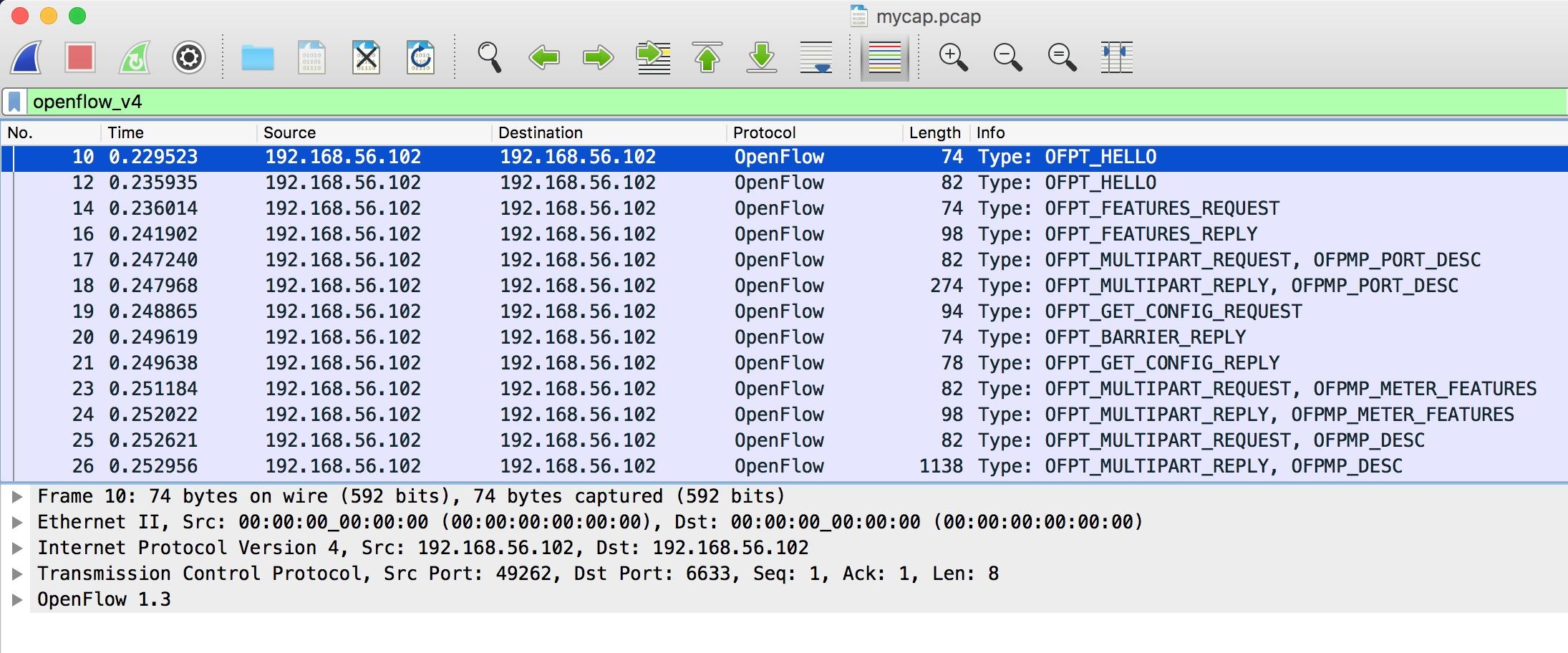


Fig 2-5. packet captured

Now that we get the Packets, let's take a look at them:

#### TCP message

,

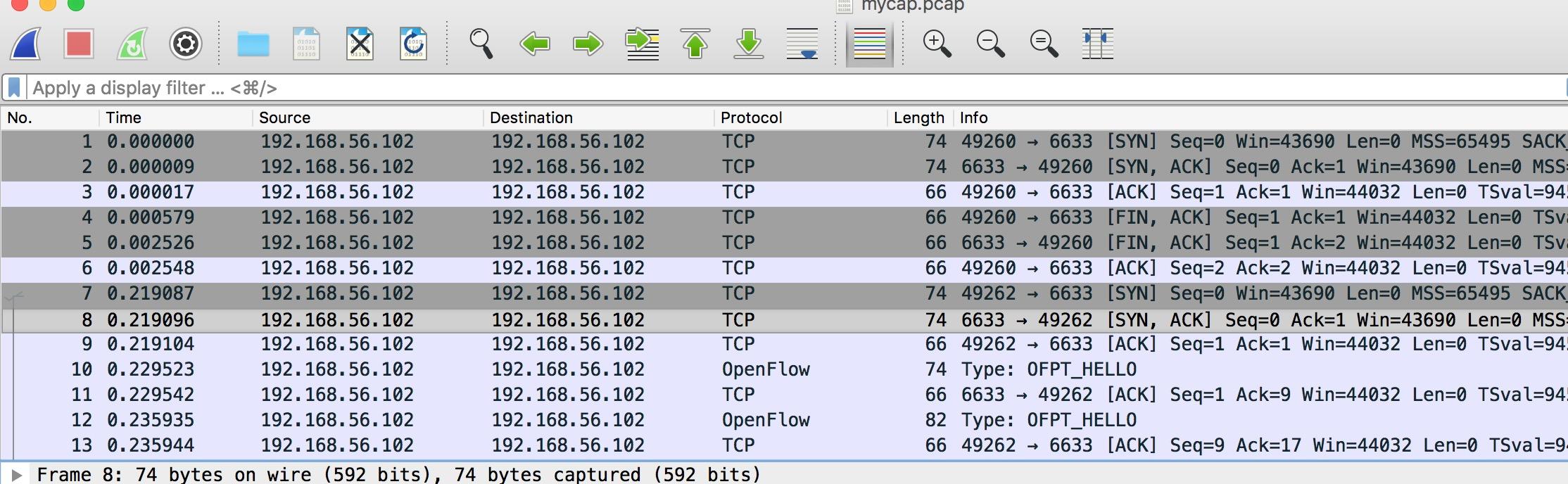


Fig 2-6. TCP messages

First we have three-way handshake tcp message here, it was initiated by the switch, only after this base level connection was established can the OFTP\_HELLO message to be sent.

#### OFPT\_HELLO message

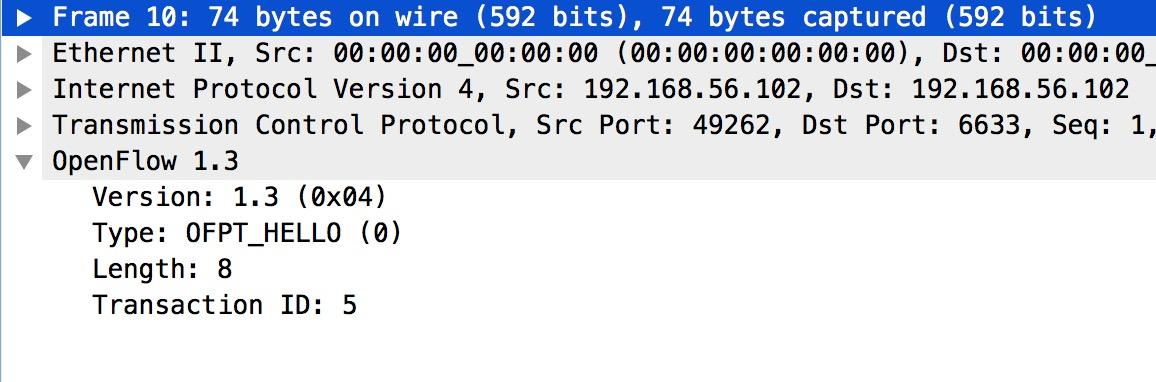


Fig 2-7. OFPT\_HELLO message

The switch send Hello message to the SDN controller to negotiate the protocol version, the version field in the message head is the highest OpenFlow version supported by the switch. Hello message only have OpenFlow message header (of\_header) without of\_message\_body.

At the end of Hello message, both ends will use the lower OpenFlow version supported by one end.

#### OFPT\_FEATURES\_\* message

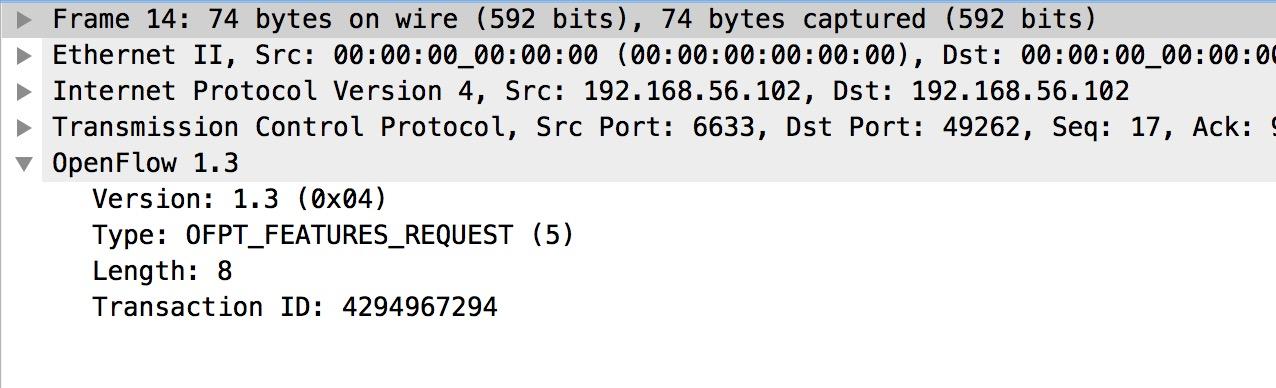


Fig 2-8. OFPT\_FEATURES\_\* message

The controller sent an OFTP\_FEATURES\_REQUEST message to obtain the features supported by the switch, it also only have of header.

The switch will send an OFPT\_FEATURES\_REPLY back to controller.

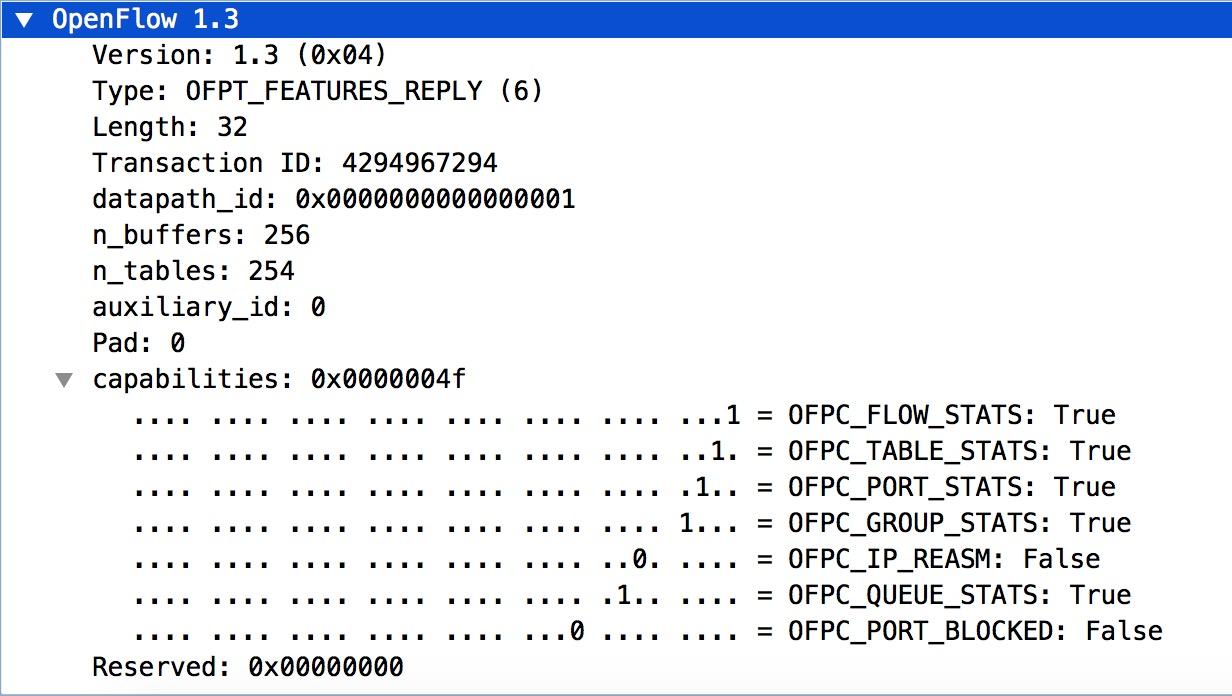


Fig 2-9. OFPT\_FEATURES\_REPLY

Take a look at it we will know that this switch can at most buffer 256 packets at once, this datapath support up to 254 tables, supports flow, table, port, group and queue statistics. Cannot reassemble IP fragments, and will not block looping ports.

#### OFPT\_BARRIER\_\* & \*\_CONFIG\_\*

After the controller get the features message from the switch, it will set the config of the switch. Now we see the packet set the max bytes of packet that datapath should send to the controller to 65535.

Also the controller will send OFPT\_BARRIER\_REQUEST message to the switch to make sure the switch finish the operations before this message and then go on to the operations after this message, so that flags and miss send length field in the config were correctly set before proceed to any other action.

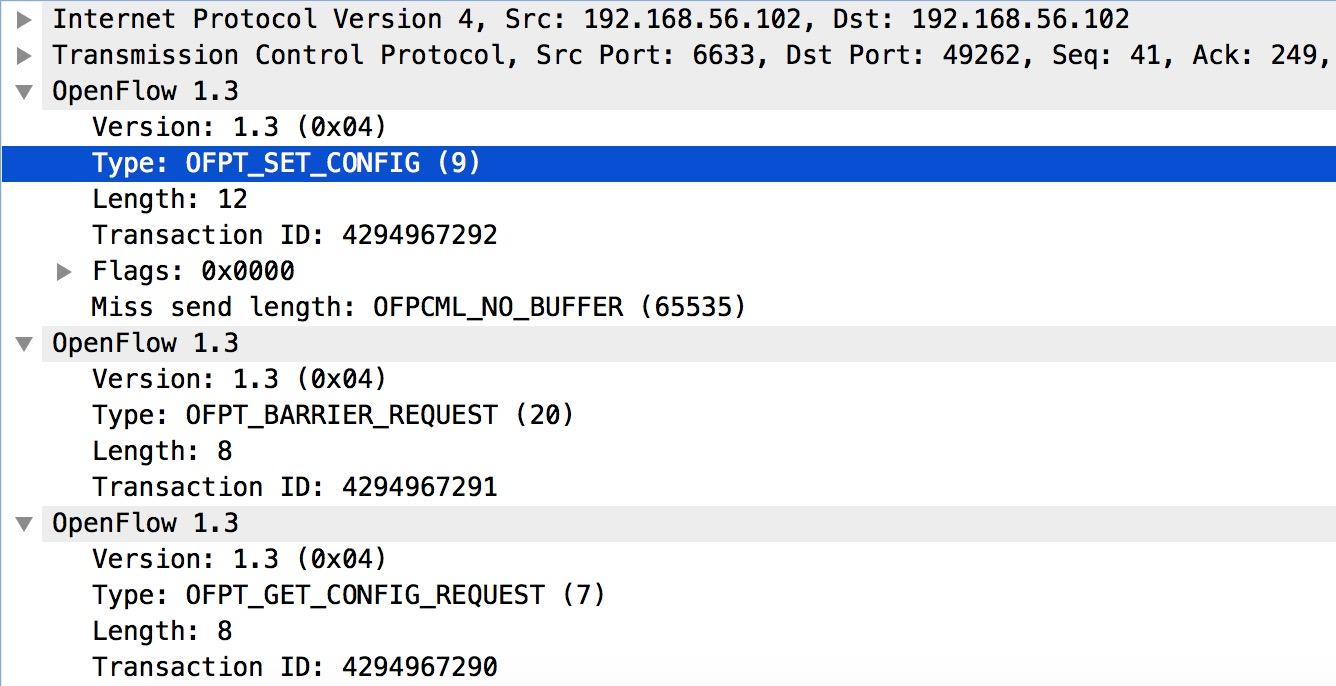


Fig 2-10. Three OpenFlow messages

The OFPT\_SET\_CONFIG will not get a reply, instead the controller send a OFPT\_GET\_CONFIG\_REQUEST to see if the configuration was correct.

After this operation was complete, the switch will send a OFPT\_BARRIER\_REPLY message.

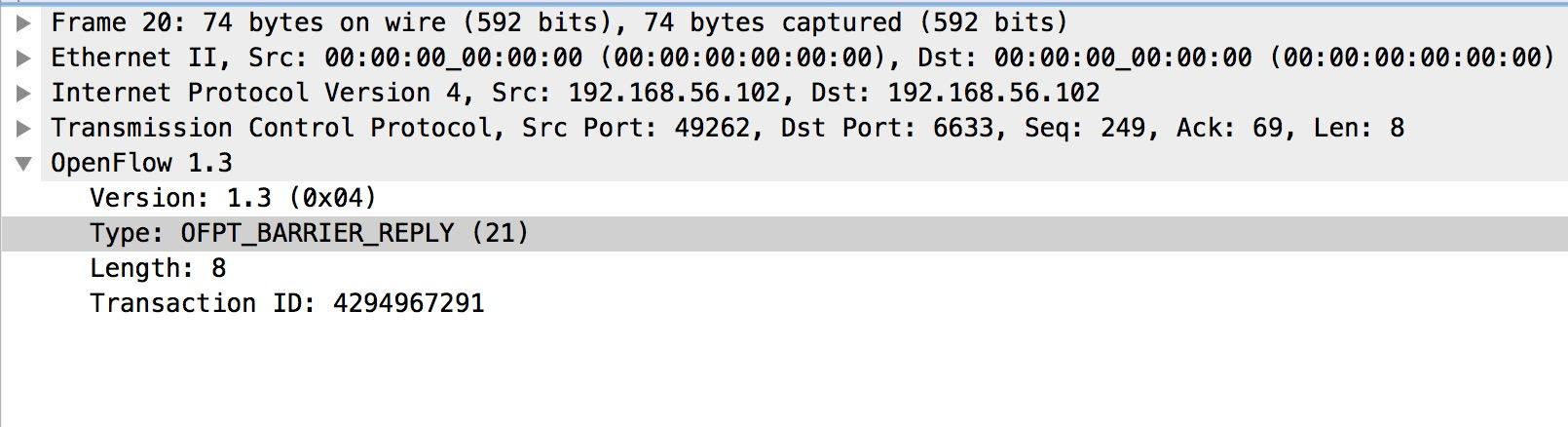


Fig 2-11. OFPT\_BARRIER\_REPLY message

#### OFPT\_ROLE\_\*

When the controller wants to change its role, it will send an OFPT\_ROLE\_REQUEST message to the switch.

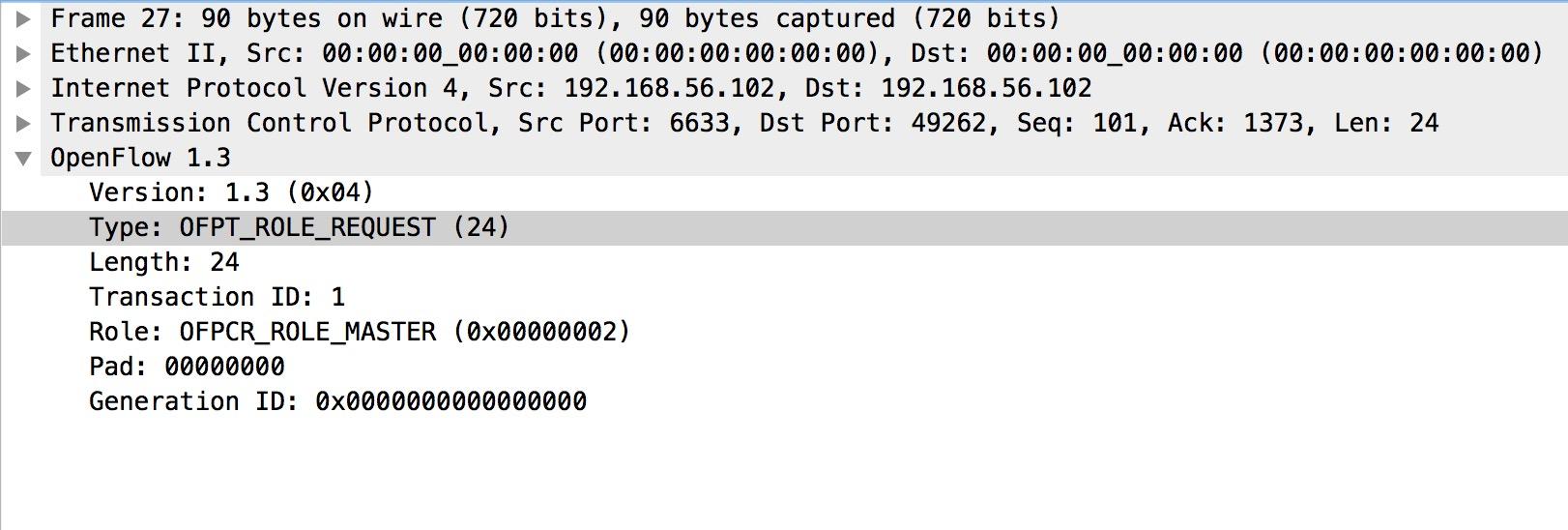


Fig 2-12. OFPT\_ROLE\_REQUEST message

When the switch receives the message, it will send a OFPT\_ROLE\_REPLY message back, containing a role field indicating the role of the controller. As we can see our controller now is master.

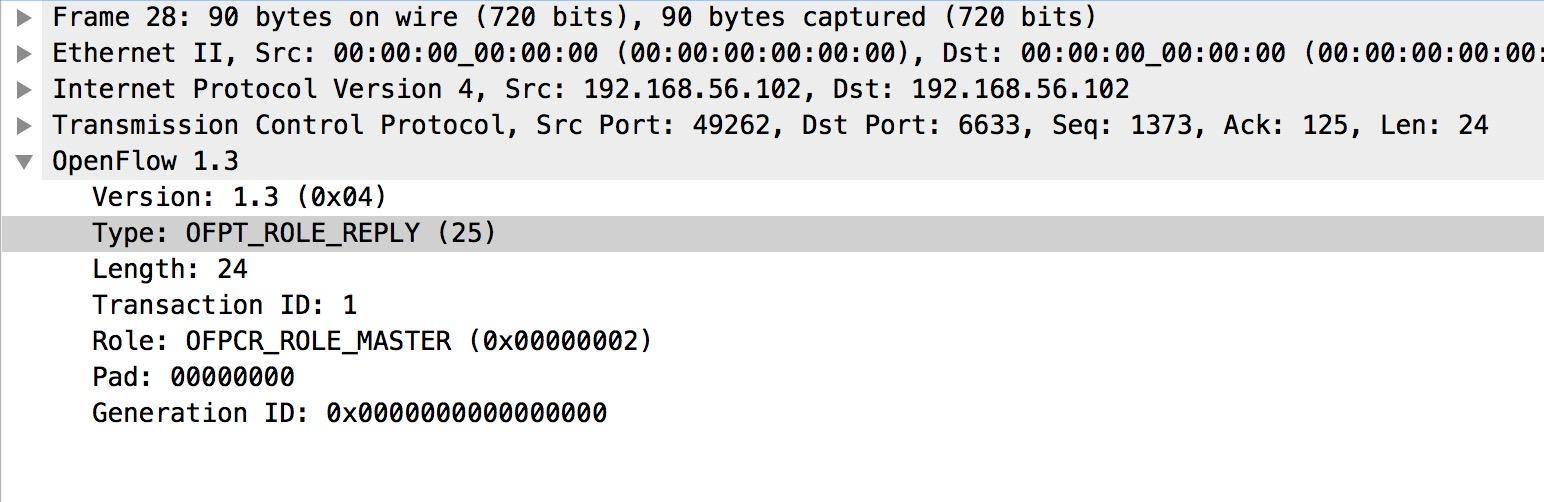


Fig 2-13. OFPT\_ROLE\_REPLY message

This operation was executed twice.

#### OFPT\_MULTIPART\_\*

When the controller is trying to get some state data from the switch, it will use OFPT\_MULITART\_REQUEST message. The switch will return one or more OFPT\_MULTPART\_REPLY message, fulfilling the request. There are many types of OFPT\_MULITART\_REQUEST messages, each have unique objective.

#### OFPMP\_DESC



Fig 2-14. OFPMP\_DESC message

OFPMP\_DESC message is for controller to ask for the vendor, hardware version, software version and serial number of the switch.

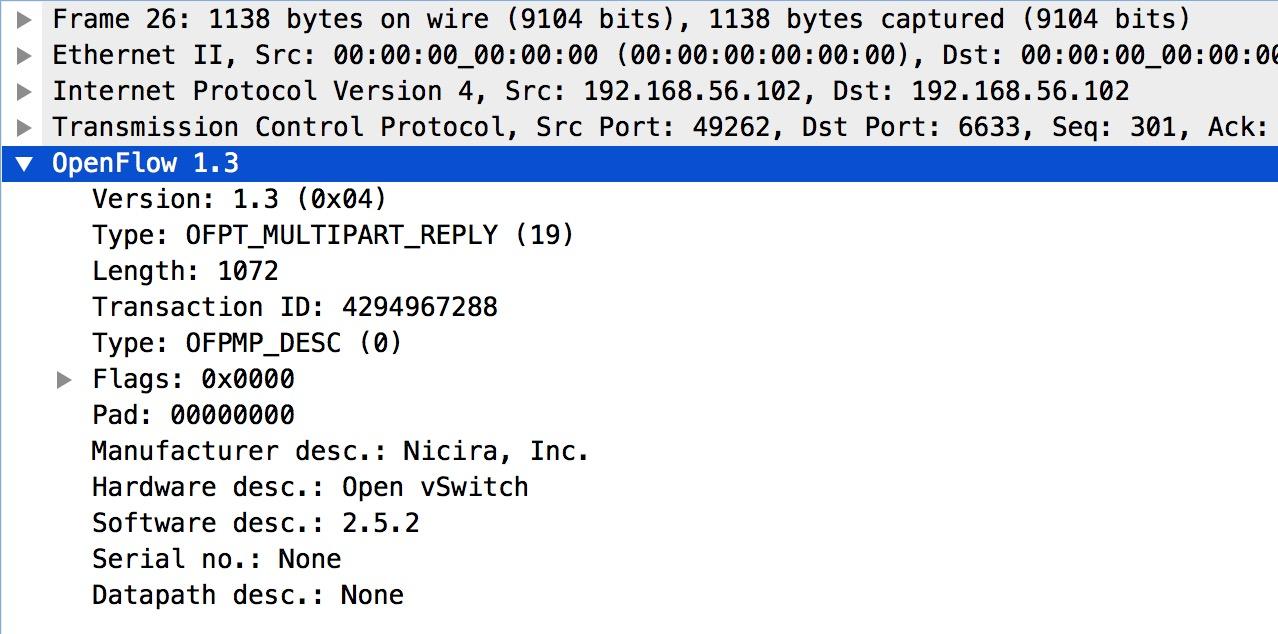


Fig 2-15. OFPMP\_DESC message

As we can see now, this switch is manufactured by Nicira, Inc. Have a software version of 2.5.2

#### OFPMP\_METER\_FEATURES

OFPMP\_METER\_FEATURES message is for features of metering subsystem.

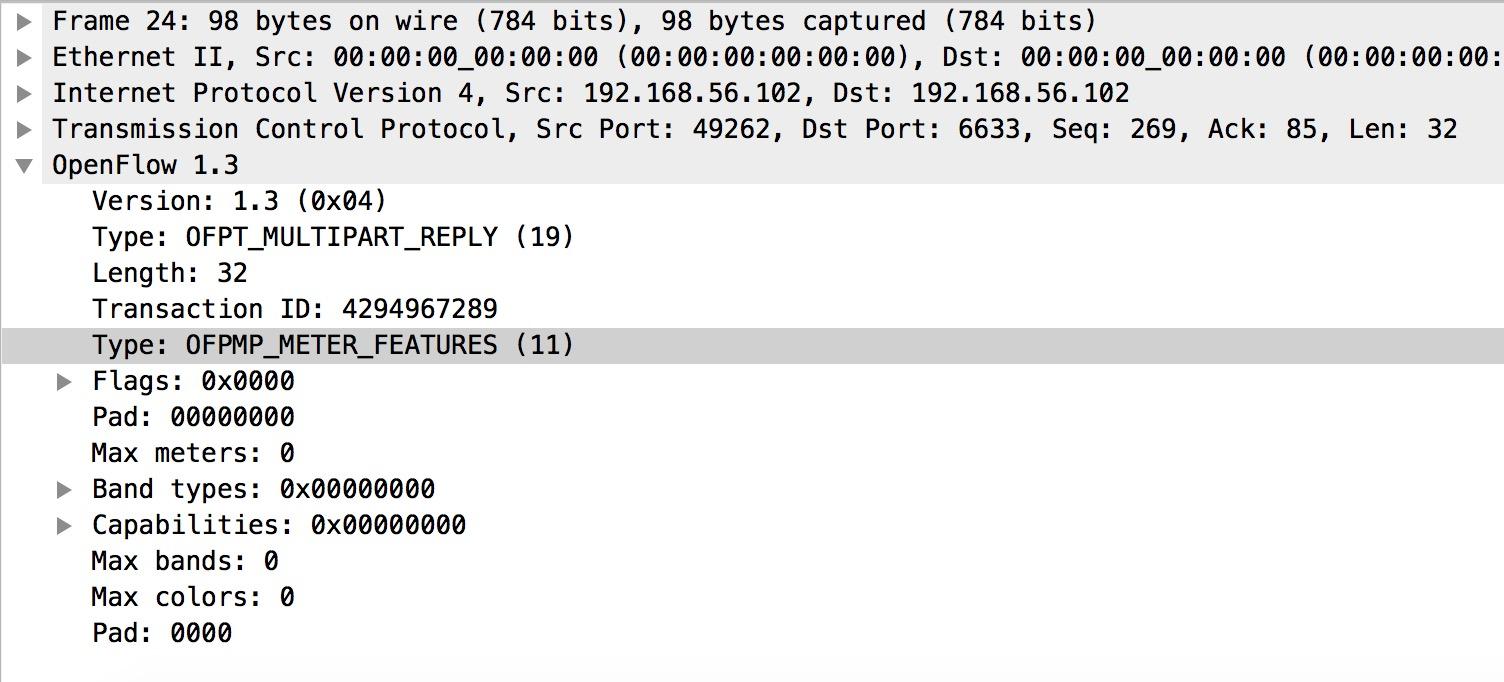


Fig 2-16. OFPMP\_METER\_FEATURES message

We can see some band types of the metering subsystem here.

#### OFPMP\_GROUP\_DESC

This message is for getting description of the group, and group's capabilities.

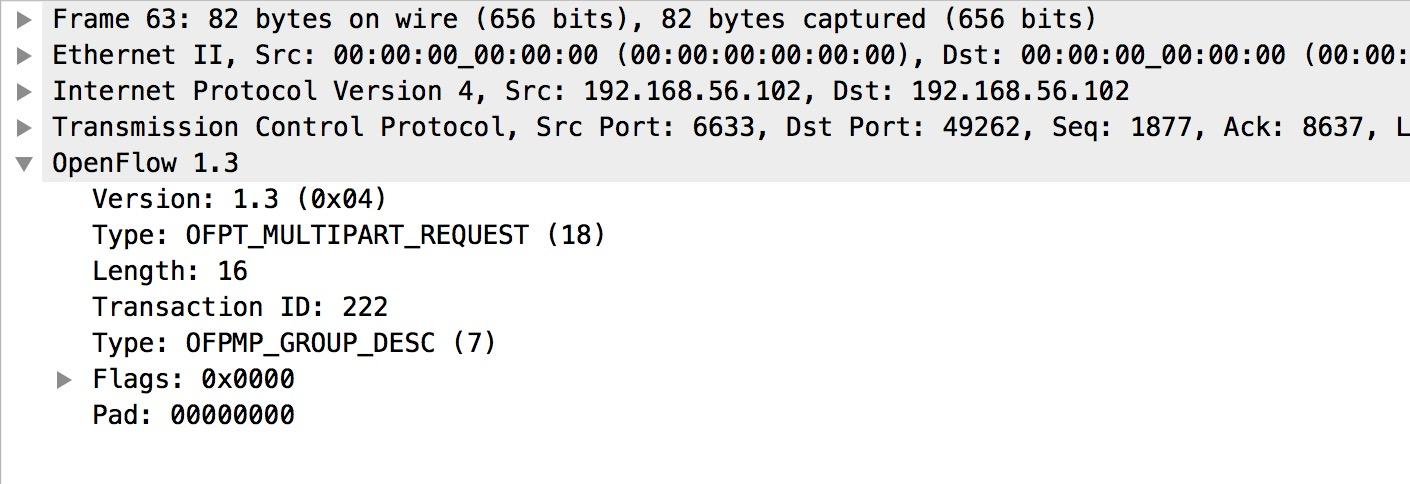


Fig 2-17. OFPMP\_GROUP\_DESC message

#### OFPMP\_PORT\_DESC

As OpenFlow 1.3 have removed ofp\_phy\_port[] field from switch\_feature message, The controller now use OFPMP\_PORT\_DESC message to get the description information of all ports that support OpenFlow on this switch.

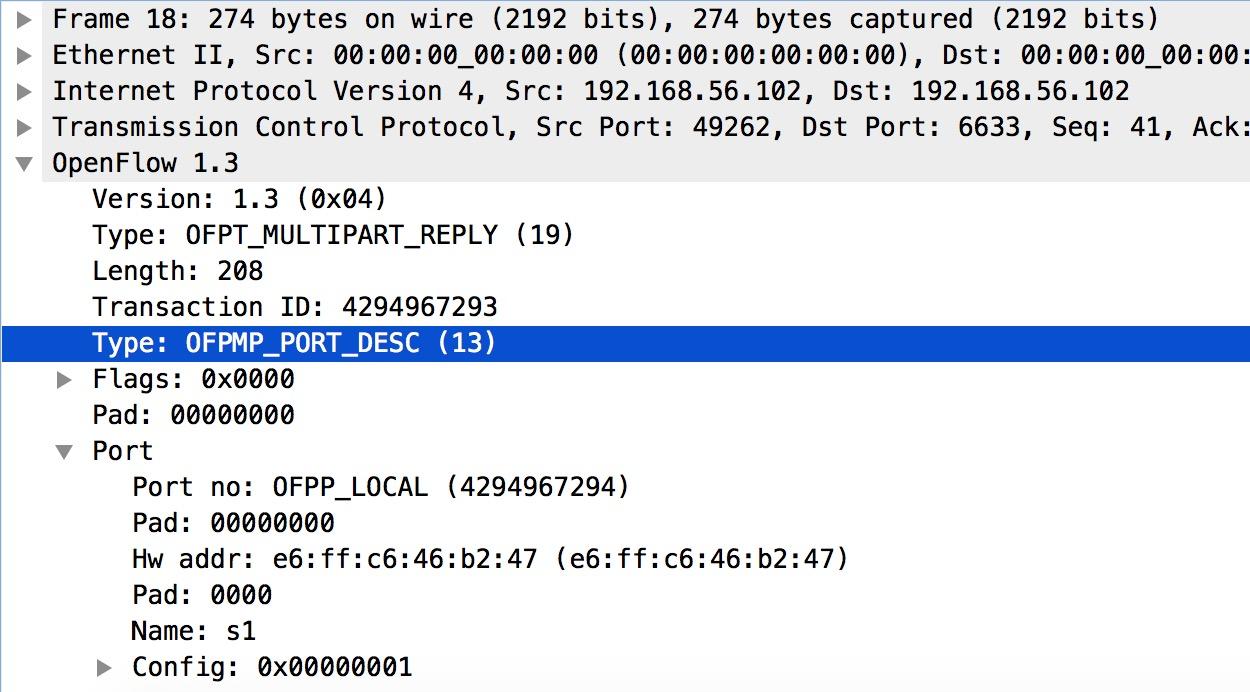


Fig 2-18. OFPMP\_PORT\_DESC message

Here you can see some information of the port like its name, number and hardware address.

#### OFPMP\_PORT\_STATS

The controller use this message to get some statistics information about the port.

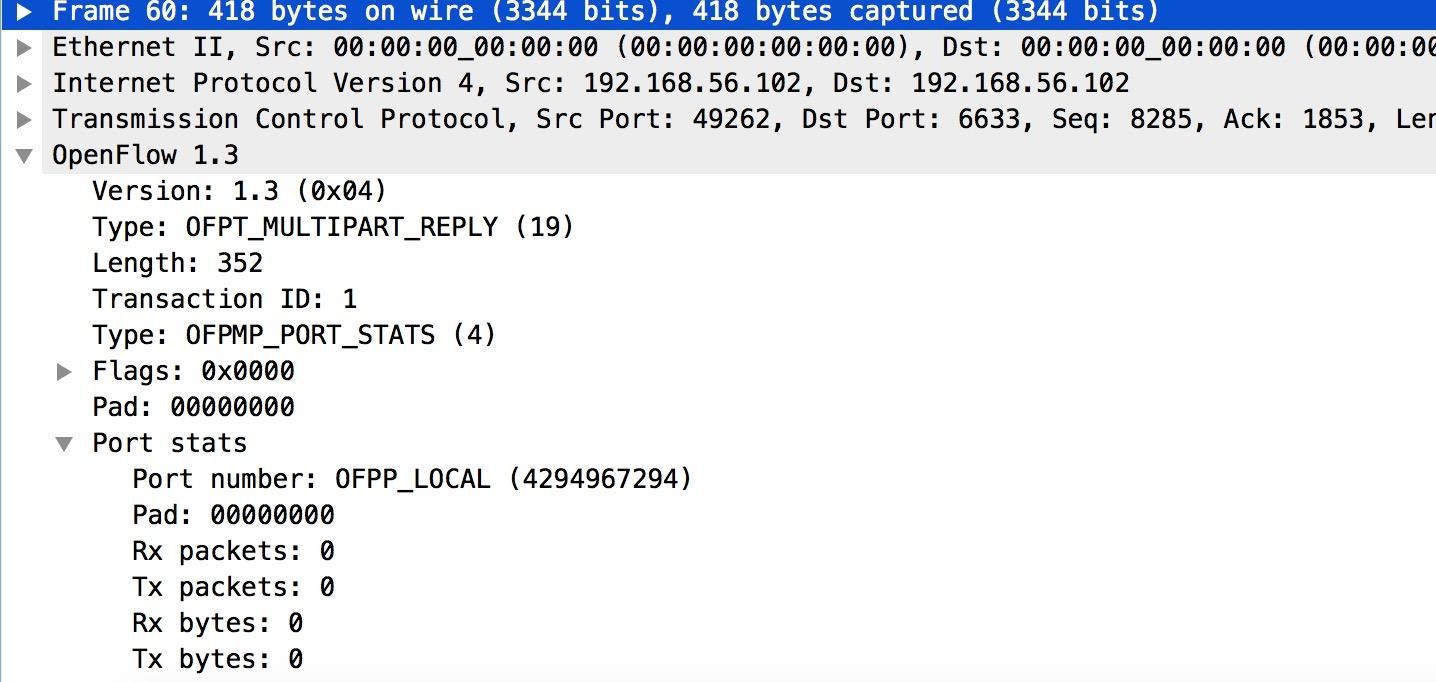


Fig 2-19. OFPMP\_PORT\_DESC message

#### OFPMP\_TABLE

Controller use this message to get table statistics information of the switch.

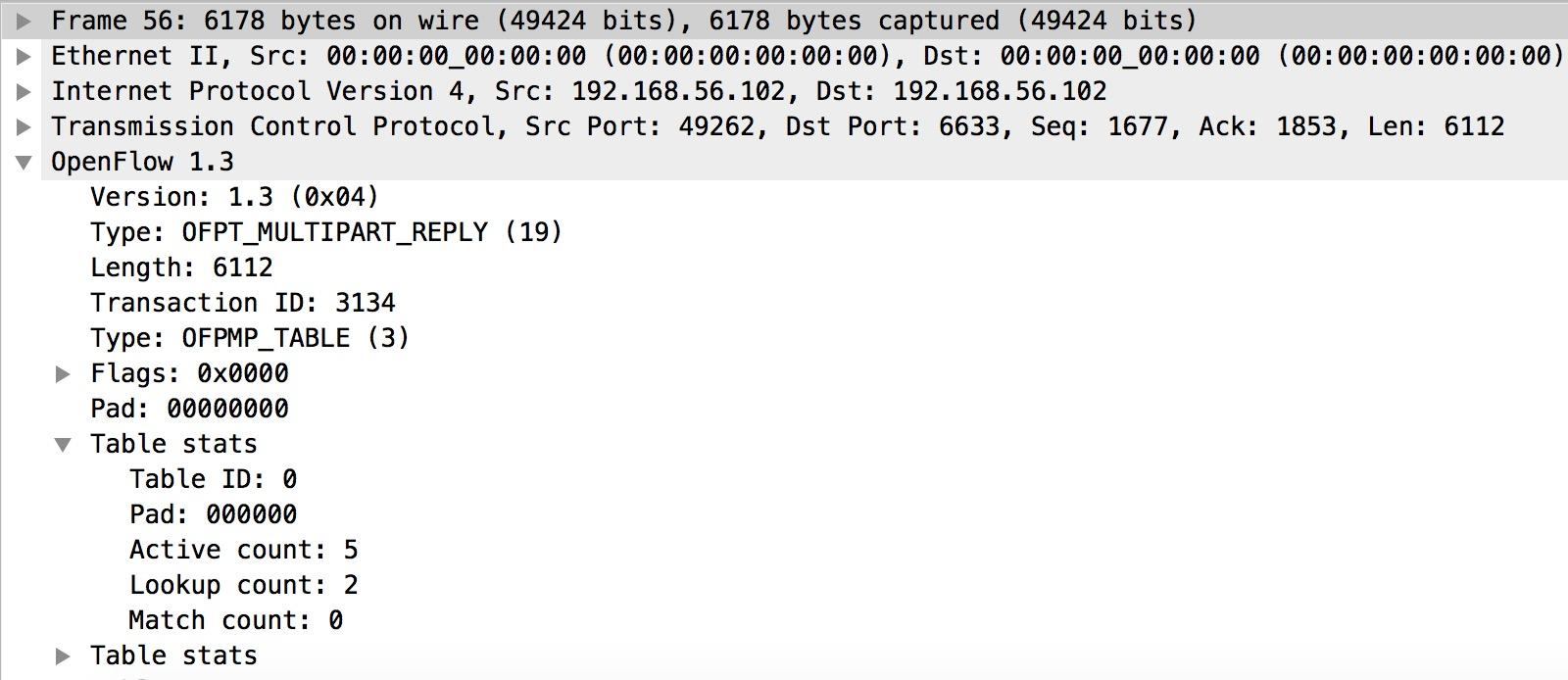


Fig 2-20. OFPMP\_TABLE message

#### OFPMP\_FLOW

Controller use this message to get flow statistics information of the switch.

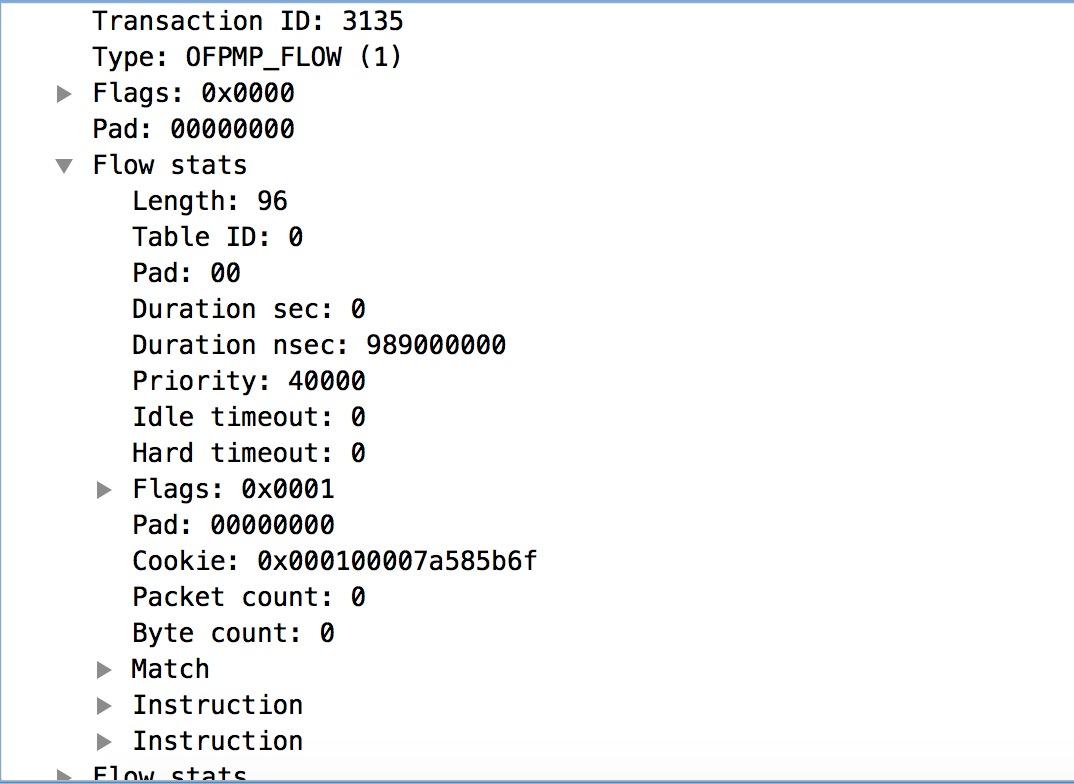


Fig 2-21. OFPMP\_FLOW message

#### OFPT\_FLOW\_MOD

OFPT\_FlOW\_MOD consists of header+match+flow\_mod+action[]. The controller send OFPT\_FLOW\_MOD messages to the switch to modify the flow tables.

Operation type in command field determined the find type of the flow\_mod action. As you can see now our command is OFPFC\_ADD.

buffer\_id：datapth buffer\_id, if it is done manually it should be -1 which means no buffer for it.

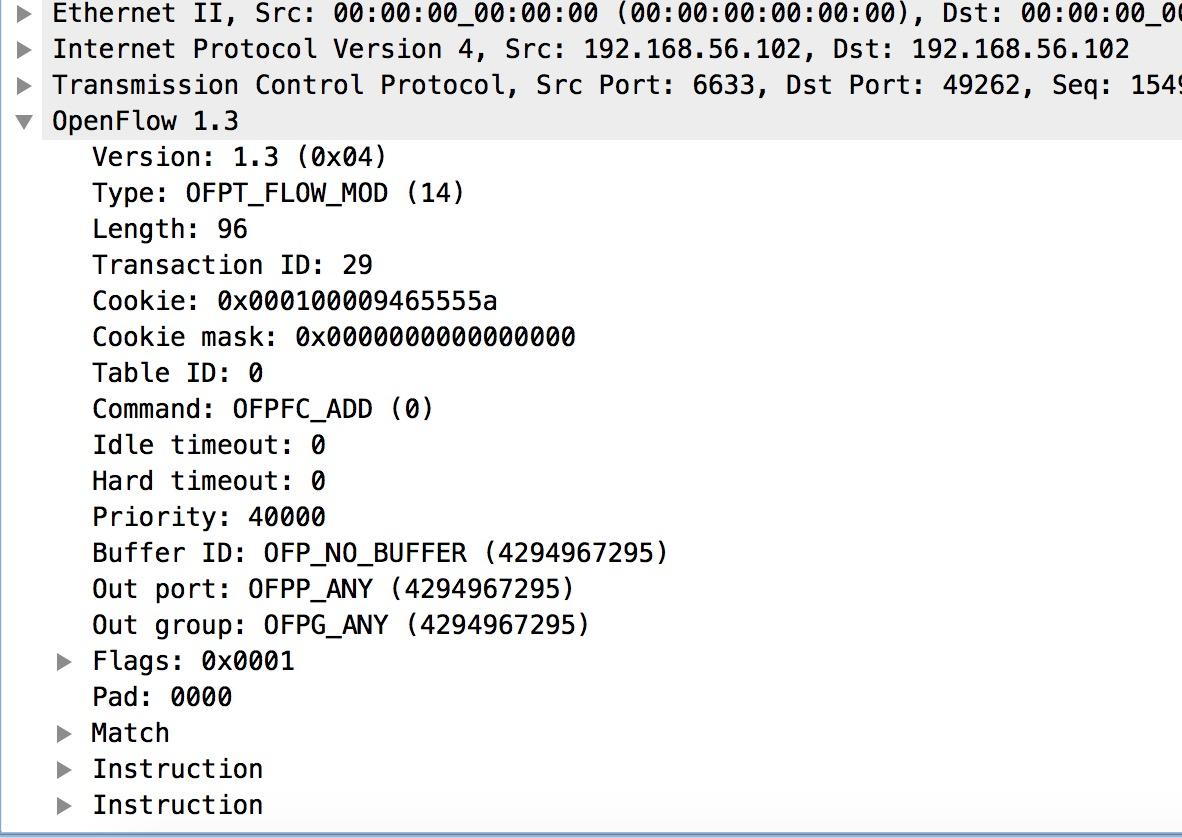


Fig 2-22. OFPT\_FLOW\_MOD message

#### OFPT\_PACKET\_IN

There are three reasons for the switch to send packet-in message to the controller: No matching flow, Action explicitly output to the controller, and packets have invalid TTL.



Fig 2-23. OFPT\_PACKET\_IN message

There could be many types of message encapsulated in packet-in message like arp,icmp. packet-in message can get a packet-out message as reply or a flow-mod message. Here we know this is an action explicitly outputted to the controller.

#### OFPT\_PACKET\_OUT

The controller encapsulated the data into LLDP packets and forward it in the switch. This message will be used only when the controller wants to send a message through the switch.

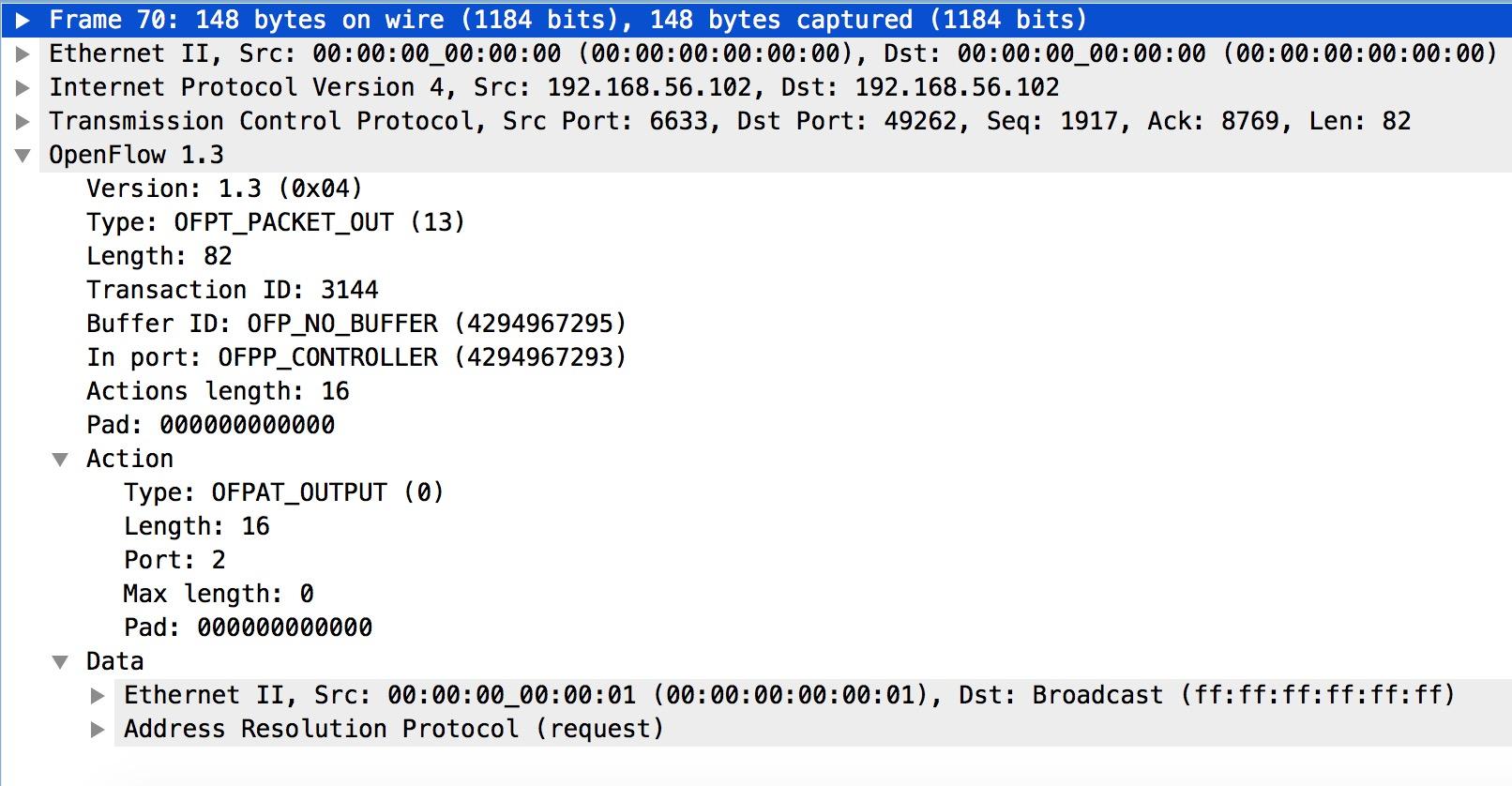


Fig 2-24. OFPT\_PACKET\_OUT message

The full list of OpenFlow messages and their order were shown in the appendix section of this paper.

# Chapter3: NETCONF

## NETCONF unfolded

### Important Terminology

As briefly introduced previously in this paper, NETCONF is a network management protocol which can be used as a south bound interface of ONOS SDN controller.

It is important to acknowledge some of the important terminologies from IETF RFC 6241 before the actual analysis.

##### Datastore: a conceptional place to store the information

##### Configuration data: the data that the device system need to transfer from one state to another.

##### Running, candidate and startup datastore: these are three types of configuration data. The first one is the current using one, the one that you are running on. The second one is the one that performs as a local version of the configuration data that will not be applied to the device upon modification. It require a commit operation and a set of validation to replace the running configuration. The last one is the one you will load to running configuration when booting up the device.

##### Server and client: In this case, the NETCONF device is the server and the SDN controller is the client.

##### message: It is a protocol element sent over a session, usually are XML documents.

##### notification: A server-initiated message indicating that some event were recognized by the server.

### Architecture

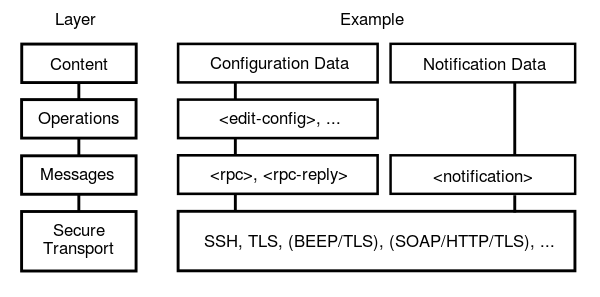


Fig 3-1. NETCONF Architecture[1]

NETCONF have four layers.

The transport layer is usually SSH or TLS, which is defined by NETCONF standard to ensure the safety of the NETCONF protocol. With encryption, It would be very difficult for the adversary to perform arbitrary operation on the NETCONF device by faking NETCONF messages.

The message layer is mainly basic rpc and rpc-reply remote procedure calls, it serves as a fundamental level on which the actual NETCONF operations built on. The rpcs can be considered as nuclear operations that cannot be divided any more. A typical rpc looks as follows:

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<some-method>

<!-- method parameters here... -->

</some-method>

</rpc>

Once the server receives the rpc message, it will reply with a rpc-reply message to notify the recognition of the rpc message. If the rpc request message have errors or some other error occurred during the communication, the server will respond with a rpc-error message indicating an error have happened.

The operation layer is the actual operations you can perform on the NETCONF devices. Note that when the NETCONF devices first establishing communications. They will exchange capabilities. It is a kind of features support by this device but not required by NETCONF. You can base your operations on capabilities as well.

The content layer consists of three types of data. The configuration data, state data and notification data. All three types of data have different usages. The configuration data is the key to the configuration of the device, it is the set of data that a device require to transform from one state to another. The state data ,on the other hand, have a purpose of carrying statistics and usually read-only. Notification data cannot be directly retrieved from a running system. It can only be send from the server to notify the client that a certain event is recognized.

NETCONF have a well-designed four level architecture. With all four layers working together, NETCONF will have a set of advantages such as flexible, robust, compatible over other protocols.

### NETCONF operations

Operations are an important part of NETCONF, they are all formatted in the form of XML and based on the RPC. Here we take a look at the most common ones to get a good understanding of how NETCONF manage and configure network devices.

#### get-config

this operation can be used get all three types of configuration data, specify the type and operation get-config, the server will respond with the corresponding configuration data you are trying to get. Example message as follows:

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<get-config>

<source>

<running/>

</source>

<filter type="subtree">

<top xmlns="http://example.com/schema/1.2/config">

<users/>

</top>

</filter>

</get-config>

</rpc>

#### edit-config

This operation allows you to edit the configuration data, you can specify the config you want to edit and you will have five options to choose. Options determined the way you edit the config. Merge means you merge the candidate config with the running one, replace means you replace the running config with candidate, create is when you want to create a new running config directly from a file, delete is when you want to delete this config but it will reply error when the target config does not exist, remove is similar to delete but it will not trigger rpc-error when the config does not exist.

You can have three type of option specified in one edit config message: default-operation, test-option and error-option. Default operation is the operation that will be performed if nothing went wrong, if something did went wrong, the error-option will be performed as well. Note that the test-option is a capability which might not be supported by some devices. Example edit config operation are as follows:

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<edit-config>

<target>

<running/>

</target>

<config>

<top xmlns="http://example.com/schema/1.2/config">

<interface>

<name>Ethernet0/0</name>

<mtu>1500</mtu>

</interface>

</top>

</config>

</edit-config>

</rpc>

#### copy-config

It create a configuration data by copying the target configuration data. Requires source target parameters.Example:

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<copy-config>

<target>

<running/>

</target>

<source>

<url>https://user:password@example.com/cfg/new.txt</url>

</source>

</copy-config>

</rpc>

#### delete-config

This operation delete a configuration data store. Example:

<rpc message-id="101"xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<delete-config>

<target>

<startup/>

</target>

</delete-config>

</rpc>

#### lock

This is an operation that you want to perform when you are trying to modify the NETCONF device without anyone else modifying it. Similar to database locks, if you performed lock operation, no other NETCONF device can interact with your current one. Example:

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<lock>

<target>

<running/>

</target>

</lock>

</rpc>

#### unlock

similar to lock operation, now just reverse the effect and remove the lock from the NETCONF device. Example:

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<unlock>

<target>

<running/>

</target>

</unlock>

</rpc>

#### close-session

This is a mild way to end a NETCONF session, the server will release the locks and stop interaction with this client. If the server replies with an ok message it means the session is successfully closed. Example:

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<close-session/>

</rpc>

#### kill-session

This is an operation that require the NETCONF server to abort all of its current work and end the session immediately. Also the server will need to revert the configuration changes if it receives the kill-session operation rpc when it is still processing a commit confirm message. Example:

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<kill-session>

<session-id>4</session-id>

</kill-session>

</rpc>

## NETCONF setup on ONOS

### Prerequisites

Now we demonstrate how do we use NETCONF on ONOS.

First we start with our machine with mininet & ONOS & OpenFlow installed.

By default the ONOS & mininet are associated with OpenFlow Protocol instead of NETCONF. In order to use it with NETCONF, we need to install of-config on our VM. It’s a wrapper of ovs protocol so we could use NETCONF to communicate with our devices. It can be done on the same virtual machine or another, we recommend here to install of-config on another machine.

#### OpenVswitch Installation

To use of-config we must have openvswitch first, First get source from openvswitch.org using command:

Note that other versions of ovs might not work with of-config.

###### wget http://openvswitch.org/releases/openvswitch-2.3.1.tar.gz

Then configure it with:

###### ./configure --prefix=/ --datarootdir=/usr/share

And then:

###### make && make install

After the Installation, start it using:

###### /usr/local/share/openvswitch/scripts/ovs-ctl start

In case you want to start it as an service:

###### sed 's,/usr/share/,/usr/local/share/,' rhel/etc\_init.d\_openvswitch > /etc/init.d/openvswitch

###### chkconfig --add openvswitch

###### chkconfig openvswitch on

#### Preinstalled of-config VM

There is a [mininet-vm](http://downloads.onosproject.org/vm/onos-ofconfig-netconf.ova) provided by ONOS project with of-config installed, you can download it if you do not want to install it yourself.

#### Install manually

of-config need libnetconf openvswitch to work while those two require libssh libxml2 libcurl libxslt and many other dependencies to work.

#### 

#### libxml2

try

###### apt-get install libxml2

###### apt-get install libxml2-dev

###### apt-get install python-libxml2

#### 

#### libcurl

try

###### apt-get install libcurl3

###### apt-get install libcurl4-openssl-dev

#### 

#### libxslt

try

###### apt-get install libxslt1-dev

#### libssh

This one you have to build and install yourself since the apt-get version is 0.6.3, which is too old for libnetconf.

###### wget https://red.libssh.org/attachments/download/218/libssh-0.7.5.tar.xz

###### unxz libssh-0.7.5.tar.xz

###### tar -xvf libssh-0.7.5.tar

###### cd libssh-0.7.5/

###### mkdir build

###### cd build/

###### cmake ..

###### make

###### sudo make install

if the error configure: error: Missing libssh (>=0.6.4) persists

try

###### apt-get remove libssh

It will remove the older version of libssh

#### dbus

try

###### apt-get install libdbus-1-dev

#### Other dependencies

try

###### apt-cache search ^libxxxx

Then install the libxxxx or libxxx-dev listed there.

#### Install Pyang

Clone the pyang from github:

###### git clone https://github.com/mbj4668/pyang.git

Then install it using:

###### cd pyang && python setup.py install

#### Install libnetconf

Clone libnetconf from github:

###### git clone https://github.com/CESNET/libnetconf

Then install it using:

###### cd libnetconf

###### ./configure && make

###### sudo make install

### Installation

#### of-config configure

Note you need openvswitch source folder to do this, your ovs directory may be different so make sure to change it.

###### ./configure --with-ovs-srcdir=/root/home/onos/openvswitch-2.3.1 PKG\_CONFIG\_PATH=/usr/local/lib/pkgconfig/

#### Install of-config

###### make

###### sudo make install

#### Run of-config

Run command

###### sudo ofc-server -v 3 -f

### Result

and if you see something like this it means your of-config was successfully installed.

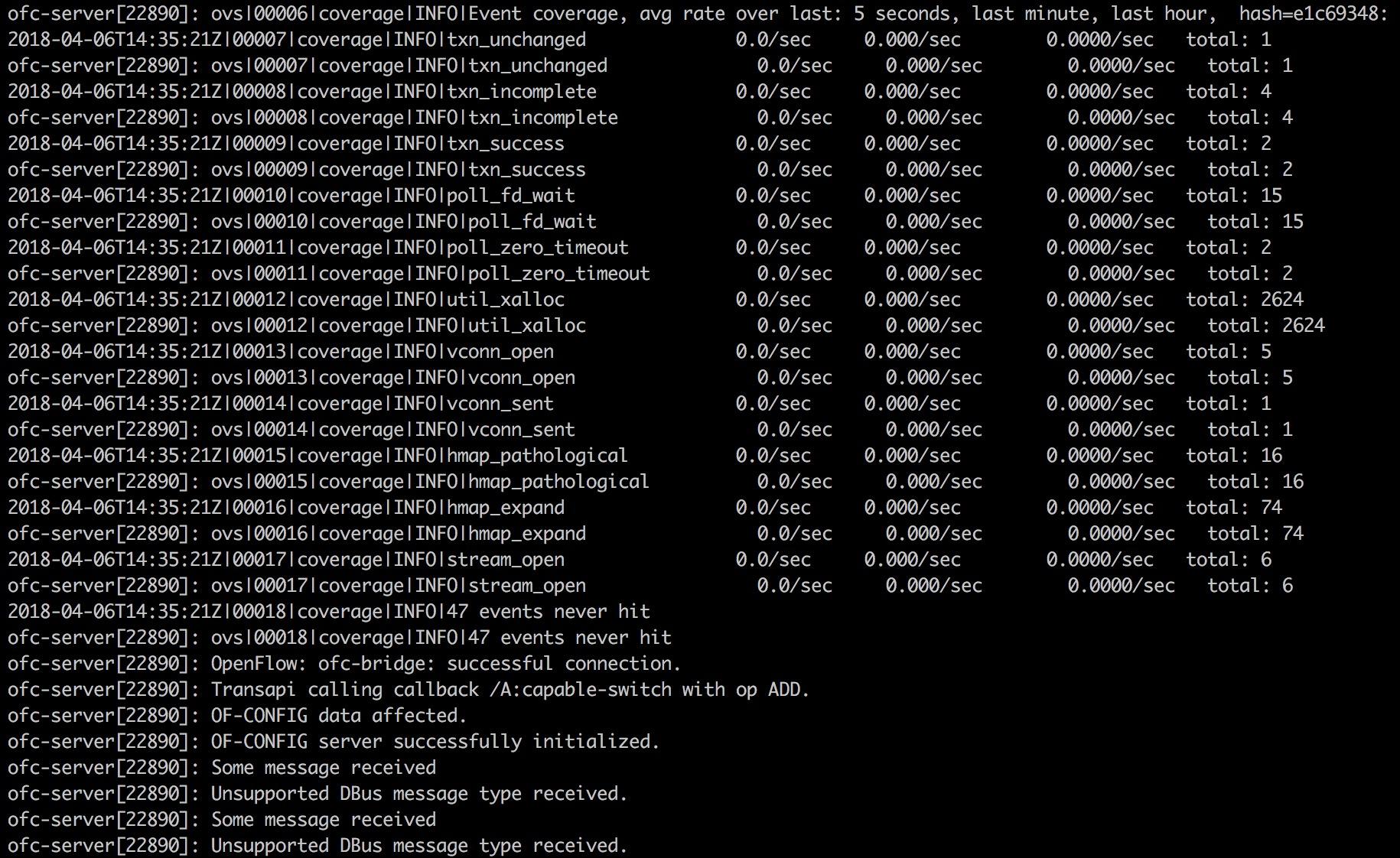


Fig 3-2. Successfully running of-config

#### Connecting of-config devices with ONOS

To continue the experiment you need at least 2 VMs. One for ONOS and another for of-config+mininet to work as a virtual device. Use clone VM function in VirtualBox if needed. Also make sure to enable inter-vm communication.

#### Start mininet-vm

This is the VM with mininet and of-config installed.

Here we have this VM at IP of 192.168.56.101, run command:

###### sudo ofc-server -v 3 -f

to start the of-config server, It will listen on port 830 now:

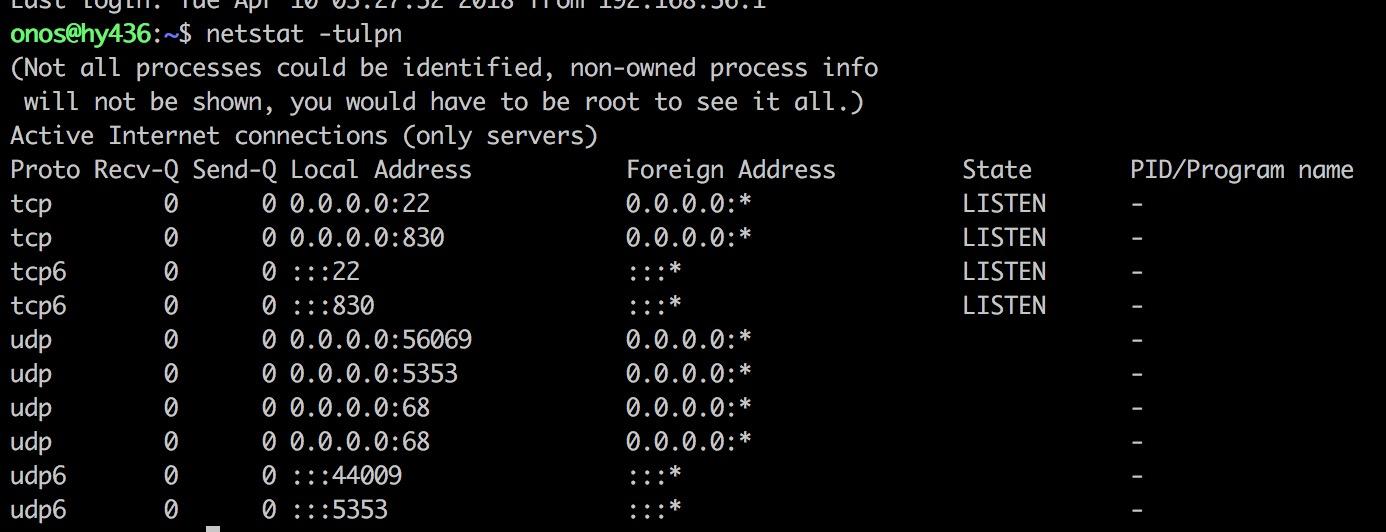


Fig 3-3. of-config listening port

#### Start ONOS-vm

This is the vm we run onos on,Here we have this VM at IP of 192.168.56.102 Run ONOS first, then in onos, type

###### app activate org.onosproject.netconf

###### app activate org.onosproject.drivers.netconf

to enable netconf & netconf drivers. Now on ONOS-vm you should edit config file provided as example at onos/tools/test/configs/netconf-cfg.json

The config files will be given in Appendix C.

Make sure that you changed the IP in both fields to your mininet-vm IP.

Finally we send the config file to ONOS using command:

###### curl -X POST -H "content-type:application/json" http://localhost:8181/onos/v1/network/configuration -d @/home/onos/onos/tools/test/configs/netconf-cfg.json --user onos:rocks

Make sure that you have changed the path of the config file to yours.

now if the IP and port are correct, we can see this result in ONOS log

you can access the ONOS log using command tl

or using ol <ip address with your onos instance> if you run ONOS remotely.

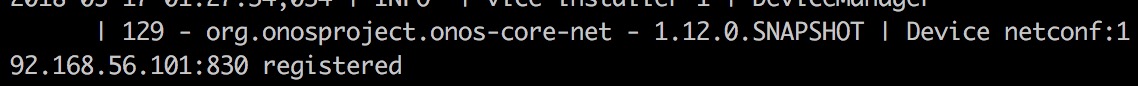


Fig 3-4. ONOS log about NETCONF device

and we can also see it in our devices list:

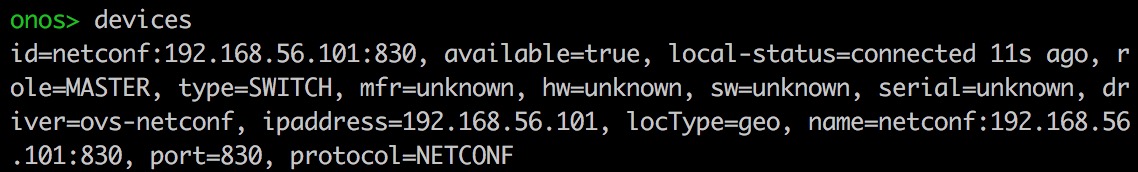


Fig 3-5. ONOS devices CLI command result

And check the mininet-vm, it indicates the success of connection.

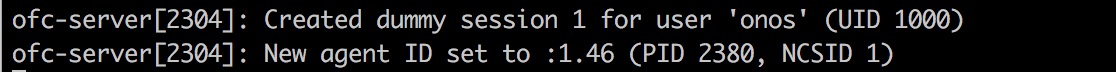


Fig 3-6. NETCONF device log

If something went wrong it will appear in onos log like

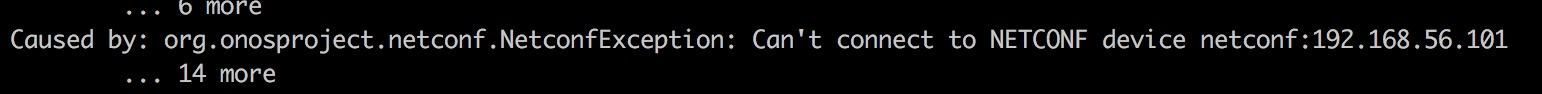


Fig 3-7. Sample Error log

Make sure to check the IP and port to see if they are correct and try again. After you finished your work with your NETCONF device, you can type this in onos cli:

###### onos> device-remove netconf:192.168.56.101:830

To get the device removed, and use ctrl+c to stop the of-config on your mininet-vm.

#### NETCONF messages capture

As the NETCONF is based on SSH protocol, we cannot analyze the packet directly through wireshark since they are encrypted.

In order to see the detailed message content between ONOS controller and NETCONF devices. Set

log:set DEBUG org.onosproject.netconf

in ONOS-cli.

Observe the log and you will find the following NETCONF messages.

First when the ONOS is establishing connection with the mininet-vm,

the ONOS controller sends hello message to the NETCONF device.

The logged message will be given in Appendix C.

Note that netconf-get-config command can only be used as a developer tool, do not base your deployment/production application’s functionality on this command.

To take full use of netconf capabilities, refer to NETCONFControllerConfig.java to see how to configure the controller in of-config capable switch via NETCONF. You can use get/set operations to achieve your target operation on the configuration.

# Chapter4: Comparison of OpenFlow and NETCONF

Now that we have learned about OpenFlow and NETCONF, we will have a deep look into those two and have a comparison between them.

OpenFlow is a protocol that focus on Traffic engineering. The most important idea about OpenFlow itself is the separation of control plane and data plane. This enabled the SDN to be built upon OpenFlow and the SDN can focus on controlling while the network devices can focus on forwarding. The centralized SDN controller can generate flow entries that contain the forwarding rule and then deploy them into the OpenFlow devices. In this way the SDN controller can manage a large amount of devices with a relatively low cost.

NETCONF, on the other hand, provided a set of tools for managing the network devices. SDN controller can obtain information from the device with the help of NETCONF and YANG APIs. NETCONF devices can send the state and configuration data to the SDN controller and apply operations issued by the controller. It established a standard RPC way to exchange message in the form of XML and have outstanding functionality and flexibility.

Here we use an example of usage of OpenFlow and NETCONF. Suppose we have a simple network topology as follows:

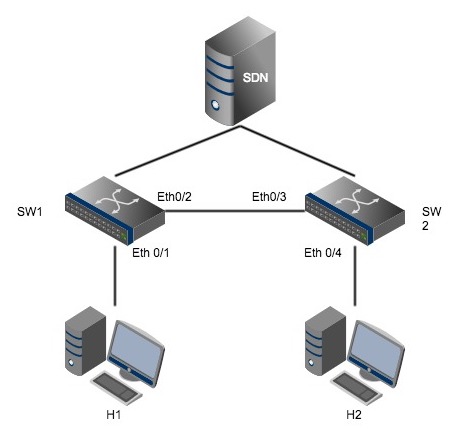


Fig 4-1. Example Network Topology

For example these devices are trying to build up a forwarding rule for H1 to communicate with H2.

When the protocol is OpenFlow, the steps are:

##### SW1 receives ARP request from H1.

##### SW1 cannot match the request, so it uses a Packet\_in to send the packet to the SDN controller.

##### SDN controller sends a Packet\_out message to all the edges of the network based on the topology it has since this is a broadcast message. And saves the Eth 0/1’s MAC address and position in the network.

##### H2 receives the arp message and send reply to SW2.

##### SW2 cannot match the reply, so it uses a Packet\_in to send the packet to the SDN controller

##### SDN controller now saves Eth 0/4’s MAC address and position in the network. Also with the information obtained earlier, SDN controller now send FLOW\_MOD message to both switches.

##### Flow\_mod key part: SW1(sMAC=H1, dMAC=H2,Output Eth0/2; sMAC=H2, dMAC=H1,Output Eth0/1) SW2 (MAC=H1, dMAC=H2,Output Eth0/4; sMAC=H2, dMAC=H1,Output Eth0/3)

##### Now SDN controller send the arp reply from H2 to SW1

##### After this H1 and H2 can communicate normally.

When the protocol is NETCONF, the steps are:

##### SDN controller using NETCONF to send learning unknown MAC address and broadcast address configuration to both SW1 and SW2.

##### SW1 receives ARP request from H1.

##### SW1 have already learned about (H1,Eth 0/1)’s MAC address, and it sends the arp message to SW2 because it is a broadcast message.

##### SW2 have already learned about (H2,Eth 0/4)’s MAC address, and it sends the arp message to H2 because it is a broadcast message.

##### H2 receives the arp message and send reply to SW2.

##### Now since there is already a MAC address of H1’s in the MAC address list of both switches, the message will be forwarded according to the MAC address list.

##### Now H1 and H2 can communicate normally.

From the example, we can see that OpenFlow sends the flow table the devices and the devices forward the packets according to it. NETCONF sends the configuration to the devices and the devices learn about the forwarding rule themselves with the configuration.

It is clear that NETCONF can modify many aspects of the device while OpenFlow only modify the flow tables. If you want to configure a device, NETCONF is the best choice. If you just want to implement some traffic rules, OpenFlow is better because of its focus on traffic engineering and sometimes it will be hard to realize the same function using the API provided by device. NETCONF’s modification is permanent and will still function after device reboot whereas OpenFlow’s flow tables will be lost after reboot.

Now we know that OpenFlow is more focused on the traffic and the packets rather than the devices since OpenFlow devices only work on keeping the flow table working. NETCONF is more about the device itself and not limited to configuration of device. In the long run, the two protocols will work together and become powerful tools in the field of SDN.

# Chapter5: System test with TestON

## Introducing TestON

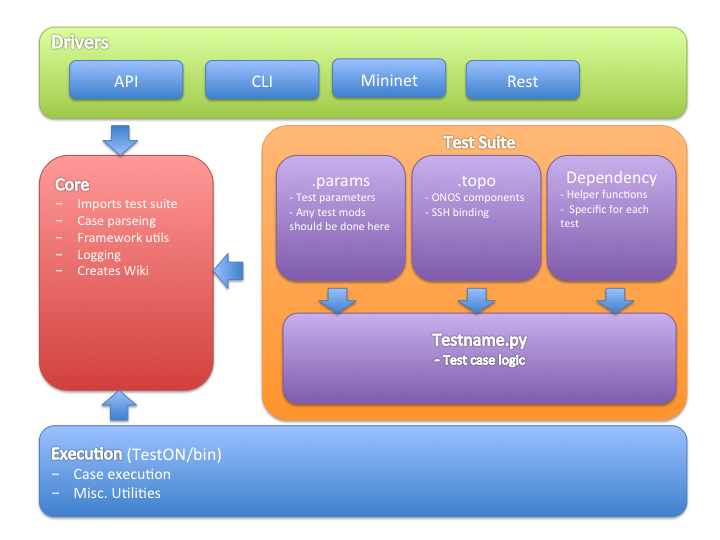


Fig 5-1. TestON Architecture[6]

TestON is a testing tool built for ONOS. It is made out of a set of drivers that can connect various components. TestON is based on Python and SSH. As it connect to a VM and execute a set of Python test scripts. It is extremely useful when trying to perform test on ONOS.

In this section of the paper, we are going to introduce how to use TestON with NETCONF.

## Exemplary TestON usage

TestON is the testing platform for all the ONOS tests.

You can refer to TestON's wiki page to see the installation guide.

After you successfully installed TestON on your environment. We will need to see the way we use it with NETCONF.

#### Prerequisites

In order to use TestON properly you need to set up an user named sdn and setup passwordless login between nodes.

To achieve this first you need to add the sdn user as sudoer:

###### 

###### sudo visudo

###### sdn ALL=(ALL) NOPASSWD:ALL     # Add this line at the bottom of the file

Then you can enable passwordless login for all nodes.

You need to create a .ssh folder if it doesn't exist:

###### mkdir ~/.ssh

Now enter this folder you can create the SSH key:

###### ssh-keygen -t rsa

Note that do not specify passphrase with your SSH key.

Then you can deploy your keys to your nodes either manually or automatically.

#### Manually

###### vi ~/.ssh/authorized\_keys     # Create the authorized\_keys file and copy into it your management machine public key (you can find it on the managemnt machine under .ssh/id\_rsa.pub)

###### chmod 700 ~/.ssh               # Give the correct permissions to your .ssh directory

###### chmod 600 ~/.ssh/authorized\_keys

#### Automatically

###### onos-push-keys $TARGET\_MACHINE\_IP

Note that if you do it automatically you must have already created sdn user as a sudoer on the target machine and set it up with a password.

#### Test files

First we need to prepare the Test files for TestON, they are necessary to perform a test.

There are three types of file we need, first, the .params file, which contains user defined variables, as well as specifies the order of which test cases to run. The .topo file, which defines the components and options that TestON will use to execute the test. User name, password, IP addresses, drivers, and/or a Mininet topology are specified in this file. The .py file, which contains the all the test cases.

The test file we used here will be provided in Appendix D.

Make sure to create a directory under your TestON's tests folder, and name all three files with the same name you use on your directory. Here we use NETCONF\_Test so we have a NETCONF\_Test folder under /OnosSystemTest/TestON/tests/ and three files NETCONF\_Test.params, NETCONF\_Test.py and NETCONF\_Test.topo under that directory.

#### Running

Now that we created our test files and put them at the right place, we can perform our test. First we need ONOS running in order to run TestON.

Enter the directory of OnosSystemTest/TestON/bin, we type:

###### $ sudo ./cli.py run NETCONF\_Test

 If you stop a test while running (by ^C or ^D), make sure to clean up before running a new test.

###### $ ./cleanup.sh

If you have set the TestON correctly, A message will show up as follows

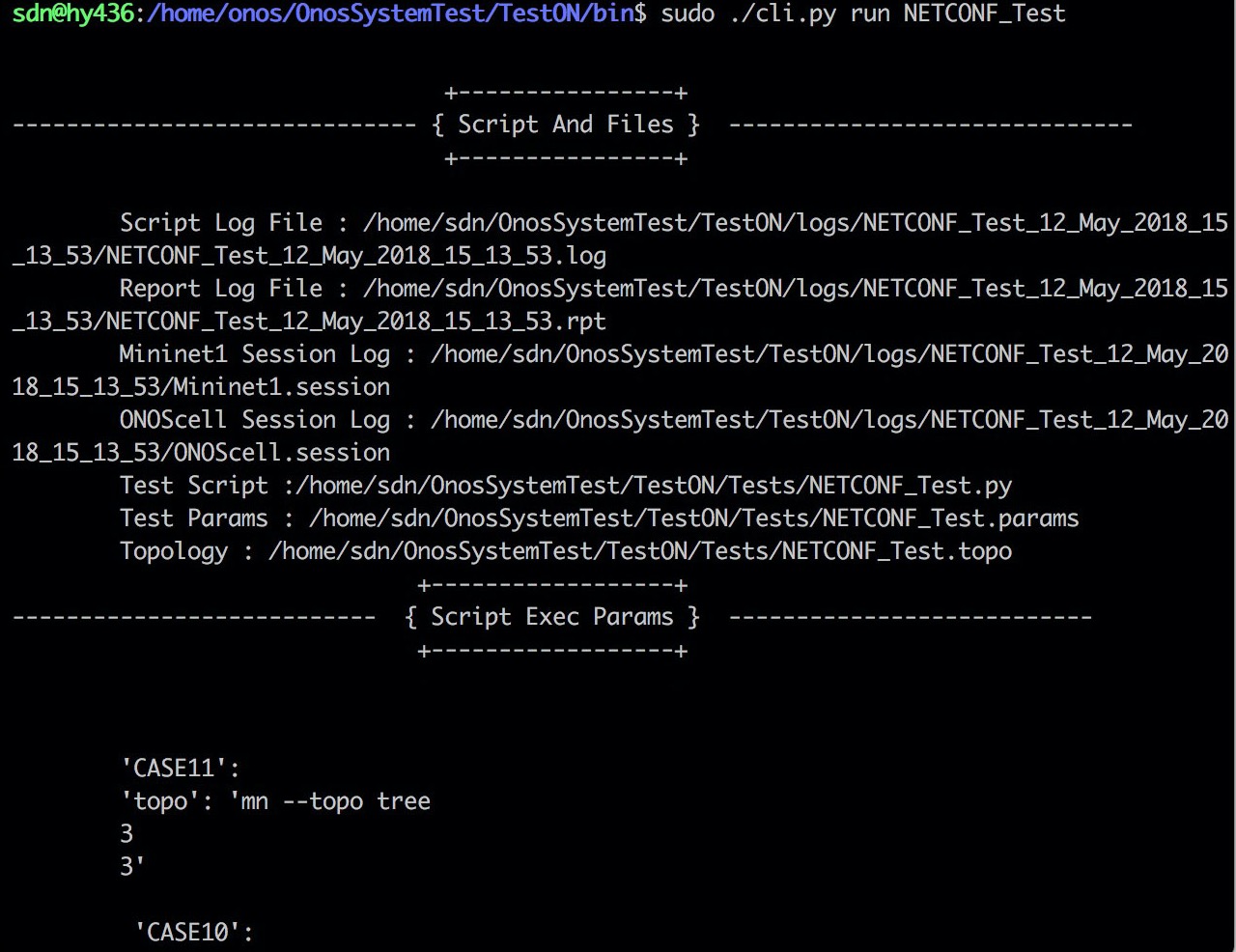


Fig 5-2. TestON running

If you passed the test, something like this will show up

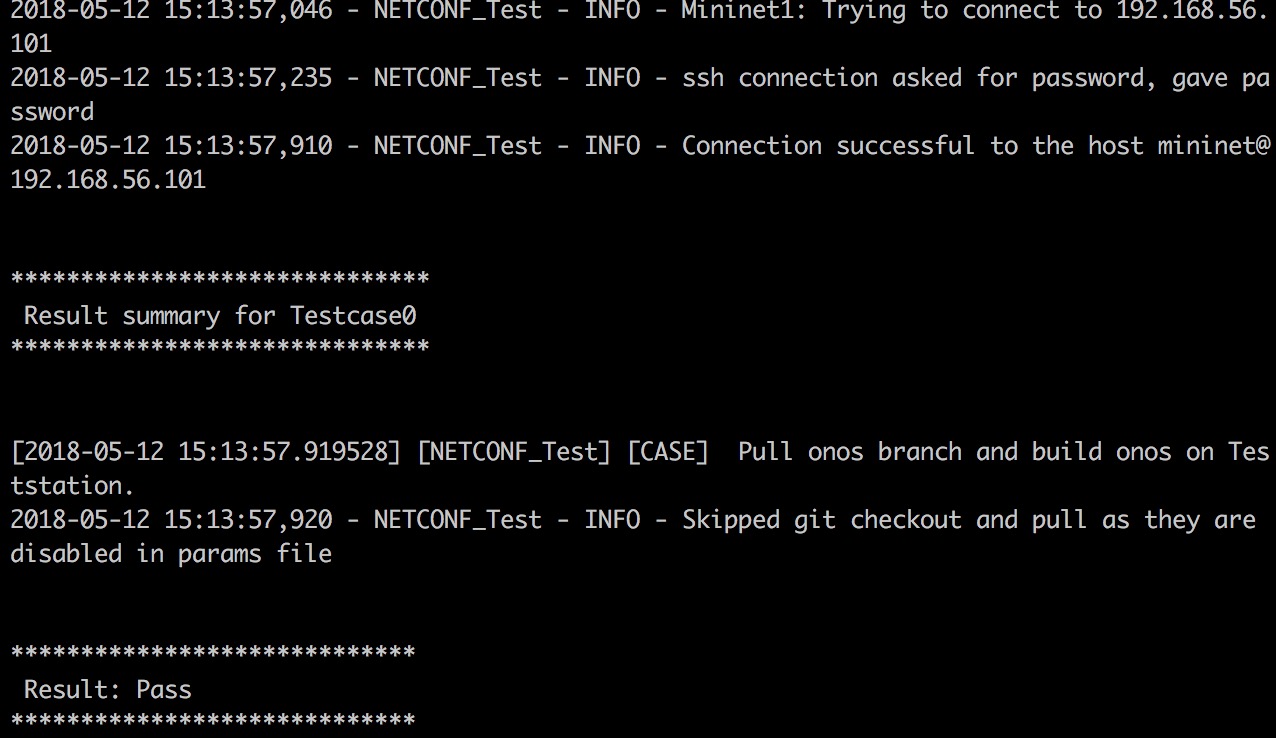


Fig 5-3. TestON result pass

And finally you can see your test result summary after all tests is complete:

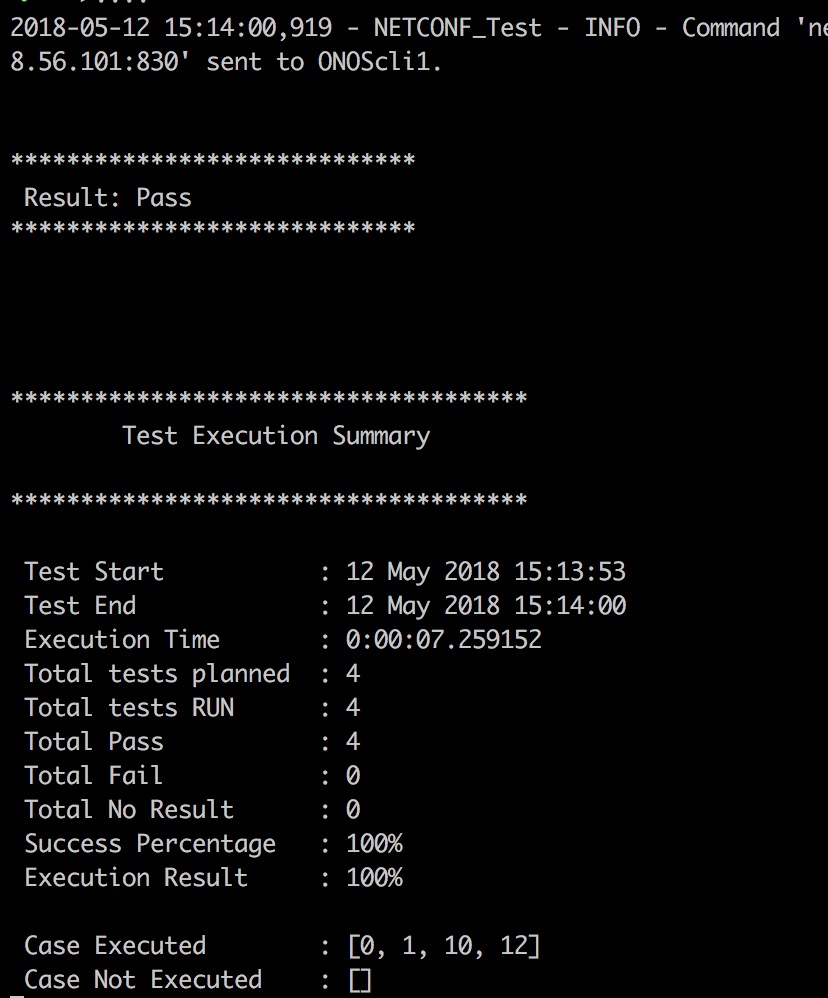


Fig 5-4. TestON result summary

After this you can refer to the Script Log file folder displayed from above and retrieve the log file for more testing result details. The test result of this test will be given in Appendix D.

# Chapter6: Optical network use-case with LINC

## Optical network and LINC

In the field of optical network, the reconfiguration of device is much harder to accomplish since the reconnection of optical cable is of high cost. The need of NETCONF in optical network is much higher than traditional ethernet based network environment. Also, It’s usually more often for optical network to be used in big data centers, where NETCONF is even more useful for its high performance and flexibility.

## Use-case showcase

To showcase the useage of NETCONF with Optical switch on ONOS, we need LINC switch to simulate an optical device.

### Installation of LINC switch

You can refer to [LINC switch github page](https://github.com/FlowForwarding/LINC-Switch) for installation guide. First we need to resolve the dependencies.

LINC-switch is written in Erlang, we need

###### sudo apt-get install erlang

###### in order to get the environment for it to run.

In some cases when the LINC does not work with your current version of Erlang or the Erlang binary provided by the system is too old, you want to build and install Erlang from source yourself, then just hit the [Erlang download page](http://www.erlang.org/downloads) and get the source, then build it yourself. In our case we used R16B03.

#### Dependencies:

###### sudo apt-get install libwxgtk3.0-dev fop libncurses5-dev unixodbc-dev libssh-dev libpng3 libxml2-utils

Then enter the Erlang folder and

###### ./configure

###### make

###### sudo make install

And LINC need some other libraries to run.

###### sudo apt-get install git-core bridge-utils libpcap0.8 libpcap-dev libcap2-bin uml-utilities

Now clone the LINC-switch repo and enter the repo, before you make, you need to edit copy rel/files/sys.config.orig to rel/files/sys.config and edit it by hand to fit your setup.

You can edit the part related to netconf in sys.config to fit your environment. we used default here. We will provide the sys.config file we used in appendix E.

After that we can start to make

###### make

You can see these messages which indicate that you have it successfully built.

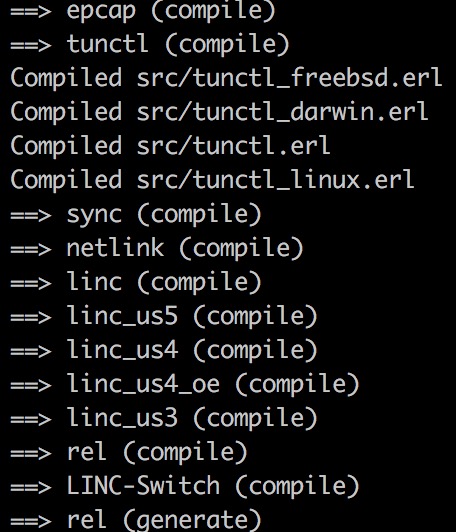


Fig 6-1. LINC-switch built result

After making it, type:

###### rel/linc/bin/linc console

to start LINC switch in console mode.

### Connect to LINC

Now we need to activate the NETCONF drivers in onos.

###### app activate org.onosproject.netconf

###### app activate org.onosproject.drivers.netconf

and prepare the configuration file for LINC switch and then upload it to onos controller.

The configuration file we used will be provided in appendix E.

then upload it to ONOS controller using REST command:

###### curl -X POST -H "content-type:application/json" http://localhost:8181/onos/v1/network/configuration -d @/home/onos/onos/tools/test/configs/netconf-LINC.json --user onos:rocks

After it's connected, you can see it in devices list:

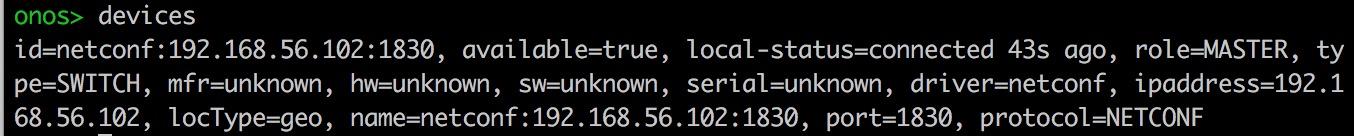


Fig 6-2 LINC switch connected

# References

[1] Internet Engineering Task Force (IETF). RFC 6241 NETCONF[EB/OL]. <https://tools.ietf.org/html/rfc6241>, 2011-6.

[2] Internet Engineering Task Force (IETF). RFC 6020 YANG[EB/OL]. <https://tools.ietf.org/html/rfc6020>, 2010-10.

[3] Wallin S. Automating network and service configuration using NETCONF and YANG[C]// International Conference on Large Installation System Administration. USENIX Association, 2011:22-22.

[4] Carl Moberg. A 30-minute Introduction to NETCONF and YANG[EB/OL]. <https://www.slideshare.net/cmoberg/a-30minute-introduction-to-netconf-and-yang>, 2011-11-18.

[5] Stancu A, Vulpe A, Fratu O, et al. Wireless Transport Emulator Based on LINC Software Switch[J]. Wireless Personal Communications, 2017(8):1-17.

[6] Kelvin Flores. ONOS TestON Tutorial[EB/OL]. <https://wiki.onosproject.org/pages/viewpage.action?pageId=2133836>, 2017-7-20.

[7] Andrea Campanella. ONOS NETCONF Tutorial[EB/OL]. <https://wiki.onosproject.org/display/ONOS/NETCONF>, 2017-11-27.

[8] O'Connor B. A Mininet-based Virtual Testbed for Distributed SDN Development[C]// ACM Conference on Special Interest Group on Data Communication. ACM, 2015:365-366.

[9] The Open Networking Foundation. OpenFlow switch specification[EB/OL]. <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>, 2015-3-26.

[10] marcsugiyama. LINC-switch github page[EB/OL]. <https://github.com/FlowForwarding/LINC-Switch>, 2015-7-16.

[11] rkrejci. of-config Installation guide[EB/OL].

<https://github.com/openvswitch/of-config/blob/master/INSTALL.md>, 2015-4-7.

[12] Linux Foundation. Openvswtich documentation[EB/OL].

<http://docs.openvswitch.org/en/latest/>, 2016.

[13] Juniper Networks. NETCONF XML Management Protocol Developer Guide[EB/OL]. <https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/netconf-guide/netconf.pdf>, 2018-04-17.

[14] rkrejci. libnetconf github page[EB/OL]. <https://github.com/CESNET/libnetconf>, 2015-11-20.

[15] OpenDaylight Project. Opendaylight NETCONF user guide[EB/OL]. <http://docs.opendaylight.org/en/stable-oxygen/user-guide/netconf-user-guide.html>, 2018.

[16] Ayaka Koshibe. ONOS System Components[EB/OL]. <https://wiki.onosproject.org/display/ONOS/System+Components>, 2016-9-7.

[17] Prayson Pate. NFV and SDN: What’s the Difference?[EB/OL].

<https://www.sdxcentral.com/articles/contributed/nfv-and-sdn-whats-the-difference/2013/03/>, 2013-3-30.

[18] Quentin Monnet. An introduction to SDN[EB/OL]. <https://qmonnet.github.io/whirl-offload/2016/07/08/introduction-to-sdn/>, 2016-7-8.

# Appendix

## Appendix A: Environment setup on ONOS

My environment:

##### Hardware:

##### Dual core CPU i5-5275U

##### 2.9GB ram

##### 18.9 GB Disk Space

##### Software:

##### Ubuntu 16.04 LTS 64-bit

##### ONOS 3.0.8

##### OpenFlow 1.3

To install and run ONOS, first get the ONOS

###### sudo wget -c http://downloads.onosproject.org/release/onos-$ONOS\_VERSION.tar.gz

Then untar it and rename it to ONOS:

###### sudo tar xzf onos-$ONOS\_VERSION.tar.gz

###### sudo mv onos-$ONOS\_VERSION onos

Verify it works:

###### /opt/onos/bin/onos-service start

After that install it as a service:

###### sudo cp /opt/onos/init/onos.initd /etc/init.d/onos

Check ONOS wiki for addition steps that you might need to take.

And you can start ONOS service by

###### sudo service onos {start|stop|status}

To run ONOS locally you can just enter the ONOS folder and type:

###### ok clean

Access the ONOS CLI using:

###### onos <ip that onos instance are running on>

## Appendix B: OpenFlow Packets

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Source | Destination | Protocol | Length | Info |
| 1 | 192.168.56.102 | 192.168.56.102 | TCP | 74 | 49260 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK\_PERM=1 TSval=945431 TSecr=0 WS=512 |
| 2 | 192.168.56.102 | 192.168.56.102 | TCP | 74 | 6633 > 49260 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK\_PERM=1 TSval=945431 TSecr=945431 WS=512 |
| 3 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49260 > 6633 [ACK] Seq=1 Ack=1 Win=44032 Len=0 TSval=945431 TSecr=945431 |
| 4 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49260 > 6633 [FIN, ACK] Seq=1 Ack=1 Win=44032 Len=0 TSval=945431 TSecr=945431 |
| 5 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49260 [FIN, ACK] Seq=1 Ack=2 Win=44032 Len=0 TSval=945431 TSecr=945431 |
| 6 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49260 > 6633 [ACK] Seq=2 Ack=2 Win=44032 Len=0 TSval=945431 TSecr=945431 |
| 7 | 192.168.56.102 | 192.168.56.102 | TCP | 74 | 49262 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK\_PERM=1 TSval=945485 TSecr=0 WS=512 |
| 8 | 192.168.56.102 | 192.168.56.102 | TCP | 74 | 6633 > 49262 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK\_PERM=1 TSval=945485 TSecr=945485 WS=512 |
| 9 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49262 > 6633 [ACK] Seq=1 Ack=1 Win=44032 Len=0 TSval=945485 TSecr=945485 |
| 10 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_HELLO |
| 11 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=1 Ack=9 Win=44032 Len=0 TSval=945488 TSecr=945488 |
| 12 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 82 | Type: OFPT\_HELLO |
| 13 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49262 > 6633 [ACK] Seq=9 Ack=17 Win=44032 Len=0 TSval=945490 TSecr=945490 |
| 14 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_FEATURES\_REQUEST |
| 15 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49262 > 6633 [ACK] Seq=9 Ack=25 Win=44032 Len=0 TSval=945490 TSecr=945490 |
| 16 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 98 | Type: OFPT\_FEATURES\_REPLY |
| 17 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 82 | Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_PORT\_DESC |
| 18 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 274 | Type: OFPT\_MULTIPART\_REPLY, OFPMP\_PORT\_DESC |
| 19 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 94 | Type: OFPT\_GET\_CONFIG\_REQUEST |
| 20 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REPLY |
| 21 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 78 | Type: OFPT\_GET\_CONFIG\_REPLY |
| 22 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=69 Ack=269 Win=45056 Len=0 TSval=945493 TSecr=945493 |
| 23 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 82 | Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_METER\_FEATURES |
| 24 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 98 | Type: OFPT\_MULTIPART\_REPLY, OFPMP\_METER\_FEATURES |
| 25 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 82 | Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_DESC |
| 26 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 1138 | Type: OFPT\_MULTIPART\_REPLY, OFPMP\_DESC |
| 27 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 90 | Type: OFPT\_ROLE\_REQUEST |
| 28 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 90 | Type: OFPT\_ROLE\_REPLY |
| 29 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 90 | Type: OFPT\_ROLE\_REQUEST |
| 30 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 90 | Type: OFPT\_ROLE\_REPLY |
| 31 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 982 | Type: OFPT\_BARRIER\_REQUEST |
| 32 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 274 | Type: OFPT\_MULTIPART\_REPLY, OFPMP\_PORT\_DESC |
| 33 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REPLY |
| 34 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REPLY |
| 35 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REPLY |
| 36 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REPLY |
| 37 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=1065 Ack=1661 Win=49152 Len=0 TSval=945510 TSecr=945510 |
| 38 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 187 | Type: OFPT\_PACKET\_OUT |
| 39 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 187 | Type: OFPT\_PACKET\_OUT |
| 40 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49262 > 6633 [ACK] Seq=1661 Ack=1307 Win=45568 Len=0 TSval=945511 TSecr=945511 |
| 41 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 187 | Type: OFPT\_PACKET\_OUT |
| 42 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 187 | Type: OFPT\_PACKET\_OUT |
| 43 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49262 > 6633 [ACK] Seq=1661 Ack=1549 Win=45568 Len=0 TSval=945511 TSecr=945511 |
| 44 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 162 | Type: OFPT\_FLOW\_MOD |
| 45 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REQUEST |
| 46 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49262 > 6633 [ACK] Seq=1661 Ack=1653 Win=45568 Len=0 TSval=945512 TSecr=945512 |
| 47 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REPLY |
| 48 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 162 | Type: OFPT\_FLOW\_MOD |
| 49 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REQUEST |
| 50 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49262 > 6633 [ACK] Seq=1669 Ack=1757 Win=45568 Len=0 TSval=945522 TSecr=945521 |
| 51 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REPLY |
| 52 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=1757 Ack=1677 Win=49152 Len=0 TSval=945534 TSecr=945522 |
| 53 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 82 | Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_TABLE |
| 54 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 122 | Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_FLOW |
| 55 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 90 | Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_PORT\_STATS |
| 56 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 6178 | Type: OFPT\_MULTIPART\_REPLY, OFPMP\_TABLE |
| 57 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=1853 Ack=7789 Win=180224 Len=0 TSval=945757 TSecr=945757 |
| 58 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 562 | Type: OFPT\_MULTIPART\_REPLY, OFPMP\_FLOW |
| 59 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=1853 Ack=8285 Win=192512 Len=0 TSval=945757 TSecr=945757 |
| 60 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 418 | Type: OFPT\_MULTIPART\_REPLY, OFPMP\_PORT\_STATS |
| 61 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=1853 Ack=8637 Win=204800 Len=0 TSval=945757 TSecr=945757 |
| 62 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 90 | Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_GROUP |
| 63 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 82 | Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_GROUP\_DESC |
| 64 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 90 | Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_METER |
| 65 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 82 | Type: OFPT\_MULTIPART\_REPLY, OFPMP\_GROUP |
| 66 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 82 | Type: OFPT\_MULTIPART\_REPLY, OFPMP\_GROUP\_DESC |
| 67 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 82 | Type: OFPT\_MULTIPART\_REPLY, OFPMP\_METER |
| 68 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=1917 Ack=8685 Win=204800 Len=0 TSval=945760 TSecr=945760 |
| 69 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 150 | Type: OFPT\_PACKET\_IN |
| 70 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 148 | Type: OFPT\_PACKET\_OUT |
| 71 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 150 | Type: OFPT\_PACKET\_IN |
| 72 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 148 | Type: OFPT\_PACKET\_OUT |
| 73 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 206 | Type: OFPT\_PACKET\_IN |
| 74 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 204 | Type: OFPT\_PACKET\_OUT |
| 75 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 206 | Type: OFPT\_PACKET\_IN |
| 76 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 170 | Type: OFPT\_FLOW\_MOD |
| 77 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REQUEST |
| 78 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49262 > 6633 [ACK] Seq=9133 Ack=2331 Win=47616 Len=0 TSval=946151 TSecr=946151 |
| 79 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REPLY |
| 80 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 204 | Type: OFPT\_PACKET\_OUT |
| 81 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 206 | Type: OFPT\_PACKET\_IN |
| 82 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 170 | Type: OFPT\_FLOW\_MOD |
| 83 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REQUEST |
| 84 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49262 > 6633 [ACK] Seq=9281 Ack=2581 Win=49664 Len=0 TSval=946154 TSecr=946154 |
| 85 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REPLY |
| 86 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 204 | Type: OFPT\_PACKET\_OUT |
| 87 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 170 | Type: OFPT\_FLOW\_MOD |
| 88 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REQUEST |
| 89 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49262 > 6633 [ACK] Seq=9289 Ack=2831 Win=51200 Len=0 TSval=946157 TSecr=946156 |
| 90 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 74 | Type: OFPT\_BARRIER\_REPLY |
| 91 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=2831 Ack=9297 Win=241152 Len=0 TSval=946167 TSecr=946157 |
| 92 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 187 | Type: OFPT\_PACKET\_OUT |
| 93 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 187 | Type: OFPT\_PACKET\_OUT |
| 94 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 187 | Type: OFPT\_PACKET\_OUT |
| 95 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 187 | Type: OFPT\_PACKET\_OUT |
| 96 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49262 > 6633 [ACK] Seq=9297 Ack=3315 Win=51200 Len=0 TSval=946285 TSecr=946285 |
| 97 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 146 | Type: OFPT\_PORT\_STATUS |
| 98 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=3315 Ack=9377 Win=241152 Len=0 TSval=946459 TSecr=946459 |
| 99 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 146 | Type: OFPT\_PORT\_STATUS |
| 100 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=3315 Ack=9457 Win=241152 Len=0 TSval=946467 TSecr=946467 |
| 101 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 146 | Type: OFPT\_PORT\_STATUS |
| 102 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=3315 Ack=9537 Win=241152 Len=0 TSval=946467 TSecr=946467 |
| 103 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 130 | Type: OFPT\_FLOW\_REMOVED |
| 104 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=3315 Ack=9601 Win=241152 Len=0 TSval=946485 TSecr=946485 |
| 105 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 130 | Type: OFPT\_FLOW\_REMOVED |
| 106 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=3315 Ack=9665 Win=241152 Len=0 TSval=946485 TSecr=946485 |
| 107 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 130 | Type: OFPT\_FLOW\_REMOVED |
| 108 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=3315 Ack=9729 Win=241152 Len=0 TSval=946485 TSecr=946485 |
| 109 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 130 | Type: OFPT\_FLOW\_REMOVED |
| 110 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=3315 Ack=9793 Win=241152 Len=0 TSval=946485 TSecr=946485 |
| 111 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 130 | Type: OFPT\_FLOW\_REMOVED |
| 112 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=3315 Ack=9857 Win=241152 Len=0 TSval=946485 TSecr=946485 |
| 113 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 146 | Type: OFPT\_FLOW\_REMOVED |
| 114 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=3315 Ack=9937 Win=241152 Len=0 TSval=946485 TSecr=946485 |
| 115 | 192.168.56.102 | 192.168.56.102 | OpenFlow | 146 | Type: OFPT\_FLOW\_REMOVED |
| 116 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [ACK] Seq=3315 Ack=10017 Win=241152 Len=0 TSval=946485 TSecr=946485 |
| 117 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49262 > 6633 [FIN, ACK] Seq=10017 Ack=3315 Win=51200 Len=0 TSval=946485 TSecr=946485 |
| 118 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 6633 > 49262 [FIN, ACK] Seq=3315 Ack=10018 Win=241152 Len=0 TSval=946485 TSecr=946485 |
| 119 | 192.168.56.102 | 192.168.56.102 | TCP | 66 | 49262 > 6633 [ACK] Seq=10018 Ack=3316 Win=51200 Len=0 TSval=946485 TSecr=946485 |

## Appendix C: NETCONF messages and config files

#### NETCONF configs:

ONOS NETCONF config file format:

{

"devices": {

"netconf:<ip>:<port>": {

"netconf": {

"ip": <ip>,

"port": <port>,

"username": <username>,

"password": <password>

},

"basic": {

"driver": <driver-name>

}

}

}

}

The config file used in experiment 3.3

{

"devices": {

"netconf:192.168.56.101:830": {

"netconf": {

"ip": "192.168.56.101",

"port": 830,

"username": "onos",

"password": "onos"

},

"basic": {

"driver": "ovs-netconf"

}

}

}

}

#### Captured NETCONF packets

NETCONF messages were encrypted. We cannot directly analyze the NETCONF messages using packets, so we have to log them from the server.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Source | Destination | Protocol | Length | Info |
| 1 | 192.168.56.101 | 192.168.56.1 | SSH | 190 | Server: Encrypted packet (len=124) |
| 2 | 192.168.56.1 | 192.168.56.101 | TCP | 66 | 51547 > 22 [ACK] Seq=1 Ack=125 Win=4092 Len=0 TSval=964114117 TSecr=2262577 |
| 3 | 192.168.56.1 | 192.168.56.255 | NBNS | 110 | Registration NB MACBOOKPRO-65AF<00> |
| 4 | 192.168.56.1 | 192.168.56.255 | NBNS | 110 | Registration NB MACBOOKPRO-65AF<00> |
| 5 | 192.168.56.1 | 192.168.56.255 | NBNS | 110 | Registration NB MACBOOKPRO-65AF<00> |
| 6 | 192.168.56.1 | 224.0.0.251 | MDNS | 436 | Standard query 0x0000 PTR \_airplay.\_tcp.local, "QM" question PTR \_raop.\_tcp.local, "QM" question PTR \_kmclipboards.\_tcp.local, "QM" question PTR \_kmclipboardss.\_tcp.local, "QM" question PTR \_uscan.\_tcp.local, "QM" question PTR \_uscans.\_tcp.local, "QM" question PTR \_ippusb.\_tcp.local, "QM" question PTR \_scanner.\_tcp.local, "QM" question PTR \_ipp.\_tcp.local, "QM" question PTR \_ipps.\_tcp.local, "QM" question PTR \_printer.\_tcp.local, "QM" question PTR \_pdl-datastream.\_tcp.local, "QM" question PTR \_ptp.\_tcp.local, "QM" question PTR \_airport.\_tcp.local, "QM" question PTR \_googlecast.\_tcp.local, "QM" question PTR \_privet.\_tcp.local, "QM" question PTR \_afpovertcp.\_tcp.local, "QM" question PTR \_smb.\_tcp.local, "QM" question PTR \_rfb.\_tcp.local, "QM" question PTR \_adisk.\_tcp.local, "QM" question PTR \_apple-mobdev.\_tcp.local, "QM" question PTR 946c081c.\_sub.\_apple-mobdev2.\_tcp.local, "QM" question PTR \_apple-pairable.\_tcp.local, "QM" question |
| 7 | 192.168.56.102 | 192.168.56.101 | TCP | 74 | 60832 > 830 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK\_PERM=1 TSval=2268400 TSecr=0 WS=512 |
| 8 | 192.168.56.101 | 192.168.56.102 | TCP | 74 | 830 > 60832 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK\_PERM=1 TSval=2269342 TSecr=2268400 WS=128 |
| 9 | 192.168.56.102 | 192.168.56.101 | TCP | 66 | 60832 > 830 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=2268400 TSecr=2269342 |
| 10 | 192.168.56.102 | 192.168.56.101 | TCP | 66 | 60832 > 830 [FIN, ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=2268401 TSecr=2269342 |
| 11 | 192.168.56.101 | 192.168.56.102 | TCP | 66 | 830 > 60832 [ACK] Seq=1 Ack=2 Win=29056 Len=0 TSval=2269344 TSecr=2268401 |
| 12 | 192.168.56.101 | 192.168.56.102 | TCP | 107 | 830 > 60832 [PSH, ACK] Seq=1 Ack=2 Win=29056 Len=41 TSval=2269344 TSecr=2268401 |
| 13 | 192.168.56.102 | 192.168.56.101 | TCP | 60 | 60832 > 830 [RST] Seq=2 Win=0 Len=0 |
| 14 | 192.168.56.102 | 192.168.56.101 | TCP | 74 | 60834 > 830 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK\_PERM=1 TSval=2268408 TSecr=0 WS=512 |
| 15 | 192.168.56.101 | 192.168.56.102 | TCP | 74 | 830 > 60834 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK\_PERM=1 TSval=2269350 TSecr=2268408 WS=128 |
| 16 | 192.168.56.102 | 192.168.56.101 | TCP | 66 | 60834 > 830 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=2268409 TSecr=2269350 |
| 17 | 192.168.56.102 | 192.168.56.101 | TCP | 66 | 60834 > 830 [FIN, ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=2268409 TSecr=2269350 |
| 18 | 192.168.56.101 | 192.168.56.102 | TCP | 66 | 830 > 60834 [ACK] Seq=1 Ack=2 Win=29056 Len=0 TSval=2269352 TSecr=2268409 |
| 19 | 192.168.56.101 | 192.168.56.102 | TCP | 107 | 830 > 60834 [PSH, ACK] Seq=1 Ack=2 Win=29056 Len=41 TSval=2269353 TSecr=2268409 |
| 20 | 192.168.56.101 | 192.168.56.102 | TCP | 66 | 830 > 60834 [FIN, ACK] Seq=42 Ack=2 Win=29056 Len=0 TSval=2269354 TSecr=2268409 |
| 21 | 192.168.56.102 | 192.168.56.101 | TCP | 60 | 60834 > 830 [RST] Seq=2 Win=0 Len=0 |
| 22 | 192.168.56.102 | 192.168.56.101 | TCP | 60 | 60834 > 830 [RST] Seq=2 Win=0 Len=0 |
| 23 | 192.168.56.102 | 192.168.56.101 | TCP | 74 | 60836 > 830 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK\_PERM=1 TSval=2268455 TSecr=0 WS=512 |
| 24 | 192.168.56.101 | 192.168.56.102 | TCP | 74 | 830 > 60836 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK\_PERM=1 TSval=2269397 TSecr=2268455 WS=128 |
| 25 | 192.168.56.102 | 192.168.56.101 | TCP | 66 | 60836 > 830 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=2268455 TSecr=2269397 |
| 26 | 192.168.56.101 | 192.168.56.102 | TCP | 107 | 830 > 60836 [PSH, ACK] Seq=1 Ack=1 Win=29056 Len=41 TSval=2269398 TSecr=2268455 |
| 27 | 192.168.56.102 | 192.168.56.101 | TCP | 66 | 60836 > 830 [ACK] Seq=1 Ack=42 Win=29696 Len=0 TSval=2268457 TSecr=2269398 |
| 28 | 192.168.56.102 | 192.168.56.101 | TCP | 88 | 60836 > 830 [PSH, ACK] Seq=1 Ack=42 Win=29696 Len=22 TSval=2268461 TSecr=2269398 |
| 29 | 192.168.56.101 | 192.168.56.102 | TCP | 66 | 830 > 60836 [ACK] Seq=42 Ack=23 Win=29056 Len=0 TSval=2269403 TSecr=2268461 |
| 30 | 192.168.56.101 | 192.168.56.102 | TCP | 1042 | 830 > 60836 [PSH, ACK] Seq=42 Ack=23 Win=29056 Len=976 TSval=2269403 TSecr=2268461 |
| 31 | 192.168.56.102 | 192.168.56.101 | TCP | 658 | 60836 > 830 [PSH, ACK] Seq=23 Ack=1018 Win=31232 Len=592 TSval=2268462 TSecr=2269403 |
| 32 | 192.168.56.101 | 192.168.56.102 | TCP | 66 | 830 > 60836 [ACK] Seq=1018 Ack=615 Win=30208 Len=0 TSval=2269417 TSecr=2268462 |
| 33 | 192.168.56.102 | 192.168.56.101 | TCP | 218 | 60836 > 830 [PSH, ACK] Seq=615 Ack=1018 Win=31232 Len=152 TSval=2268475 TSecr=2269417 |
| 34 | 192.168.56.101 | 192.168.56.102 | TCP | 66 | 830 > 60836 [ACK] Seq=1018 Ack=767 Win=31360 Len=0 TSval=2269417 TSecr=2268475 |
| 35 | 192.168.56.101 | 192.168.56.102 | TCP | 442 | 830 > 60836 [PSH, ACK] Seq=1018 Ack=767 Win=31360 Len=376 TSval=2269418 TSecr=2268475 |
| 36 | 192.168.56.102 | 192.168.56.101 | TCP | 82 | 60836 > 830 [PSH, ACK] Seq=767 Ack=1394 Win=33280 Len=16 TSval=2268481 TSecr=2269418 |
| 37 | 192.168.56.101 | 192.168.56.102 | TCP | 66 | 830 > 60836 [ACK] Seq=1394 Ack=783 Win=31360 Len=0 TSval=2269433 TSecr=2268481 |
| 38 | 192.168.56.102 | 192.168.56.101 | TCP | 134 | 60836 > 830 [PSH, ACK] Seq=783 Ack=1394 Win=33280 Len=68 TSval=2268496 TSecr=2269433 |
| 39 | 192.168.56.101 | 192.168.56.102 | TCP | 66 | 830 > 60836 [ACK] Seq=1394 Ack=851 Win=31360 Len=0 TSval=2269438 TSecr=2268496 |
| 40 | 192.168.56.101 | 192.168.56.102 | TCP | 118 | 830 > 60836 [PSH, ACK] Seq=1394 Ack=851 Win=31360 Len=52 TSval=2269438 TSecr=2268496 |
| 41 | 192.168.56.102 | 192.168.56.101 | TCP | 150 | 60836 > 830 [PSH, ACK] Seq=851 Ack=1394 Win=33280 Len=84 TSval=2268496 TSecr=2269438 |
| 42 | 192.168.56.102 | 192.168.56.101 | TCP | 66 | 60836 > 830 [ACK] Seq=935 Ack=1446 Win=33280 Len=0 TSval=2268506 TSecr=2269438 |
| 43 | 192.168.56.101 | 192.168.56.102 | TCP | 134 | 830 > 60836 [PSH, ACK] Seq=1446 Ack=935 Win=31360 Len=68 TSval=2269448 TSecr=2268496 |
| 44 | 192.168.56.102 | 192.168.56.101 | TCP | 66 | 60836 > 830 [ACK] Seq=935 Ack=1514 Win=33280 Len=0 TSval=2268506 TSecr=2269448 |
| 45 | 192.168.56.102 | 192.168.56.101 | TCP | 326 | 60836 > 830 [PSH, ACK] Seq=935 Ack=1514 Win=33280 Len=260 TSval=2268522 TSecr=2269448 |
| 46 | 192.168.56.101 | 192.168.56.102 | TCP | 134 | 830 > 60836 [PSH, ACK] Seq=1514 Ack=1195 Win=32512 Len=68 TSval=2269464 TSecr=2268522 |
| 47 | 192.168.56.102 | 192.168.56.101 | TCP | 66 | 60836 > 830 [ACK] Seq=1195 Ack=1582 Win=33280 Len=0 TSval=2268522 TSecr=2269464 |
| 48 | 192.168.56.102 | 192.168.56.101 | TCP | 166 | 60836 > 830 [PSH, ACK] Seq=1195 Ack=1582 Win=33280 Len=100 TSval=2268523 TSecr=2269464 |
| 49 | 192.168.56.101 | 192.168.56.102 | TCP | 102 | 830 > 60836 [PSH, ACK] Seq=1582 Ack=1295 Win=32512 Len=36 TSval=2269466 TSecr=2268523 |
| 50 | 192.168.56.102 | 192.168.56.101 | TCP | 134 | 60836 > 830 [PSH, ACK] Seq=1295 Ack=1618 Win=33280 Len=68 TSval=2268530 TSecr=2269466 |
| 51 | 192.168.56.101 | 192.168.56.102 | TCP | 66 | 830 > 60836 [ACK] Seq=1618 Ack=1363 Win=32512 Len=0 TSval=2269482 TSecr=2268530 |
| 52 | 192.168.56.101 | 192.168.56.102 | TCP | 1014 | 830 > 60836 [PSH, ACK] Seq=1618 Ack=1363 Win=32512 Len=948 TSval=2269487 TSecr=2268530 |
| 53 | 192.168.56.102 | 192.168.56.101 | TCP | 66 | 60836 > 830 [ACK] Seq=1363 Ack=2566 Win=35328 Len=0 TSval=2268556 TSecr=2269487 |
| 54 | 192.168.56.101 | 192.168.56.102 | TCP | 118 | 830 > 60836 [PSH, ACK] Seq=2566 Ack=1363 Win=32512 Len=52 TSval=2269498 TSecr=2268556 |
| 55 | 192.168.56.102 | 192.168.56.101 | TCP | 66 | 60836 > 830 [ACK] Seq=1363 Ack=2618 Win=35328 Len=0 TSval=2268556 TSecr=2269498 |
| 56 | 192.168.56.102 | 192.168.56.101 | TCP | 150 | 60836 > 830 [PSH, ACK] Seq=1363 Ack=2618 Win=35328 Len=84 TSval=2268556 TSecr=2269498 |
| 57 | 192.168.56.101 | 192.168.56.102 | TCP | 66 | 830 > 60836 [ACK] Seq=2618 Ack=1447 Win=32512 Len=0 TSval=2269498 TSecr=2268556 |
| 58 | 192.168.56.101 | 192.168.56.102 | TCP | 154 | 830 > 60836 [PSH, ACK] Seq=2618 Ack=1447 Win=32512 Len=88 TSval=2269498 TSecr=2268556 |
| 59 | 192.168.56.102 | 192.168.56.101 | TCP | 390 | 60836 > 830 [PSH, ACK] Seq=1447 Ack=2706 Win=35328 Len=324 TSval=2268558 TSecr=2269498 |
| 60 | 192.168.56.101 | 192.168.56.1 | SSH | 142 | Server: Encrypted packet (len=76) |
| 61 | 192.168.56.1 | 192.168.56.101 | TCP | 66 | 64266 > 22 [ACK] Seq=1 Ack=77 Win=4093 Len=0 TSval=964141763 TSecr=2269501 |
| 62 | 192.168.56.101 | 192.168.56.1 | SSH | 150 | Server: Encrypted packet (len=84) |
| 63 | 192.168.56.1 | 192.168.56.101 | TCP | 66 | 64266 > 22 [ACK] Seq=1 Ack=161 Win=4093 Len=0 TSval=964141763 TSecr=2269501 |
| 64 | 192.168.56.101 | 192.168.56.102 | TCP | 2918 | 830 > 60836 [PSH, ACK] Seq=2706 Ack=1771 Win=33792 Len=2852 TSval=2269502 TSecr=2268558 |
| 65 | 192.168.56.102 | 192.168.56.101 | TCP | 66 | 60836 > 830 [ACK] Seq=1771 Ack=5558 Win=40960 Len=0 TSval=2268560 TSecr=2269502 |
| 66 | 192.168.56.101 | 192.168.56.1 | SSH | 142 | Server: Encrypted packet (len=76) |
| 67 | 192.168.56.1 | 192.168.56.101 | TCP | 66 | 64266 > 22 [ACK] Seq=1 Ack=237 Win=4093 Len=0 TSval=964141765 TSecr=2269502 |
| 68 | 192.168.56.101 | 192.168.56.1 | SSH | 174 | Server: Encrypted packet (len=108) |
| 69 | 192.168.56.1 | 192.168.56.101 | TCP | 66 | 64266 > 22 [ACK] Seq=1 Ack=345 Win=4092 Len=0 TSval=964141765 TSecr=2269502 |
| 70 | 192.168.56.101 | 192.168.56.1 | SSH | 166 | Server: Encrypted packet (len=100) |
| 71 | 192.168.56.1 | 192.168.56.101 | TCP | 66 | 64266 > 22 [ACK] Seq=1 Ack=445 Win=4092 Len=0 TSval=964141766 TSecr=2269502 |
| 72 | 192.168.56.1 | 192.168.56.255 | NBNS | 92 | Name query NB <01><02>\_\_MSBROWSE\_\_<02><01> |
| 73 | 192.168.56.1 | 192.168.56.255 | NBNS | 92 | Name query NB <01><02>\_\_MSBROWSE\_\_<02><01> |
| 74 | 192.168.56.1 | 192.168.56.255 | NBNS | 92 | Name query NB <01><02>\_\_MSBROWSE\_\_<02><01> |

Logged NETCONF messages:

First SDN controller sends hello message:

<?xml version="1.0" encoding="UTF-8"?>

<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<capabilities>

<capability>urn:ietf:params:netconf:base:1.0</capability>

<capability>urn:ietf:params:netconf:base:1.1</capability>

</capabilities>

</hello>

]]>]]>

Device replies:

<?xml version="1.0" encoding="UTF-8"?>

<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<capabilities>

<capability>urn:ietf:params:netconf:base:1.0</capability>

<capability>urn:ietf:params:netconf:base:1.1</capability>

<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>

<capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>

<capability>urn:ietf:params:netconf:capability:startup:1.0</capability>

<capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>

<capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>

<capability>urn:ietf:params:netconf:capability:notification:1.0</capability>

<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>

<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>

<capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=explicit&amp;also-supported=report-all,report-all-tagged,trim,explicit</capability>

<capability>urn:ietf:params:netconf:capability:url:1.0?scheme=scp,file</capability>

<capability>urn:onf:config:yang?module=of-config&amp;revision=2015-02-11</capability>

<capability>urn:ietf:params:xml:ns:yang:ietf-netconf-server?module=ietf-netconf-server&amp;revision=2014-01-24&amp;features=ssh,inbound-ssh</capability>

<capability>urn:ietf:params:xml:ns:yang:ietf-x509-cert-to-name?module=ietf-x509-cert-to-name&amp;revision=2013-03-26</capability>

<capability>urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults?module=ietf-netconf-with-defaults&amp;revision=2010-06-09</capability>

<capability>urn:cesnet:params:xml:ns:libnetconf:notifications?module=libnetconf-notifications&amp;revision=2016-07-21</capability>

<capability>urn:ietf:params:xml:ns:netconf:notification:1.0?module=notifications&amp;revision=2008-07-14</capability>

<capability>urn:ietf:params:xml:ns:netmod:notification?module=nc-notifications&amp;revision=2008-07-14</capability>

<capability>urn:ietf:params:xml:ns:yang:ietf-netconf-notifications?module=ietf-netconf-notifications&amp;revision=2012-02-06</capability>

<capability>urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring?module=ietf-netconf-monitoring&amp;revision=2010-10-04</capability>

<capability>urn:ietf:params:xml:ns:netconf:base:1.0?module=ietf-netconf&amp;revision=2011-06-01&amp;features=writable-running,candidate,rollback-on-error,validate,startup,url</capability>

<capability>urn:ietf:params:xml:ns:yang:ietf-yang-types?module=ietf-yang-types&amp;revision=2013-07-15</capability>

<capability>urn:ietf:params:xml:ns:yang:ietf-inet-types?module=ietf-inet-types&amp;revision=2013-07-15</capability>

</capabilities>

<session-id>2</session-id>

</hello>

from request <?xml version="1.0" encoding="UTF-8"?>

<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<capabilities>

<capability>urn:ietf:params:netconf:base:1.0</capability>

<capability>urn:ietf:params:netconf:base:1.1</capability>

</capabilities>

</hello>

After the exchange of hello message, the device is connected to onos controller. As you can see they now have a session id and the onos controller, which serves as a netconf client, knows the capabilities of the netconf device.

Now we try to get configuration of the device:

Type this in ONOS CLI:

###### netconf-get-config netconf:192.168.56.101:830 running

It’s in a format of cli-command device-ID cfgtype

Configuration have three types: running, candidate and startup

We can see the ONOS send message to get the running configuration:

<?xml version="1.0" encoding="UTF-8"?><rpc message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<get-config>

<source>

<running/></source></get-config>

</rpc>

the device returns:

<?xml version="1.0" encoding="UTF-8"?>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">

<data>

<capable-switch xmlns="urn:onf:config:yang">

<id>openvswitch</id>

<resources>

<port>

<name>ofc-bridge</name>

<requested-number>666</requested-number>

<configuration>

<admin-state>down</admin-state>

<no-receive>false</no-receive>

<no-forward>false</no-forward>

<no-packet-in>false</no-packet-in>

</configuration>

</port>

</resources>

<logical-switches>

<switch>

<id>ofc-bridge</id>

<datapath-id>00:01:02:03:04:05:06:07</datapath-id>

<lost-connection-behavior>failSecureMode</lost-connection-behavior>

<resources>

<port>ofc-bridge</port>

</resources>

</switch>

</logical-switches>

</capable-switch>

</data>

</rpc-reply>

Change the cfgtype here we use:

###### netconf-get-config netconf:192.168.56.101:830 candidate

Now ONOS controller send this message to get the candidate configuration:

<?xml version="1.0" encoding="UTF-8"?><rpc message-id="2" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<get-config>

<source>

<candidate/></source></get-config>

</rpc>

and get a reply:

<?xml version="1.0" encoding="UTF-8"?>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">

<data/>

</rpc-reply>

Finally we try to get the startup config by

###### netconf-get-config netconf:192.168.56.101:830 startup

The SDN controller sends:

<?xml version="1.0" encoding="UTF-8"?>

<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">

<get-config>

<source>

<startup/>

</source>

</get-config>

</rpc>

The device replies:

<?xml version="1.0" encoding="UTF-8"?>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">

<data>

<capable-switch xmlns="urn:onf:config:yang">

<id>openvswitch</id>

<resources>

<port>

<name>ofc-bridge</name>

<requested-number>666</requested-number>

</port>

</resources>

<logical-switches>

<switch>

<id>ofc-bridge</id>

<datapath-id>00:01:02:03:04:05:06:07</datapath-id>

<resources>

<port>ofc-bridge</port>

</resources>

</switch>

</logical-switches>

</capable-switch>

</data>

</rpc-reply>

## Appendix D: Files used in TestON

#### Test Files

Note that you need to change the IP, username and password in those files to get TestON working for you.

#### .params

<PARAMS>

   <!--

       CASE0: pull onos code - this case should be skipped on Jenkins-driven prod test

       CASE1: setup and clean test env

       CASE2: get onos warnings, errors from log

       CASE10: start a 1-node ONOS

       CASE12: Sample case of using onos cli

   -->

   <testcases>0,1,10,12</testcases>

   <GIT>

       <pull>False</pull>

       <branch>master</branch>

   </GIT>

   <CASE0>

   </CASE0>

   <CASE1>

       <NodeList>OC1</NodeList>

       <SleepTimers>

           <onosStartup>60</onosStartup>

           <onosCfg>5</onosCfg>

           <mnStartup>15</mnStartup>

           <mnCfg>10</mnCfg>

       </SleepTimers>

   </CASE1>

   <CASE10>

       <numNodes>1</numNodes>

       <Apps>

          org.onosproject.openflow,org.onosproject.fwd

       </Apps>

       <ONOS\_Configuration>

           <org.onosproject.net.intent.impl.compiler.IntentConfigurableRegistrator>

               <useFlowObjectives>true</useFlowObjectives>

               <defaultFlowObjectiveCompiler>org.onosproject.net.intent.impl.compiler.LinkCollectionIntentObjectiveCompiler</defaultFlowObjectiveCompiler>

           </org.onosproject.net.intent.impl.compiler.IntentConfigurableRegistrator>

       </ONOS\_Configuration>

   </CASE10>

   <CASE11>

       <topo> mn --topo tree,3,3 </topo>

   </CASE11>

   <CASE12>

   </CASE12>

   <CASE22>

   </CASE22>

   <CASE32>

   </CASE32>

</PARAMS>

#### .py

"""

Copyright 2015 Open Networking Foundation ( ONF )

Please refer questions to either the onos test mailing list at <onos-test@onosproject.org>,

the System Testing Plans and Results wiki page at <https://wiki.onosproject.org/x/voMg>,

or the System Testing Guide page at <https://wiki.onosproject.org/x/WYQg>

TestON is free software: you can redistribute it and/or modify

it under the terms of the GNU General Public License as published by

the Free Software Foundation, either version 2 of the License, or

( at your option ) any later version.

TestON is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

GNU General Public License for more details.

You should have received a copy of the GNU General Public License

along with TestON. If not, see <http://www.gnu.org/licenses/>.

"""

# This is a sample template that starts up ONOS cluster, this template

# can be use as a base script for ONOS System Testing.

class SAMPstartTemplate\_1node:

def \_\_init\_\_( self ):

self.default = ''

def CASE0( self, main ):

"""

Pull specific ONOS branch, then Build ONOS on ONOS Bench.

This step is usually skipped. Because in a Jenkins driven automated

test env. We want Jenkins jobs to pull&build for flexibility to handle

different versions of ONOS.

"""

from tests.dependencies.ONOSSetup import ONOSSetup

main.testSetUp = ONOSSetup()

main.testSetUp.gitPulling()

from tests.dependencies.Network import Network

main.Network = Network()

def CASE1( self, main ):

"""

Set up global test variables;

Uninstall all running cells in test env defined in .topo file

"""

main.testSetUp.envSetupDescription()

stepResult = main.FALSE

try:

main.onosStartupSleep = float( main.params[ 'CASE1' ][ 'SleepTimers' ][ 'onosStartup' ] )

main.onosCfgSleep = float( main.params[ 'CASE1' ][ 'SleepTimers' ][ 'onosCfg' ] )

main.mnStartupSleep = float( main.params[ 'CASE1' ][ 'SleepTimers' ][ 'mnStartup' ] )

main.mnCfgSleep = float( main.params[ 'CASE1' ][ 'SleepTimers' ][ 'mnCfg' ] )

stepResult = main.testSetUp.envSetup( includeGitPull=False )

except Exception as e:

main.testSetUp.envSetupException( e )

main.testSetUp.evnSetupConclusion( stepResult )

def CASE2( self, main ):

"""

Report errors/warnings/exceptions

"""

main.log.info( "Error report: \n" )

main.ONOSbench.logReport( main.Cluster.active( 0 ).ipAddress,

[ "INFO",

"FOLLOWER",

"WARN",

"flow",

"ERROR",

"Except" ],

"s" )

def CASE10( self, main ):

"""

Start ONOS cluster ( 1 node in this example ) in three steps:

1 ) start a basic cluster with drivers app via ONOSDriver;

2 ) activate apps via ONOSCliDriver;

3 ) configure onos via ONOSCliDriver;

"""

import time

main.case( "Start up " + str( main.Cluster.numCtrls ) + "-node onos cluster." )

main.step( "Start ONOS cluster with basic (drivers) app." )

stepResult = main.testSetUp.ONOSSetUp( main.Cluster )

utilities.assert\_equals( expect=main.TRUE,

actual=stepResult,

onpass="Successfully started basic ONOS cluster ",

onfail="Failed to start basic ONOS Cluster " )

main.step( "Activate onos apps." )

main.apps = main.params[ 'CASE10' ].get( 'Apps' )

if main.apps:

main.log.info( "Apps to activate: " + main.apps )

activateResult = main.TRUE

for a in main.apps.split( "," ):

activateResult = activateResult & main.Cluster.active( 0 ).CLI.activateApp( a )

# TODO: check this worked

time.sleep( main.onosCfgSleep ) # wait for apps to activate

else:

main.log.warn( "No configurations were specified to be changed after startup" )

utilities.assert\_equals( expect=main.TRUE,

actual=activateResult,

onpass="Successfully set config",

onfail="Failed to set config" )

main.step( "Set ONOS configurations" )

config = main.params[ 'CASE10' ].get( 'ONOS\_Configuration' )

if config:

main.log.debug( config )

checkResult = main.TRUE

for component in config:

for setting in config[ component ]:

value = config[ component ][ setting ]

check = main.Cluster.active( 0 ).CLI.setCfg( component, setting, value )

main.log.info( "Value was changed? {}".format( main.TRUE == check ) )

checkResult = check and checkResult

utilities.assert\_equals( expect=main.TRUE,

actual=checkResult,

onpass="Successfully set config",

onfail="Failed to set config" )

else:

main.log.warn( "No configurations were specified to be changed after startup" )

def CASE11( self, main ):

"""

Start mininet and assign controllers

"""

import time

topology = main.params[ 'CASE11' ][ 'topo' ]

main.log.report( "Start Mininet topology" )

main.case( "Start Mininet topology" )

main.step( "Starting Mininet Topology" )

topoResult = main.Network.startNet( mnCmd=topology )

stepResult = topoResult

utilities.assert\_equals( expect=main.TRUE,

actual=stepResult,

onpass="Successfully loaded topology",

onfail="Failed to load topology" )

# Exit if topology did not load properly

if not topoResult:

main.cleanAndExit()

main.step( "Assign switches to controllers." )

assignResult = main.TRUE

for i in range( 1, 8 ):

assignResult = assignResult & \

main.Network.assignSwController( sw="s" + str( i ),

ip=main.Cluster.getIps(),

port='6653' )

time.sleep( main.mnCfgSleep )

utilities.assert\_equals( expect=main.TRUE,

actual=stepResult,

onpass="Successfully assign switches to controllers",

onfail="Failed to assign switches to controllers" )

def CASE12( self, main ):

"""

Tests using through ONOS CLI handles

"""

main.case( "Test some onos commands through CLI. " )

main.log.debug( main.Cluster.active( 0 ).CLI.sendline( "summary" ) )

main.log.debug( main.Cluster.active( 0 ).CLI.sendline( "devices" ) )

def CASE22( self, main ):

"""

Tests using ONOS REST API handles

"""

main.case( " Sample tests using ONOS REST API handles. " )

main.log.debug( main.Cluster.active( 0 ).REST.send( "/devices" ) )

main.log.debug( main.Cluster.active( 0 ).REST.apps() )

def CASE32( self, main ):

"""

Configure fwd app from .param json string with parameter configured.

Check if configuration successful

Run pingall to check connectivity

Check ONOS log for warning/error/exceptions

"""

main.case( "Configure onos-app-fwd and check if configuration successful. " )

main.step( "Install reactive forwarding app." )

installResults = main.Cluster.active( 0 ).CLI.activateApp( "org.onosproject.fwd" )

utilities.assert\_equals( expect=main.TRUE,

actual=installResults,

onpass="Configure fwd successful",

onfail="Configure fwd failed" )

main.step( "Run pingall to check connectivity. " )

pingResult = main.FALSE

passMsg = "Reactive Pingall test passed"

pingResult = main.Network.pingall()

if not pingResult:

main.log.warn( "First pingall failed. Trying again..." )

pingResult = main.Network.pingall()

passMsg += "on the second try"

utilities.assert\_equals( expect=main.TRUE,

actual=pingResult,

onpass=passMsg,

onfail="Reactive Pingall failed, " + "one or more ping pairs failed." )

#### .topo

<TOPOLOGY>

   <COMPONENT>

​

       <ONOScell>

           <host>192.168.56.102</host>

           <user>sdn</user>

           <password></password>

           <type>OnosClusterDriver</type>

           <connect\_order>1</connect\_order>

           <COMPONENTS>

               <cluster\_name>ONOS</cluster\_name>

               <diff\_clihost></diff\_clihost>

               <karaf\_username></karaf\_username>

               <karaf\_password></karaf\_password>

               <web\_user>onos</web\_user>

               <web\_pass>rocks</web\_pass>

               <rest\_port></rest\_port>

               <prompt>onos</prompt>

               <onos\_home>~/onos</onos\_home>

               <nodes>1</nodes>

           </COMPONENTS>

       </ONOScell>

​

       <Mininet1>

           <host>192.168.56.101</host>

           <user>mininet</user>

           <password>mininet</password>

           <type>MininetCliDriver</type>

           <connect\_order>2</connect\_order>

           <COMPONENTS>

               <home>~/mininet</home>

           </COMPONENTS>

       </Mininet1>

​

   </COMPONENT>

</TOPOLOGY>

Test result log file:

+----------------+

------------------------------ { Script And Files } ------------------------------

+----------------+

Script Log File : /home/sdn/OnosSystemTest/TestON/logs/NETCONF\_Test\_12\_May\_2018\_15\_13\_53/NETCONF\_Test\_12\_May\_2018\_15\_13\_53.log

Report Log File : /home/sdn/OnosSystemTest/TestON/logs/NETCONF\_Test\_12\_May\_2018\_15\_13\_53/NETCONF\_Test\_12\_May\_2018\_15\_13\_53.rpt

Mininet1 Session Log : /home/sdn/OnosSystemTest/TestON/logs/NETCONF\_Test\_12\_May\_2018\_15\_13\_53/Mininet1.session

ONOScell Session Log : /home/sdn/OnosSystemTest/TestON/logs/NETCONF\_Test\_12\_May\_2018\_15\_13\_53/ONOScell.session

Test Script :/home/sdn/OnosSystemTest/TestON/Tests/NETCONF\_Test.py

Test Params : /home/sdn/OnosSystemTest/TestON/Tests/NETCONF\_Test.params

Topology : /home/sdn/OnosSystemTest/TestON/Tests/NETCONF\_Test.topo

+------------------+

--------------------------- { Script Exec Params } ---------------------------

+------------------+

'CASE11':

'topo': 'mn --topo tree

3

3'

'CASE10':

'ONOS\_Configuration':

'org.onosproject.net.intent.impl.compiler.IntentConfigurableRegistrator':

'useFlowObjectives': 'true'

'defaultFlowObjectiveCompiler': 'org.onosproject.net.intent.impl.compiler.LinkCollectionIntentObjectiveCompiler'

'Apps': 'org.onosproject.openflow'

'numNodes': '1'

'testcases': '0

1

10

12'

'GIT':

'pull': 'False'

'branch': 'master'

'CASE0': ''

'CASE32': ''

'CASE1':

'SleepTimers':

'mnCfg': '10'

'onosCfg': '60'

'mnStartup': '15'

'onosStartup': '60'

'NodeList': 'OC1'

'GRAPH':

'nodeCluster': 'VM'

'builds': '20'

+---------------+

----------------------------- { Components Used } -----------------------------

+---------------+

Mininet1

ONOScell

+--------+

---------------------------- { Topology } ----------------------------

+--------+

'Mininet1':

'connect\_order': '2'

'host': '192.168.56.101'

'user': 'mininet'

'COMPONENTS':

'home': '~/mininet'

'prompt': None

'password': 'mininet'

'type': 'MininetCliDriver'

'ONOScell':

'connect\_order': '1'

'host': '192.168.56.102'

'user': 'sdn'

'COMPONENTS':

'web\_user': 'onos'

'diff\_clihost': None

'web\_pass': 'rocks'

'prompt': 'onos'

'cluster\_name': 'ONOS'

'onos\_home': '~/onos'

'nodes': '1'

'karaf\_password': None

'karaf\_username': None

'rest\_port': None

'password': None

'type': 'OnosClusterDriver'

------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CASE INIT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2018-05-12 15:13:53,665 - NETCONF\_Test - INFO - Creating component Handle: ONOScell

2018-05-12 15:13:53,684 - NETCONF\_Test - INFO - ONOScell: Found ['192.168.56.102'] ONOS IPs

2018-05-12 15:13:53,684 - NETCONF\_Test - INFO - ONOScell: Trying to connect to 192.168.56.102

2018-05-12 15:13:54,302 - NETCONF\_Test - INFO - Password not required logged in

2018-05-12 15:13:54,575 - NETCONF\_Test - INFO - Creating component Handle: ONOScli1

2018-05-12 15:13:54,592 - NETCONF\_Test - INFO - ONOScli1: Trying to connect to 192.168.56.102

2018-05-12 15:13:55,207 - NETCONF\_Test - INFO - Password not required logged in

2018-05-12 15:13:55,360 - NETCONF\_Test - INFO - Creating component Handle: ONOSrest1

2018-05-12 15:13:55,471 - NETCONF\_Test - INFO - ONOSrest1: ip set to 192.168.56.102

2018-05-12 15:13:55,472 - NETCONF\_Test - INFO - Creating component Handle: ONOSbench1

2018-05-12 15:13:55,481 - NETCONF\_Test - INFO - ONOSbench1: Found ['192.168.56.102'] ONOS IPs

2018-05-12 15:13:55,482 - NETCONF\_Test - INFO - ONOSbench1: Trying to connect to 192.168.56.102

2018-05-12 15:13:56,096 - NETCONF\_Test - INFO - Password not required logged in

2018-05-12 15:13:56,248 - NETCONF\_Test - INFO - Creating component Handle: ONOSserver1

2018-05-12 15:13:56,256 - NETCONF\_Test - INFO - ONOSserver1: Found ['192.168.56.102'] ONOS IPs

2018-05-12 15:13:56,257 - NETCONF\_Test - INFO - ONOSserver1: Trying to connect to 192.168.56.102

2018-05-12 15:13:56,882 - NETCONF\_Test - INFO - Password not required logged in

2018-05-12 15:13:57,034 - NETCONF\_Test - INFO - Creating component Handle: Mininet1

2018-05-12 15:13:57,046 - NETCONF\_Test - INFO - Mininet1: Trying to connect to 192.168.56.101

2018-05-12 15:13:57,235 - NETCONF\_Test - INFO - ssh connection asked for password, gave password

2018-05-12 15:13:57,910 - NETCONF\_Test - INFO - Connection successful to the host mininet@192.168.56.101

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Result summary for Testcase0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[2018-05-12 15:13:57.919528] [NETCONF\_Test] [CASE] Pull onos branch and build onos on Teststation.

2018-05-12 15:13:57,920 - NETCONF\_Test - INFO - Skipped git checkout and pull as they are disabled in params file

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Result: Pass

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Result summary for Testcase1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[2018-05-12 15:13:57.923743] [NETCONF\_Test] [CASE] Constructing test variables and building ONOS package

[2018-05-12 15:13:57.924707] [NETCONF\_Test] [STEP] 1.1: Constructing test variables

2018-05-12 15:13:57,927 - NETCONF\_Test - INFO - Verifying the Expected is equal to the actual or not using assert\_equal

2018-05-12 15:13:57,927 - NETCONF\_Test - INFO - Successfully construct test variables

2018-05-12 15:13:57,927 - NETCONF\_Test - INFO - Assertion Passed

2018-05-12 15:13:57,928 - NETCONF\_Test - INFO - self.home =

2018-05-12 15:13:57,928 - NETCONF\_Test - INFO - ~/onos

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Result: Pass

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Result summary for Testcase10

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[2018-05-12 15:13:58.243424] [NETCONF\_Test] [CASE] Start up 1-node onos cluster.

[2018-05-12 15:13:58.244685] [NETCONF\_Test] [STEP] 10.1: Start ONOS cluster with basic (drivers) app.

[2018-05-12 15:13:58.245739] [NETCONF\_Test] [CASE] Starting up 1 node(s) ONOS cluster

2018-05-12 15:13:58,246 - NETCONF\_Test - INFO - NODE COUNT = 1

[2018-05-12 15:13:58.844104] [NETCONF\_Test] [STEP] 10.2: Apply cell to environment

2018-05-12 15:13:58,949 - NETCONF\_Test - INFO - ONOSbench1: Successfully set cell:

[01;31m[KONOS\_CELL[m[K=temp

[01;31m[KOCI[m[K=192.168.56.102

OC1=192.168.56.102

[01;31m[KOCN[m[K=192.168.56.101

ONOS\_APPS=drivers

ONOS\_GROUP=sdn

ONOS\_MN\_PY=/home/sdn/

2018-05-12 15:13:59,051 - NETCONF\_Test - INFO - Verify cell returned: .py

ONOS\_NIC=192.168.3.\*

ONOS\_SCENARIOS=/home/sdn/onos

2018-05-12 15:13:59,051 - NETCONF\_Test - INFO - Verifying the Expected is equal to the actual or not using assert\_equal

2018-05-12 15:13:59,052 - NETCONF\_Test - INFO - Successfully applied cell to environment

2018-05-12 15:13:59,052 - NETCONF\_Test - INFO - Assertion Passed

[2018-05-12 15:13:59.053174] [NETCONF\_Test] [STEP] 10.3: Uninstalling ONOS package

2018-05-12 15:13:59,155 - NETCONF\_Test - INFO - ONOS 192.168.56.102 was uninstalled

2018-05-12 15:13:59,156 - NETCONF\_Test - INFO - Verifying the Expected is equal to the actual or not using assert\_equal

2018-05-12 15:13:59,156 - NETCONF\_Test - INFO - Successfully uninstalled ONOS package

2018-05-12 15:13:59,156 - NETCONF\_Test - INFO - Assertion Passed

[2018-05-12 15:13:59.156933] [NETCONF\_Test] [STEP] 10.4: Creating ONOS package

2018-05-12 15:13:59,747 - NETCONF\_Test - INFO - Verifying the Expected is equal to the actual or not using assert\_equal

2018-05-12 15:13:59,747 - NETCONF\_Test - INFO - Successfully created ONOS package

2018-05-12 15:13:59,747 - NETCONF\_Test - INFO - Assertion Passed

[2018-05-12 15:13:59.747879] [NETCONF\_Test] [STEP] 10.5: Set up ONOS secure SSH

2018-05-12 15:13:59,850 - NETCONF\_Test - INFO - Secure SSH performed on 192.168.56.102

2018-05-12 15:13:59,850 - NETCONF\_Test - INFO - Verifying the Expected is equal to the actual or not using assert\_equal

2018-05-12 15:13:59,851 - NETCONF\_Test - INFO - Test step PASS

2018-05-12 15:13:59,851 - NETCONF\_Test - INFO - Assertion Passed

[2018-05-12 15:13:59.851845] [NETCONF\_Test] [STEP] 10.6: Checking ONOS service

2018-05-12 15:14:00,379 - NETCONF\_Test - INFO - ONOSbench1: 192.168.56.102 is up

2018-05-12 15:14:00,380 - NETCONF\_Test - INFO - ONOS1 is up and ready

2018-05-12 15:14:00,380 - NETCONF\_Test - INFO - Verifying the Expected is equal to the actual or not using assert\_equal

2018-05-12 15:14:00,381 - NETCONF\_Test - INFO - ONOS service is ready on all nodes

2018-05-12 15:14:00,381 - NETCONF\_Test - INFO - Assertion Passed

[2018-05-12 15:14:00.381639] [NETCONF\_Test] [STEP] 10.7: Starting ONOS CLI sessions

2018-05-12 15:14:00,694 - NETCONF\_Test - INFO - 192.168.56.102 CLI Started successfully

2018-05-12 15:14:00,695 - NETCONF\_Test - INFO - Successful CLI startup

2018-05-12 15:14:00,695 - NETCONF\_Test - INFO - Verifying the Expected is equal to the actual or not using assert\_equal

2018-05-12 15:14:00,695 - NETCONF\_Test - INFO - Successfully start ONOS cli

2018-05-12 15:14:00,695 - NETCONF\_Test - INFO - Assertion Passed

2018-05-12 15:14:00,696 - NETCONF\_Test - INFO - Verifying the Expected is equal to the actual or not using assert\_equal

2018-05-12 15:14:00,696 - NETCONF\_Test - INFO - Successfully started basic ONOS cluster

2018-05-12 15:14:00,696 - NETCONF\_Test - INFO - Assertion Passed

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Result: Pass

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Result summary for Testcase12

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[2018-05-12 15:14:00.700996] [NETCONF\_Test] [CASE] Test some onos commands through CLI.

2018-05-12 15:14:00,919 - NETCONF\_Test - INFO - Command 'netconf-get-config netconf:192.168.56.101:830' sent to ONOScli1.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Result: Pass

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Test Execution Summary

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Test Start : 12 May 2018 15:13:53

Test End : 12 May 2018 15:14:00

Execution Time : 0:00:07.259152

Total tests planned : 4

Total tests RUN : 4

Total Pass : 4

Total Fail : 0

Total No Result : 0

Success Percentage : 100%

Execution Result : 100%

Case Executed : [0, 1, 10, 12]

Case Not Executed : []

## Appendix E Files used in LINC-switch

sys.config for make:

[

{linc,

[

%% Following setting enables or disables OF-Config subsystem which consists

%% of three applications: ssh, enetconf and of\_config.

%% Allowed values: 'enabled' | 'disabled'

{of\_config, enabled},

%% Following setting enables or disables spawning a new process for each

%% packet to be routed. If it's set to false a new process will be spawned,

%% if it's set to true (or not defined) it will stay in the parent process

{sync\_routing, true},

%% Configure ports available to the switch when using the

%% userspace backend according to your system setup.

%% \* Under Linux all TAP interfaces must be set up beforehand

%% as persistent.

%% \* Under MacOSX TAP interfaces are created during node

%% startup (which requires setting an ip parameter).

%% Remember to set 'rate' option when you plan to assign queues

%% to the given port.

{capable\_switch\_ports,

[

%% Examples:

%% - regular hardware interface with queues disabled

%% {port, 1, [{interface, "eth0"}]},

%% - regular hardware interface and port rate when queues enabled

%% {port, 2, [{interface, "eth0"}, {port\_rate, {100, kbps}}]},

%% - hardware interface with explicit type and queues disabled

%% {port, 3, [{interface, "net0"}, {type, eth}]},

%% - regular tap interface with queues disabled

%% {port, 4, [{interface, "tap0"}]},

%% - tap interface under MacOSX with dynamically assigned IP

%% {port, 5, [{interface, "tap1"}, {ip, "10.0.0.1"}]},

%% - tap interface with explicit type and queues disabled

%% {port, 6, [{interface, "net1"}, {type, tap}]}

%% - emulated optical interface without queues (the interface name

%% has no meaning

%% {port, 7, [{interface, "opt1"}, {type, optical}]}

]},

{capable\_switch\_queues,

[

%% Examples:

%% {queue, 1, [{min\_rate, 100}, {max\_rate, 100}]},

%% {queue, 2, [{min\_rate, 100}, {max\_rate, 100}]}

]},

%% Configuration of emulated optical links that connect emulated optical

%% ports. Each element of the list has format:

%% {{LOGICAL\_SWITCH\_A, PORT\_A}, {LOGICAL\_SWITCH\_B, PORT\_B}}

%% {optical\_links, [{{1,2}, {2,1}}, {{2,2},{3,1}}]},

%% Configuration of the logical switches.

{logical\_switches,

[

{switch, 0,

[

%% Configuration of switch backend implementation used by ofs\_logic

%% process.

%% By default an Erlang userspace 1.3.2 implementation is selected.

%% Allowed values: 'linc\_us3' | 'linc\_us4' | 'linc\_us4\_oe' | 'linc\_us5'

{backend, linc\_us4},

%% Configuration of the controllers switch will connect to. Ideally

%% this list should be empty and assignement should be handled by an

%% OF-Config client.

%% Default OFP controller port is 6653.

{controllers,

[

%% {"Switch0-DefaultController", "localhost", 6653, tcp}

%% To establish auxiliary connections to the controller specify them

%% in the list of additional options. For example to start 2 tcp and one

%% tls auxiliary connections provide config as follows:

%% {"Switch0-DefaultController", "localhost", 6653, tcp,

%% [{auxiliary\_connections, [{tcp, 2}, {tls, 1}]}]

%% }

]},

%% Enable or disable accepting connections from OFP controllers.

%% The switch may optionally accept TCP connections from OFP controllers

%% trying to connect to it.

%% Allowed values: 'disabled' | '{BIND\_ADDRESS, LISTENING\_PORT, tcp}'

%% Example:

%% {controllers\_listener, {"127.0.0.1", 6653, tcp}}

{controllers\_listener, disabled},

%% Enable or disable queues subsystem. Queue configuration is not part

%% of the OpenFlow specification and as such should be considered

%% EXPERIMENTAL. This feature is disabled by default.

%% Allowed values: 'enabled' | 'disabled'

{queues\_status, disabled},

%% The datapath\_id uniquely identifies a datapath. The lower

%% 48 bits are intended for the switch MAC address, while

%% the top 16 bits are up to the implementer.

{datapath\_id, "00:00:00:00:00:00:00:01"},

%% To assign a port to a logical switch two requirements has to be met:

%% a. the port has to be defined in the `capable\_switch\_ports`,

%% b. the port has to be listed in the `ports` for the logical switch.

%%

%% Note that logical switch port numbers do not have to be the same

%% as capable switch port numbers. If the `port\_no` option is not

%% provided it defaults to capable switch port number.

%%

%% If `port\_name` option is not provided it defaults to

%% `Port{LOGICAL\_SWITCH\_PORT\_NO}` or `Port{CAPABLE\_SWITCH\_PORT\_NO}`

%% if logical port number is not provided.

%%

%% If queues are enabled, assign them to ports.

%% Remember to set appropriate port rates in `capable\_switch\_ports`.

%% Queue configuration is not part of the OpenFlow specification

%% and as such should be considered EXPERIMENTAL.

{ports, [

%% Examples:

%% - port without queues:

%% {port, 1, [{queues, []}]},

%% - port with two queues:

%% {port, 2, [{queues, [1, 2]}]},

%% - port with explicit number and name

%% {port, 3, [{queues, []},

%% {port\_no, 11},

%% {port\_name, "Banshee"}]}

%% NOTE: The third element of the port tuple can be

%% either a {queues, \_} tuple or a list of options. It

%% is recommended to use a list of options.

]}

]}

%% Capable Switch context allows you to start multiple Logical Switches

%% inside one instance of LINC.

%% , {switch, 1,

%% [

%% {backend, linc\_us3},

%% {controllers, []},

%% {ports, []},

%% {queues\_status, disabled},

%% ]}

]}

%% TLS configuration. Put your switch certificate and private RSA key here.

%% Values should be base64 encoded, DER encoded strings.

%% , {certificate, ""},

%% {rsa\_private\_key, ""}

]},

{epcap,

%% epcap is a LINC dependency that reads packets from the eth interfaces (not

%% tap interfaces). It's a wrapper for pcap.

[

%% Setting this option to true will enable logging in libpcap. The log

%% messages will be printed to:

%% a. rel/linc/log/erlang.log.N file when started as daemon (more

%% information here: http://www.erlang.org/doc/man/run\_erl.html under

%% "Notes concerning the log files"),

%% b. the application console when started in console mode.

{verbose, false},

%% This options sets interval of printing pcap statistics. The value is in

%% seconds.

%% NOTE: This option will have effect ONLY if verbose is set to true.

{stats\_interval, 10}

%% This option allows manipulating internal buffer size of pcap. By

%% increasing this buffer dropping packet can be mitigated. The value is in

%% bytes.

%% , {buffer\_size, 73400320}

]},

{of\_protocol,

[

%% This flag allows to disable splitting messages into multipart

%% messages.

%% {no\_multipart, true}

]},

{enetconf,

[

{capabilities, [{base, {1, 0}},

{base, {1, 1}},

{startup, {1, 0}},

{'writable-running', {1, 0}}]},

{callback\_module, linc\_ofconfig},

{sshd\_ip, any},

{sshd\_port, 1830},

{sshd\_user\_passwords,

[

{"linc", "linc"}

]}

]},

{lager,

[{colored, true},

{handlers,

[{lager\_console\_backend, info},

%% To enable 'debug' logs on the console change 'info' to 'debug'.

%% However that will produce big amount of logs from dependencies.

%% To filter them out and display only 'debug' messages from LINC

%% application, change the above 'lager\_console\_backend' into the

%% one presented below. Note that all the 'debug' logs can be for

%% example redirected to a file by setting appropriate log level

%% in the 'lager\_file\_backend'.

%% {lager\_console\_backend,

%% [debug,

%% {lager\_default\_formatter,

%% [{linc,

%% [time, " [", severity, "] ", message, "\n"],

%% [""]}]

%% }]

%% },

{lager\_file\_backend, [{file, "log/error.log"}, {level, error},

{size, 10485760}, {date, "$D0"},{count, 5}]},

{lager\_file\_backend, [{file, "log/console.log"}, {level, info},

{size, 10485760}, {date, "$D0"},{count, 5}]}

]}

]},

{sasl,

[

{sasl\_error\_logger, {file, "log/sasl-error.log"}},

{errlog\_type, error},

{error\_logger\_mf\_dir, "log/sasl"}, % Log directory

{error\_logger\_mf\_maxbytes, 10485760}, % 10 MB max file size

{error\_logger\_mf\_maxfiles, 5} % 5 files max

]},

{sync,

%% Sync is a developer utility that reloads changed beam files into the VM

%% without the need to restart it. It is started when the Makefile's dev

%% target is invoked to start the development environment.

[

%% By default only the procket module is excluded from the scan process

%% as it does not support reloading.

{excluded\_modules, [procket]}

]}

].

NETCONF configuration file for onos:

{

"devices": {

"netconf:192.168.56.102:1830": {

"netconf": {

"ip": "192.168.56.102",

"port": 1830,

"username": "linc",

"password": "linc"

},

"basic": {

"driver": "netconf"

}

}

}

}