

A Comparison of Multiprotocol Label Switching (MPLS) and OpenFlow Communication Protocols

By Dariusz Terefenko

X00088029

A thesis submitted in partial fulfilment of the requirements for
the
MSc. in Distributed and Mobile Computing



Institute of Technology Tallaght
Dublin 24
April 2018

Table of Contents

Abstract	8
1. Introduction	9
1.1. Limitations of IP	10
1.2. MPLS	10
1.2.1. MPLS in Linux	14
1.2.2. Cisco and MPLS	15
1.3. OpenFlow	15
1.3.1. OF Protocol	15
1.3.2. Switch Architecture	17
1.3.3. Flow Matching	19
1.3.4. Switch Ports	21
1.3.5. Pure vs Hybrid	22
1.3.6. Connection Interruption	23
1.3.7. Real World	26
1.3.8. Message Types	26
1.4. SDN	32
1.4.1. SDN Considerations	32
1.4.2. NBI vs SBI	33
1.4.3. NFV	36

1.4.4. CORD	36
1.4.5. Available Controllers	37
1.4.6. White-box Switching	37
1.4.7. Software Defined WAN	39
1.4.8. Advantages and Disadvantages	39
1.4.9. Deployment Approaches	40
1.4.10. Controller Security	42
1.4.11. Traditional vs OF Forwarding	47
1.4.12. Reactive vs Proactive	49
2. Test Environment	52
2.1. Cisco 2801	52
2.1.1. Hardware Routers	52
2.1.2. Hardware Configuration	53
2.2. Traffic Generators	53
2.2.1. MGEN	54
2.2.2. iPerf	54
2.3. Hyper-V	54
2.4. Mininet	55
2.5. Wireshark	59
2.6. Controllers and Mininet	59

2.6.1. OpenDaylight	59
2.6.2. Floodlight	62
2.6.3. HPE Aruba VAN	64
2.6.4. ONOS	66
2.6.5. POX	69
2.6.6. Ryu	71
2.7. Test Methodology	75
2.7.1. Software Routers	75
2.7.2. Hardware Routers	76
2.7.3. Software Switches	77
3. Performance and Compatibility Tests	78
3.1. Compatibility between MPLS in Linux and Cisco MPLS	78
3.1.1. Configuration	79
3.1.2. Results	82
3.2. IP Performance	89
3.2.1. Configuration	89
3.2.2. Results	92
3.2.2.1. Throughput	92
3.2.2.2. Delay	95
3.2.2.3. Packet Loss	97

3.2.2.4. RTT	100
3.3. Scalability	102
3.3.1. Configuration	102
3.3.2. Results	106
3.4. QoS	108
3.4.1. Configuration	109
3.4.1.1. MPLS	109
3.4.1.2. OpenFlow	115
3.4.2. Results	126
3.4.2.1. MPLS	126
3.4.2.2. OpenFlow	137
3.5. Traffic Engineering	148
3.5.1. MPLS	148
3.5.2. OpenFlow	152
3.6. Failover	164
3.6.1. MPLS	164
3.6.2. OpenFlow	167
3.7. Summary of Tests	170
4. Summary	171
4.1. Research Work	171

4.2. Conclusions	172
4.3. Future Work	174
List of Abbreviations	176
List of Figures	183
List of Appendices	190
Appendix 1	190
Appendix 2	193
Appendix 3	194
Appendix 4	198
Appendix 5	206
Appendix 6	212
Appendix 7	229
Appendix 8	252
Appendix 9	264
Appendix 10	279
Appendix 11	311
List of References	317

Declaration of Own Work

I hereby certify that the material, which I now submit for assessment on the programmes of study leading to the award of Master of Science, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution.

Signature of Candidate

Date

Acknowledgements

Personally, I would like to thank my family and friends for supporting me throughout last year, practically and with moral support, especially my wife. I am exceptionally grateful to Dr. David White for providing me with advice and support as my supervisor.

Thanks also go to Nasir Hafeez for providing me with beneficial information based on his experience whilst conducting extensive research to MPLS protocol on Cisco devices.

Abstract

The main reason behind this thesis was to compare the effectiveness of Multiprotocol Label Switching (MPLS) and OpenFlow (OF) protocols in computer networks. Paper discussed examples of these Internet Protocols (IPs) for network Traffic Engineering (TE) and Quality of Service (QoS) as well as scalability and interoperability with outlook for performance comparison. Test environments were created using Hardware (HW) routers and Hyper-V technology as well as a Mininet environment while performing experiments with Software Defined Networking (SDN). During the experiments software routers were used with the Linux operating system with the addition of MPLS in Linux and FRRouting (FRR), Cisco physical and virtual routers as well as a set of tools specially installed to generate the traffic and capture results for analysis such as Wireshark, iPerf, and MGEN.

Keywords: Access Control List (ACL), Assured Forwarding (AF), Best Effort (BE), Broadcast Domain Discovery Protocol (BDDP), Border Gateway Protocol (BGP), Class-Based Weighted Fair Queuing (CBWFQ), Customer Edge (CE), Cisco, Class of Service (CoS), Cloud Services Router (CSR), Datacentre, Differentiated Services Codepoint (DSCP), Enhanced Interior Gateway Routing Protocol (EIGRP), Floodlight, FFRouting (FRR), Generic Routing Encapsulation (GRE), Hybrid, Hyper-V, Internetwork Operating System (IOS), Internet of Things (IoT), iPerf, Label Distribution Protocol (LDP), Label Edge Router (LER), Linux, Label Switching Path (LSP), Label Switched Router (LSR), Mininet, Multiprotocol Border Gateway Protocol (MP-BGP), Multiprotocol Label Switching (MPLS), Network Address Translation (NAT), MPLS-TE, Network-Based Application Recognition (NBAR), Northbound Interface (NBI), OpenDaylight (ODL), ONOS, OpenFlow, Open Virtual Switch Database Management Protocol (OVSDB), Policy Based Routing (PBR), Provider Edge (PE), POX, Quality of Service (QoS), Resource Reservation Protocol (RSVP), Route Target (RT), Round-Trip Time (RTT), Ryu, Southbound Interface (SBI), Software Defined Networking (SDN), Spanning Tree Protocol (STP), Transmission Control Protocol (TCP), Traffic Engineering (TE), Type of Service (ToS), Traffic Engineering, Ubuntu, User Datagram Protocol (UDP), Virtual Application Network (VAN), Virtual Host (vHost), Virtual Network Interface Card (vNIC), Virtual Operating System Environment (VOSE), Virtual Private Network (VPN), Virtual Routing and Forwarding (VRF), vSwitch, Wireshark..

1. Introduction

Nowadays it's hard to imagine a computer without a network connection. The most common, and one that is growing at a truly breath-taking speed is the Internet which, according to Internet Live Stats (2017), has grown to more than 3 billion users from 1 billion in 2005. We must highlight that currently, users access the Internet not only from their Personal Computers (PCs) but also from their mobile phones, tablets, cameras or even household appliances and cars.

This exponentially growing number of connections means that computer networks, in the field of Information Technology (IT), have become integral to research and experimentation in order to keep pace with and allow for, the Internet's rapid growth.

This thesis also matches the trends of these studies where we conducted a series of experiments and tests which were designed to examine the operational efficiency and manageability of network traffic with MPLS and OpenFlow. These technologies are assumed to help solve problems such as transmission of multimedia in real time, providing services that meet the criteria for Quality of Service (QoS), the construction of scalable virtual private networks and efficient and effective Traffic Engineering (TE).

Subsequent chapters describe in detail the scope of work performed and lessons learned from practical scenarios and they also point to the potential use of these results. The next sections contain a description of the network protocols such as OpenFlow and MPLS as well as SDN and its operation within OpenFlow boundaries. It also includes an explanation of why we should use OpenFlow as an extension of IP technology, what are the security risks and how to deploy such protocol. It follows next to chapter two which includes concise description of the environment used to perform the study of protocol effectiveness. This section also presents a detailed walkthrough of the environment set up based on available Cisco Internetwork Operating System (IOS) routers, Ubuntu with software router implementation and Hyper-V virtualization technology and how it can be used for the testing of performance and capabilities of MPLS and OpenFlow protocols such as scalability, QoS, TE and link failover. It also includes explanation of various OpenFlow topologies and SDN controllers to highlight their operational differences. The third chapter is an accurate description of the experiments with the obtained results and a brief commentary on them. The last, fourth chapter has been dedicated to present the findings which were concluded with work carried out as well as future work proposed for further research. Due to complexity of the research it also includes a list of appendices with scripts and commands used to configure the devices.

1.1. Limitations of IP

As already mentioned in the introduction, the Internet network is constantly evolving and in recent years we have seen a very dynamic growth of its importance in our daily lives. Looking at it from the technical side we find that it is based on the IP protocol. While using this technology, however, we are not able to provide good performance for data services with guaranteed quality (Smith and Accedes, 2008). This is a serious problem if we want to deliver multimedia in high quality through a network satisfying the conditions of real-time. To solve this problem, many began to work on an approach for data, voice, and video to broadcast via telecommunications networks in a unified way, in the form of a packet (Gupta 2012). This, however, requires a modification to the network architecture which is generally referred to as a Next Generation Network (NGN).

Another issue which we cannot solve with the use of IP technology is the creation of effective mechanisms to control and manage the movement of packets across the network, the so-called Traffic Engineering (TE). This is according Mishra and Sahoo (2007) due to the restrictions applied to dynamic routing protocols, e.g. Open Shortest Path First (OSPF) which doesn't allow to define the arbitrary data flow paths.

Described problems, however, can be solved using MPLS or OpenFlow.

1.2. MPLS

MPLS was developed in the late 90's of the twentieth century by a group of the Internet Engineering Task Force (IETF). In principle, it not supposed to substitute any already used communication protocols including the most common IP, but it should extend it (Rosen et. al, 2001). MPLS can work with network technologies such as TCP/IP, ATM, Frame Relay as well as Synchronous Optical Networking (SONET). In this work, we will focus on showing the use of MPLS in conjunction with networks operating in the TCP/IP suite.

MPLS is called a layer 2.5 protocol in the ISO Open Systems Interconnection (OSI) model, because it operates between data and network layers, as seen in *Figure 1.1*.

OSI Model - 7 Layers

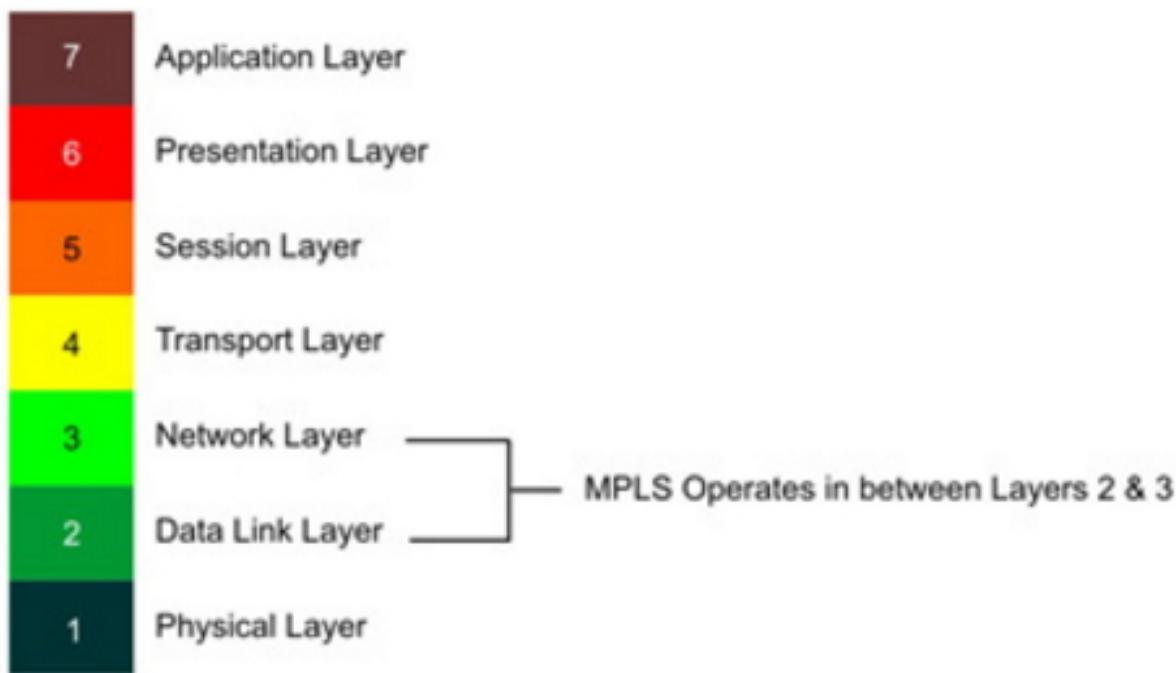


Figure 1.1: OSI Model and MPLS.

According to the Request for Comments (RfCs) it combines the advantages of the data link layer, such as performance and speed, as well as the network layer (and its scalability).

With MPLS we get a richer set of tools for network management and TE which allows transmission of packets through arbitrarily specified routes which are not possible to be defined with the use of classical routing protocols. In addition, MPLS allows the creation of Virtual Private Networks (VPNs).

In the MPLS domain, IP routing is replaced by a label switching mechanism, so it is necessary to understand some of the terms below (MPLS Info, 2017).

- Label Edge Router (LER) - a router running on the edge of an MPLS domain. It analyses the IP packets that enter the MPLS-enabled network, then adds the ingress router header and removes the MPLS header from the egress router.
- Label Switching Path (LSP) - a path that connects two edge routers (LERs) determining the flow path of packets marked with a label.
- Label Switched Router (LSR) - a router operating within an MPLS-based network. It is responsible for switching labels which allows to transfer packets in a defined path.
- Forwarding Equivalence Class (FEC) - a group of packets that have the same requirements for moving them. All packets in this group are treated identically and

travel in the same way.

According to Abinaiya and Jayageetha (2015), an MPLS label is attached by the LER at the time the packet enters the network while using this protocol. A 32-bit label is added between the second layer (Ethernet header) and the third layer (IP header), as seen in *Figure 1.2*.

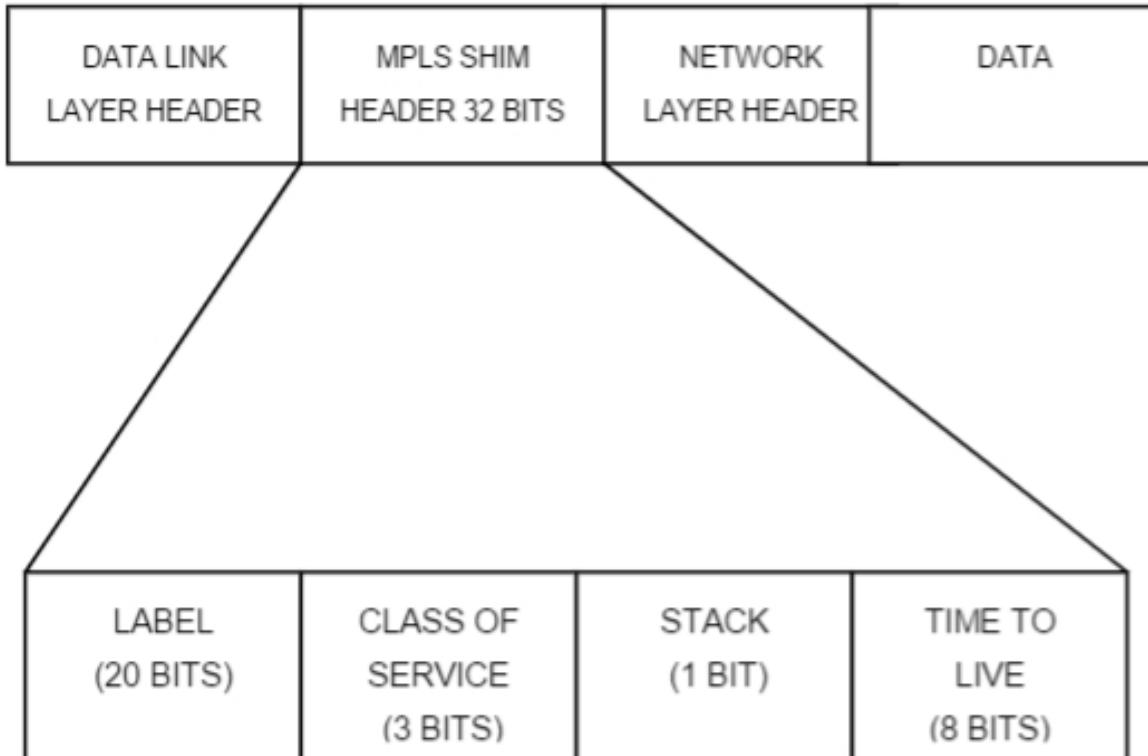


Figure 1.2: MPLS Label Format (Abinaiya and Jayageetha, 2015).

The MPLS header consists of (MPLS Info, 2017):

- Label field of 20 bits long.
- Experimental (EXP) or Class of Service (CoS) field of 3 bits which determines affiliation to a given traffic class.
- A 1-bit Stack (St) field that denotes to the bottom of the stack during label disposition (for the highest level this field is set to 1, for the remaining is 0).
- Time to Live (TTL) field whose function is analogous to the TTL field in the layer 3 header to prevent the looping.

When a label is attached, the LER passes the packet to the next node according to the corresponding entry in its Label Information Base (LIB). Each LSP router has defined appropriate rules for removing labels, adding and replacing as well as forwarding packets to subsequent nodes. This procedure is repeated until the packet reaches the last LER that is

responsible for removing the MPLS label. Donahue (2011) states that by replacing routing based on an IP header with labels of the same length during the path analysis we theoretically should speed up the data transmission. MPLS technology allows us to build a stack of labels where path switching operations are always performed on the top-level labels. This allows us to define tunnels which are useful for the creation of VPNs.

With MPLS paths we can apply TE. This is because the list of hosts through which the packet is routed is determined by the LER at the time the packet enters the MPLS domain, as seen in *Figure 1.3*.

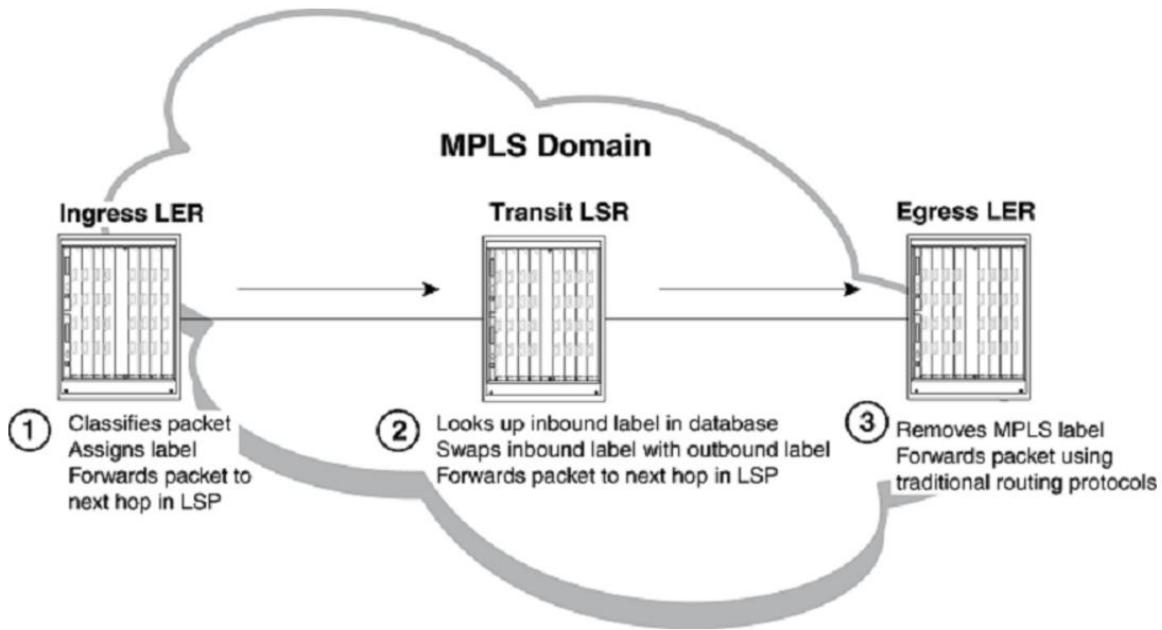


Figure 1.3: MPLS Label Switching (Brocade Communications Systems, 2015).

This allows us to route traffic other than it would be from the classical routing protocols, as we can select a path that has reserved resources that meet QoS requirements. To do so we can use Open Shortest Path First (OSPF) or Intermediate System to Intermediate System (IS-IS) protocols to monitor the current connection status and then provide adequate LSP for each traffic class.

According to Partsenidis (2011) with MPLS we can easily set up tunnels between the selected network nodes that can be used to create VPNs. The stack of labels is used together with Virtual Routing and Forwarding (VRF) table at the ingress LER of the Customer Edge (CE) router where it adds two MPLS labels to the packet, as seen in *Figure 1.4*.

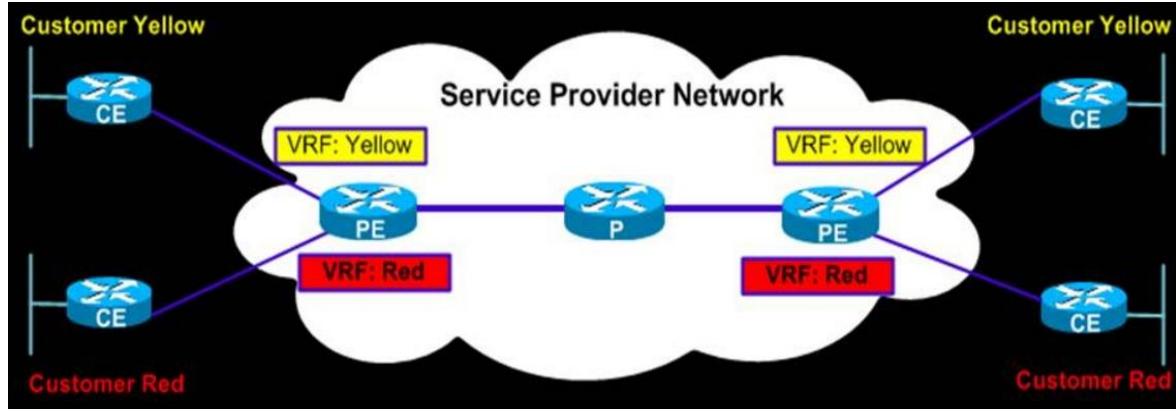


Figure 1.4: VRF and VPN (Partsenidis, 2011).

The first label provides the outgoing interface within the specified VPN the information about the target boundary router and the second defines the path to the egress LER.

With this approach, we can guarantee logical separation between different VPNs using one common network infrastructure. In comparison to the implementation of a VPN based on the physical separation of communication channels by using MPLS, we gain much greater scalability and ease of adding new users. In addition, each type of user-to-user communication is possible without the creation of an extensive Point to Point (P2P) connections.

A characteristic feature of the MPLS protocol is the multitude of its versions that are not always fully compliant with the standard which in turn causes them to be incompatible with each other. There are open source solutions as well as those typically commercially developed by such companies as Cisco (Cisco, 2007), Hewlett Packard (Havrila, 2012) and Juniper (Juniper, 2016).

In the next subsections, I will briefly discuss two different projects that I have dealt with during the research conducted for this work.

1.2.1. MPLS in Linux

MPLS for Linux is an open source project which adds to the Linux kernel the support of MPLS (Leu, 2013). Creators of the project provided an upgrade to the Linux kernel which allows the use of this protocol where we can transform a PC into a software router with MPLS.

The project has not been extensively developed since it was manifested with the compatibility of kernel version of Linux marked as 2.6.x. Its advantage was, however, the stability of operation and therefore in 2015 MPLS support was widely introduced in Linux 4.1 and developed since then by adding extra functionalities (Kernel Newbies, 2015).

1.2.2. Cisco and MPLS

Cisco has developed an MPLS protocol for commercial technology as opposed to the free software solution presented above. The manufacturer advertises it as the perfect solution which can be a backbone of smart grids for NGNs. In accordance with the requirements of the version of the protocol implemented by Cisco, it can work as an extension to IP, ATM, Frame Relay, and Ethernet. These technologies are supported by a wide range of device manufacturers.

According to Cisco (2002), it caters for the scalability of VPNs to accelerate the Internet growth for service providers, provides “*constraint-based routing*” to use the shortest path for the traffic flow to reduce the congestion as well as to make the best use of network resources.

1.3. OpenFlow

1.3.1. OF Protocol

OpenFlow (2011) is an open source project which was developed in the first decade of the twenty-first century by Stanford and California U.S. universities. Its first official specification was announced in late 2009 and is designated as version 1.0.0. Currently, the further work around this protocol is held by the Open Networking Foundation and the latest version announced in March 2015 is 1.5.1 (Open Networking Foundation, 2015). The creation of OpenFlow protocol is closely related to the concept of Software Defined Networking (SDN) which allows defining the work of computer networks. That notion was also invented at Stanford University in the same period as the protocol itself. In short, this concept involves separating high-level control over the flow of network traffic in the control plane from the hardware solutions responsible for the forwarding of the packets in the data plane. With this approach, we can manage traffic in the network in the same way regardless of equipment used.

Use of OpenFlow provides similar benefits as offered by MPLS. We receive a rich set of tools that will let us engineer the traffic to optimize the transmission to ensure adequate throughput, avoid delays or the number of connections through which the packets are routed. According to Chato and Emmanuel (2016), it also provides us with a solution which can offer services that meet QoS requirements.

With OpenFlow, we can create VPNs and even something more because of the network multitenancy (Sanchez-Monge and Szarkowicz, 2016). This protocol introduces the concept of traffic flows which caters for both network virtualization and separation of traffic. This gives administrators new opportunities for expansion, modification, and testing of networking infrastructure. They can explore new solutions using the production environment and at the same time do not interfere with the normal operation of the network.

OpenFlow is not a revolutionary solution as it would take into consideration the capabilities of its MPLS predecessor, so the question arises, why the new technology is so intensively developed? Its creators believe that thanks to one strictly defined standard we can programmatically define the operation of the network regardless of the manufacturer of equipment being used to implement it. MPLS, though, has a specific standard which is often implemented differently by vendors. In addition, the OpenFlow protocol is simpler than its predecessor. Undoubtedly, OpenFlow's advantage is that the concept of an “*Open Network*” is currently being developed by the Open Networking Foundation (ONF), an organization founded by the most respected computer technology giants such as Facebook, Google, Microsoft, Intel, Cisco and many others (Open Networking Foundation, 2017). This is an open source solution supported by an increasingly widespread dedicated network device, but it can also work on ordinary PCs running Linux Operating System (OS).

OpenFlow is a protocol operating in the second layer of the ISO OSI model what distinguishes it from the MPLS protocol which works in both the data link and the network layer. Network switches are the most common devices that operate in the data link layer. According to Goransson and Black (2014), three components are needed to create a network based on OpenFlow technology: a switch that supports this protocol, a controller and a communication channel which is used by OpenFlow protocol through which the controller and switch can communicate, as seen in *Figure 1.5*.

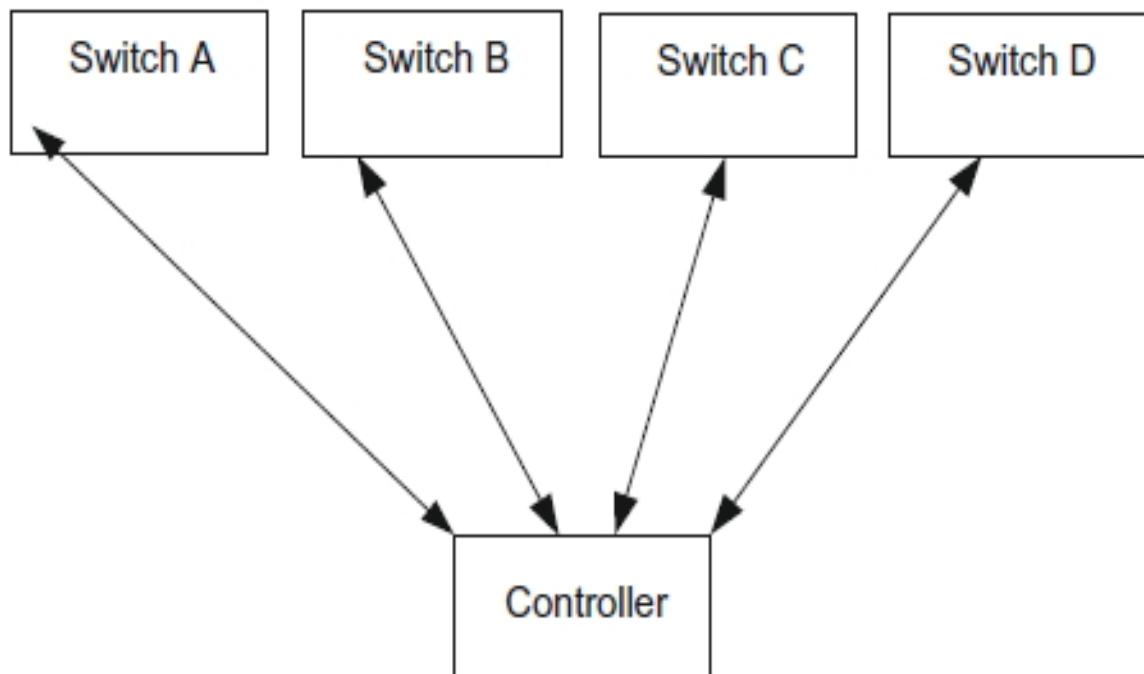


Figure 1.5: OpenFlow Components (Goransson and Black, 2014).

1.3.2. Switch Architecture

An OpenFlow switch is a network infrastructure component that operates in the second layer of the ISO OSI that holds in its memory flow tables and has a communication channel that can be used to communicate with the controller. We can distinguish specialized physical devices such as dedicated switches and programmable switches that support these technologies which are usually ordinary computers operating under the control of a properly prepared OS. The concept of the flow table deserves further discussion (*Figure 1.6*).

Flow Entry 0		Flow Entry 1		Flow Entry F		Flow Entry M	
Header Fields	Import 12 192.32.10.1, Port 1012	Header Fields	Import * 209.*.*., Port *		Header Fields	Import 2 192.32.20.1, Port 995		Header Fields	Import 2 192.32.30.1, Port 995
Counters	val	Counters	val		Counters	val		Counters	val
Actions	val	Actions	val		Actions	val		Actions	val

Figure 1.6: OpenFlow Flow Tables (Goransson and Black, 2014).

It consists of three major elements: header fields which are created from the packet header, counters which are statistical information such as the number of packets and bytes sent and the time since the last packet matched the rule, action fields which specify the way the package was processed. Entries to it are added via the controller. They specify how the switch should behave after receiving the packet that meets the matching condition. The switch can send data to the output port, reject it or send it to the controller. This last action is most often performed when the switch, after receiving the packet, is unable to match it to any of the existing rules. In this situation, the controller makes the appropriate decision on which rules to add to the flow table so that action against the proceeding similar packet is taken only by the switch.

The appropriate rule is added to its flow table in the switch via the controller. The functions of the controller may be served by some very sophisticated program that controls the flow according to certain criteria or a very simple mechanism that will add static entries to the switch memory.

The communication channel is a very important element in networks based on OpenFlow technology. It is used to facilitate communication between switches and the controllers which is crucial since the actual decisions about network traffic management are made by the controller and must be propagated among the switches. The data sent through this channel must conform to the OpenFlow specification and is usually encrypted using Transport Layer Security

(TLS), as seen in *Figure 1.7*.

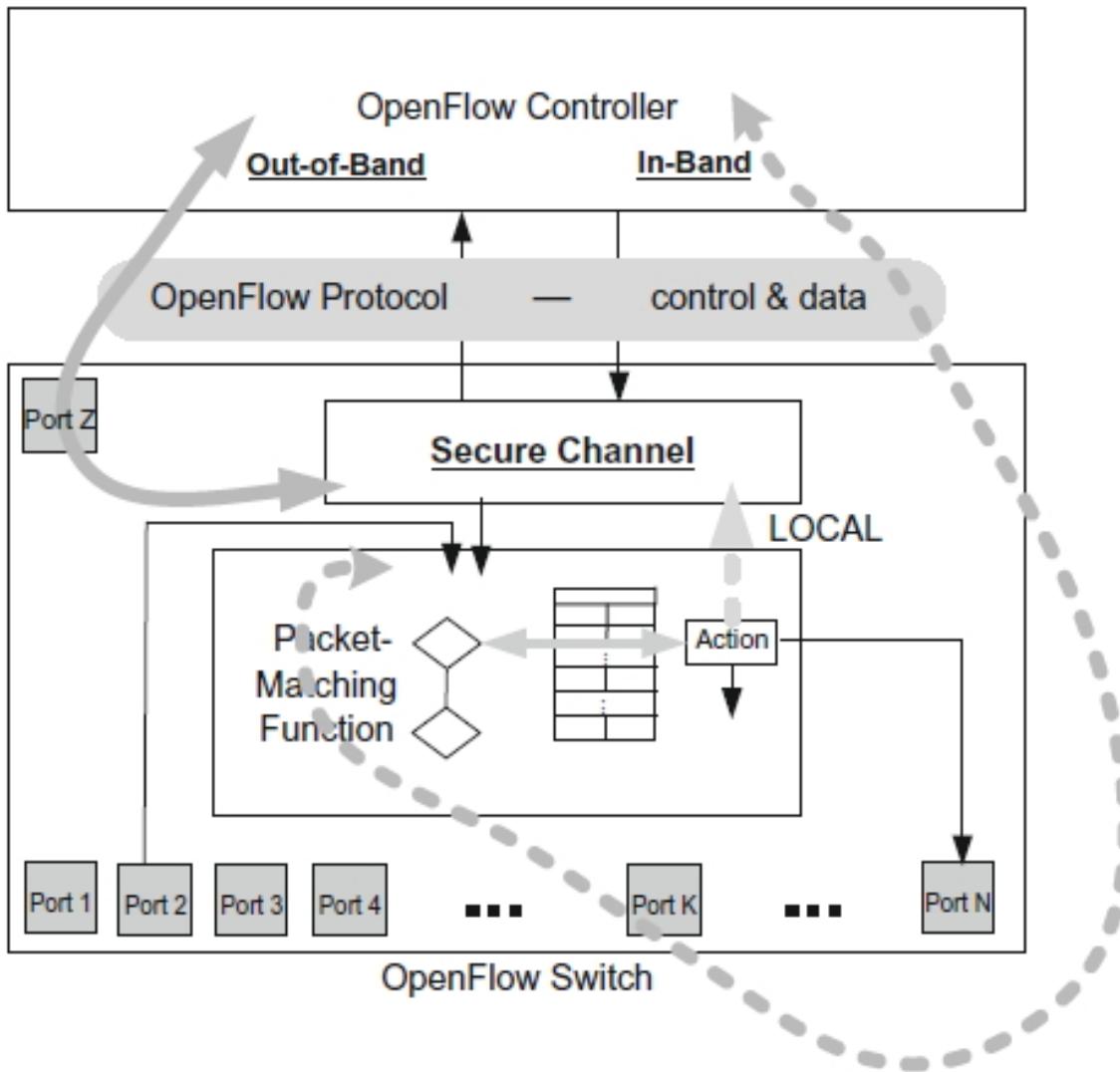


Figure 1.7: OpenFlow Controller to Switch Secure Channel Communication (Goransson and Black, 2014).

Most controllers support OpenFlow version 1.3.2 rather than 1.4 or 1.5. OF switch might support one or more flow tables within the pipeline, but only needs to support one table to be compliant with the standard. However, use of multiple tables provides scalability for larger infrastructures. Switch OpenFlow channel provides a connection to the external controller and a group table which is responsible for grouping the interfaces into a single channel for broadcast or multicast requests, as seen in *Figure 1.8*.

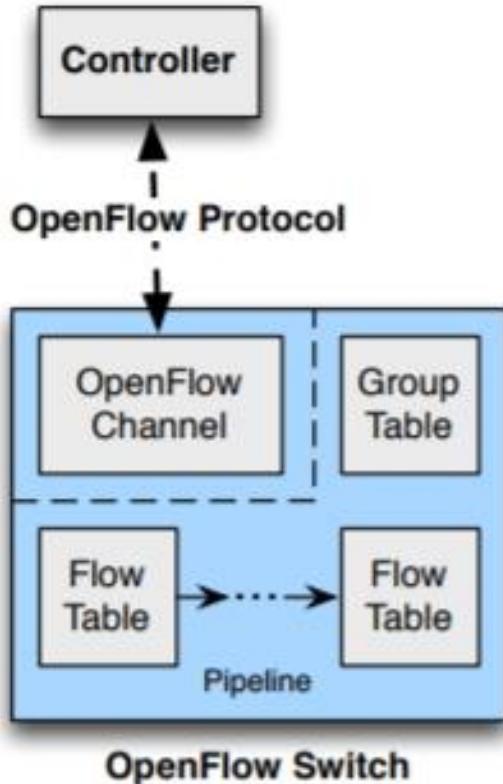


Figure 1.8: Components of OpenFlow Switch (Open Networking Foundation, 2013).

The switch can be either a Mininet, OVS or physical HW which uses OF protocol to communicate with the external controller via TCP or TLS to perform packet lookups for forwarding decisions.

The controller is decoupled from the switch in the control plane usually running on a Linux box and it manages the switch via OF to add, update and delete flow entries in a reactive or proactive manner.

Each switch can have up to 254 flow tables and matching of the packets starts on Flow Table 0 which was originally supported in OF version 1.0, while other versions support multiple tables in the pipeline by using goto-table instructions. Flow entries are matched in order of priority from higher to lower when instructions are executed and if no entries are matched then a table-miss entry with the priority of 0 is used.

1.3.3. Flow Matching

To explain the concept of the flow tables we are going to use Mininet and HPE Aruba VAN SDN controller with a topology which will consist of two switches and two hosts.

Since our Mininet environment is remote from the controller we are going to use SSH to connect to it and execute this command: `sudo mn --controller=remote, ip=192.16.20.254 --`

`topo=single,2 --switch=ovsk,protocols=OpenFlow10-mac`, which will build the infrastructure with OF10 and make it simple to read MAC addresses. Next, we execute `pingall` to allow the controller to discover the hosts.

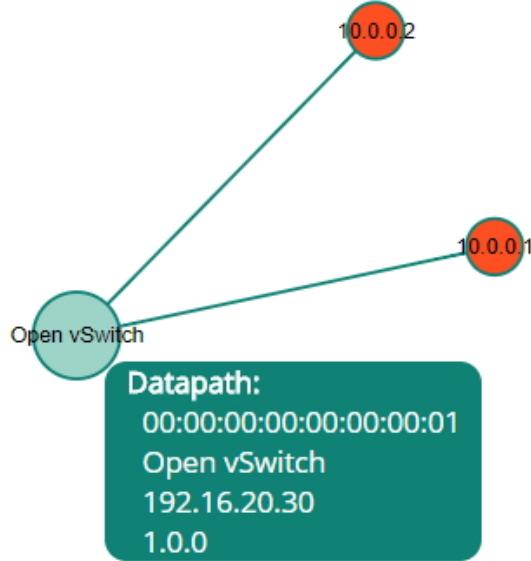


Figure 1.9: Single Topo with HPE Aruba VAN Controller.

From *Figure 1.10* below we can see the Datapath ID of the switch and the single flow table with all matches and actions which, as well as the table-miss entry, is responsible for allowing our ICMP packets to flow via traditional forwarding mechanism.

Flows for Data Path ID: 00:00:00:00:00:00:00:01					
Table ID	Flow Count	Table Name	Match	Actions/Instructions	
0	5				
60000	0		eth_type: bddp	output: CONTROLLER	
31500	0		eth_type: ipv4 ip_proto: udp udp_src: 67 udp_dst: 68	output: CONTROLLER	output: NORMAL
31500	0		eth_type: ipv4 ip_proto: udp udp_src: 68 udp_dst: 67	output: CONTROLLER	output: NORMAL
31000	4		eth_type: arp	output: CONTROLLER	
0	24			output: NORMAL	output: NORMAL

Figure 1.10: Table-miss Flow Entry with HPE Aruba VAN Controller.

Now we have created a flow entry for ICMP packets with a higher priority without any action associated to it which resulted in a lack of connectivity between the hosts, as seen in *Figure 1.11*.

Flows for Data Path ID: 00:00:00:00:00:00:00:01					
Table ID	Flow Count	Table Name			
0	6				
▶ 60001	32	Packets	Bytes	Match eth_type: ipv4 ip_proto: icmp	Actions/Instructions
▶ 60000	0	0	0	eth_type: bddp	output: CONTROLLER
▶ 31500	0	0	0	eth_type: ipv4 ip_proto: udp udp_src: 67 udp_dst: 68	output: CONTROLLER output: NORMAL
▶ 31500	0	0	0	eth_type: ipv4 ip_proto: udp udp_src: 68 udp_dst: 67	output: CONTROLLER output: NORMAL
▶ 31000	6	252	252	eth_type: arp	output: CONTROLLER output: NORMAL
▶ 0	26	2068	2068		output: NORMAL

Figure 1.11: New Flow Entry with HPE Aruba VAN Controller.

In OF 1.3 packets are only received on ingress ports while OF 1.5 also uses egress port matching in the pipeline.

In the test case, we have matched the flow entries to specific IP protocol, but OF 1.0 can match against any of the below:

Ingress Port	Ether source	Ether dst	Ether type	VLAN id	VLAN priority	IP src	IP dst	IP proto	IP ToS bits	TCP/UDP src port	TCP/UDP dst port

Figure 1.12: Fields to Match against Flow Entries (OpenFlow, 2009).

In OF 1.0 we only use one table in the pipeline, but OF 1.3 and higher can support a larger amount of flow tables starting from Flow Table 0. A singular table has its limitations, especially when taking into consideration MAC address learning with VLAN IDs or Reverse Path Forwarding Check (RPFC) which can result in overflow (Open Networking Foundation, 2015).

1.3.4. Switch Ports

Switches connect to each other via OF ports which are virtual/logic or HW ports where we can only enable specific ports. Physical ports map one-to-one to logical ports, so if we have link-aggregation with two NICs, then OF will not know if one of them will fail as it sees them as one logical port.

OF switches also have reserved ports which are defined by the specification and these represent forwarding actions such as sending traffic to the controller, flooding packets out, or forwarding with normal switch processing, as seen in *Figure 1.13*.

Flows for Data Path ID: 00:00:00:00:00:00:00:01					
Table ID	Flow Count	Table Name		Actions/Instructions	
0	6				
► 60001	3642	Bytes 356916	Match eth_type: ipv4 ip_proto: icmp	output: CONTROLLER	
► 60000	0	0	eth_type: bddp	output: CONTROLLER	
► 31500	0	0	eth_type: ipv4 ip_proto: udp udp_src: 67 udp_dst: 68	output: NORMAL	
► 31500	0	0	eth_type: ipv4 ip_proto: udp udp_src: 68 udp_dst: 67	output: CONTROLLER	
► 31000	216	9072	eth_type: arp	output: NORMAL	
► 0	32	2488		output: CONTROLLER	
				output: NORMAL	

Figure 1.13: Reserved Ports with HPE Aruba VAN Controller.

Port on the controller is the logical port, which caters for TCP or TLS session which can be used as ingress (incoming port) up to OF13 or egress (outgoing port) for OF15 where normal port represents the traditional routing and switching pipeline.

1.3.5. Pure vs Hybrid

OpenFlow-only switches support normal or flood ports within the pipeline, so they only operate in the data plane and rely on the intelligence of the controller to make the decision about the forwarding of the packets.

OpenFlow-hybrid switches support pure OF operations as well as normal switching mechanisms such as L2 switching, L3 routing, ACLs, VLANs and QoS. This means that they operate within normal pipeline with use of classification mechanisms such as VLAN tagging or input ports which also can go through an OpenFlow-only pipeline via flood or reserved ports. For example, one VLAN can use a pure OF pipeline while another would use traditional routing and switching on the same device.

In an OF pipeline, switch decisions are made by the controller wherein a traditional mechanism they're made locally on the switch. It's also possible to use an OF pipeline to use a normal forwarding method for traditional mechanism and then send it to the normal port for traditional routing and switching. For example, we can set up HTTP and FTP traffic on the controller to be sent out on specific ports while normal traffic can be sent to a normal port for traditional processing by use of table miss-entry in the flow table. This way it would use the MAC Address Table to forward the packets to the destination for entries not matching within the flow table. A very good example of a controller which works in hybrid mode is the previously used HPE Aruba VAN where we have seen that actions for matches against the flow

entries were set both to output packets on normal and controller ports.

1.3.6. Connection Interruption

When the switch will not receive a periodic echo reply message back, it would mean that there is a problem with the connection to the controller that would result in going into fail secure or fail standalone mode.

In fail secure mode, during first connection attempt, or in event of the disconnection of a switch from the controller, packets traversing to the controller will be dropped and flow entries will become deleted or they will expire as per their timeout settings. Hybrid switches, however, can operate in fail standalone mode during failure, so packets can be delivered with the use of a traditional forwarding method with the normal port.

Now we are going to add a simple flow entry into the previously discussed topology and then stop the controller to see its behaviour. OVS is operating in secure mode and our flow entry with the highest priority is in the flow table, as seen in *Figure 1.14*.

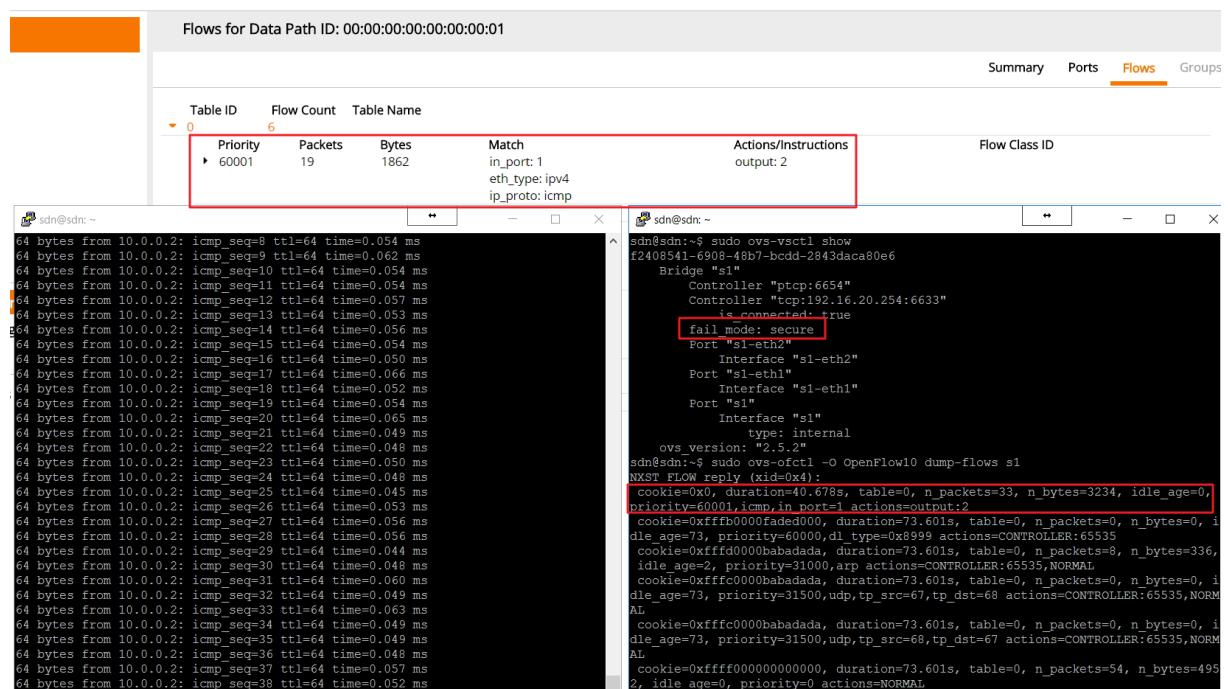


Figure 1.14: Hybrid Switch Secure Mode with Running HPE Aruba VAN Controller.

Next, we have stopped the controller and we still were able to see packets traversing from Port1 to Port2 on the switch, as seen in *Figure 1.15*.

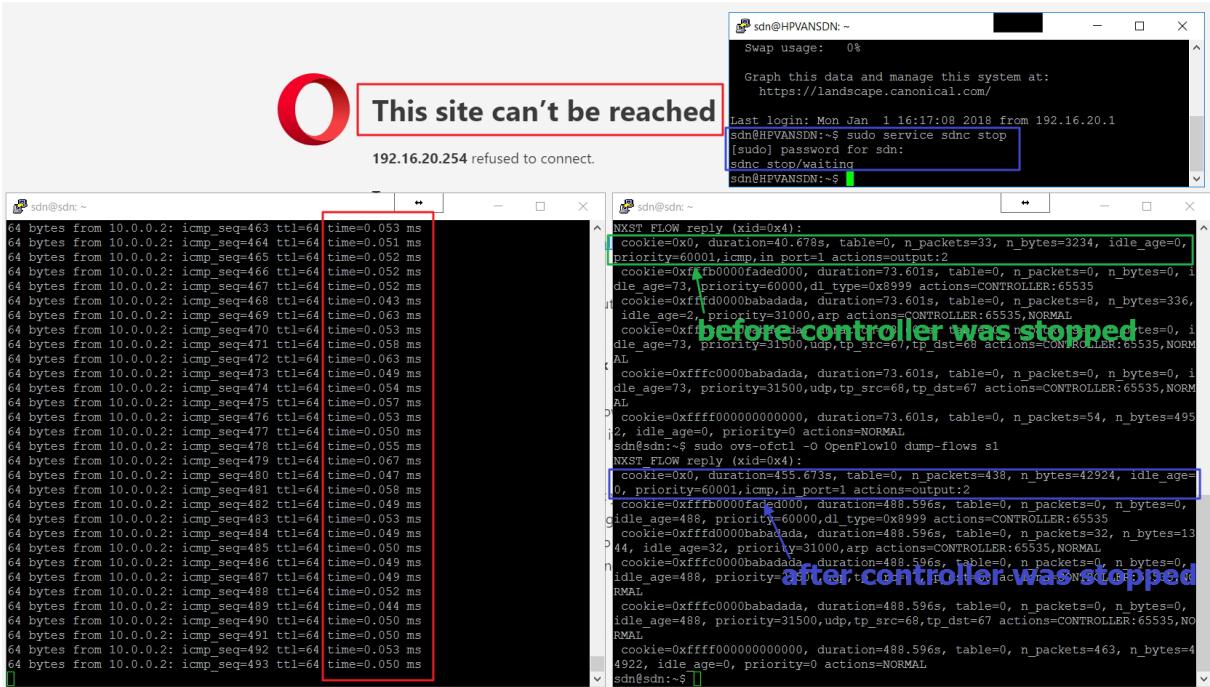


Figure 1.15: Hybrid Switch Secure Mode with Stopped HPE Aruba VAN Controller.

In the next test case we have turned off the hybrid mode and we managed to see that traffic was still being forwarded as the controller previously was using the secure mode during the connection, so flow entries were not removed from the flow table. However, any new entries by default will be forwarded to a normal OF pipeline, while other controllers might cause the packets to be dropped, as switches will not be able to match the entries while the controller is offline.

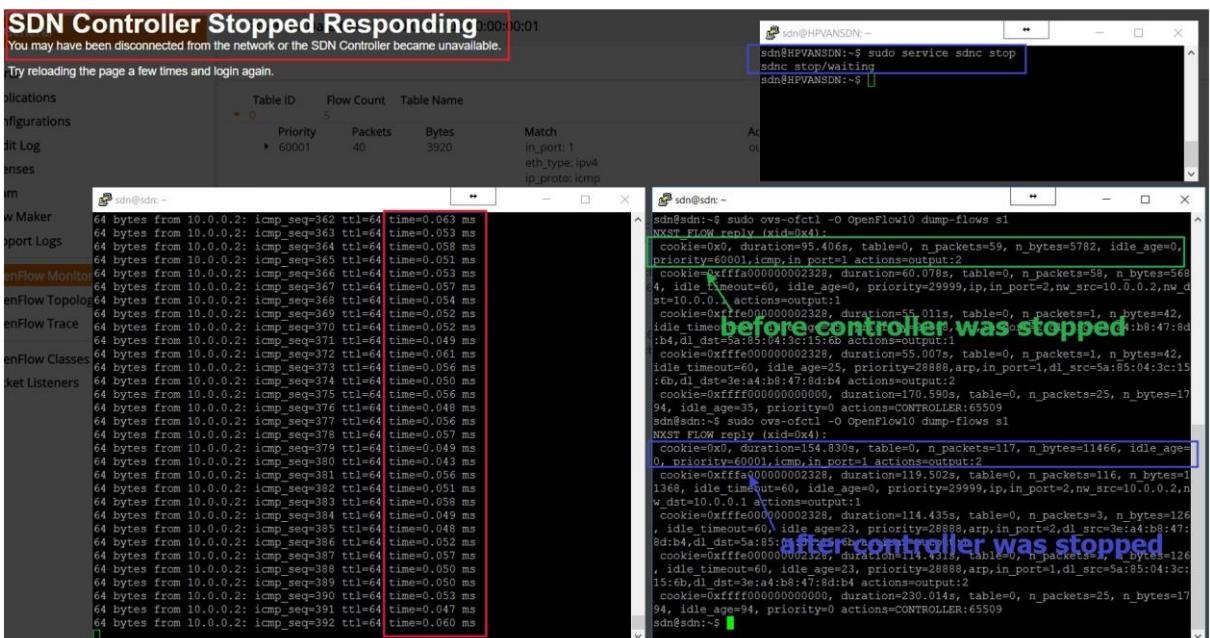


Figure 1.16: Pure OF Switch Secure Mode with Stopped HPE Aruba VAN Controller.

Now we are going to use ODL while we have changed the fail mode on the OVS to standalone via *ovs-vsctl set-fail-mode s1 standalone*, as seen in *Figure 1.17*.

The screenshot shows two terminal windows. The left window displays a ping test between hosts h1 and h2, showing 64 bytes of data being sent from 10.0.0.2 to 10.0.0.1 over 10.0.0.2. The right window shows the command `sudo ovs-vsctl set-fail-mode s1 standalone` being run, followed by the output of `sudo ovs-vsctl show`. The output indicates that the bridge "s1" is in standalone fail mode, with its ports "s1-eth2" and "s1-eth1" also in standalone mode.

```
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.453 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.052 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.038 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.047 ms
sdn@sdn:~$ sudo ovs-vsctl set-fail-mode s1 standalone
sdn@sdn:~$ sudo ovs-vsctl show
f2408541-6908-48b7-bcdd-2843daca80e6
Bridge "s1"
    Controller "ptcp:6654"
    Controller "tcp:192.16.20.32:6653"
        is_connected: true
    fail_mode: standalone
    Port "s1-eth2"
        Interface "s1-eth2"
    Port "s1"
        Interface "s1"
            type: internal
    Port "s1-eth1"
        Interface "s1-eth1"
            type: internal
    ovs_version: "2.5.2"
```

Figure 1.17: OVS and Standalone Mode with ODL Controller.

We can see that traffic between the hosts is still traversing, but there are no flow entries on the switch, as seen in *Figure 1.18*.

The screenshot shows two terminal windows. The left window displays a ping test between hosts h1 and h2, showing 64 bytes of data being sent from 10.0.0.2 to 10.0.0.1 over 10.0.0.2. The right window shows the command `sudo ovs-ofctl -O OpenFlow13 dump-flows s1` being run, followed by the output. The output shows several flow entries on port s1, but none on the controller ports s1-eth1 and s1-eth2. The command `sudo ovs-ofctl -O OpenFlow13 dump-flows s1` is run again, showing the same result.

```
sdn@sdn:~$ ping -c 10 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=74 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=75 ttl=64 time=0.052 ms
64 bytes from 10.0.0.2: icmp_seq=76 ttl=64 time=0.050 ms
64 bytes from 10.0.0.2: icmp_seq=77 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=78 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=79 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=80 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=81 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=82 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=83 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp_seq=84 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=85 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=86 ttl=64 time=0.145 ms
64 bytes from 10.0.0.2: icmp_seq=87 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=88 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=89 ttl=64 time=0.053 ms
sdn@sdn:~$ ovs-vsctl "ptcp:6654"
sdn@sdn:~$ ovs-vsctl "tcp:192.16.20.32:6653"
sdn@sdn:~$ is_connected: true
sdn@sdn:~$ fail_mode: standalone
sdn@sdn:~$ Port "s1"
sdn@sdn:~$     Interface "s1"
sdn@sdn:~$         type: internal
sdn@sdn:~$ Port "s1-eth2"
sdn@sdn:~$     Interface "s1-eth2"
sdn@sdn:~$ Port "s1-eth1"
sdn@sdn:~$     Interface "s1-eth1"
sdn@sdn:~$     ovs_version: "2.5.2"
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
OFPT_FLOW reply (xid=0x2):
cookie=0x2b00000000000002, duration=43.525s, table=0, n_packets=0, n_bytes=0, priority=100, dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x2a00000000000004, duration=12.844s, table=0, n_packets=13, n_bytes=1218, idle_timeout=600, hard_timeout=300, priority=10, dl_src=00:00:00:00:01, dl_dst=00:00:00:00:02 actions=output:2
cookie=0x2a00000000000005, duration=12.844s, table=0, n_packets=14, n_bytes=1260, idle_timeout=600, hard_timeout=300, priority=10, dl_src=00:00:00:00:02, dl_dst=00:00:00:00:01 actions=output:1
cookie=0x2b00000000000004, duration=41.528s, table=0, n_packets=3, n_bytes=266, priority=2, in_port=2 actions=output:1, CONTROLLER:65535
cookie=0x2b00000000000005, duration=41.528s, table=0, n_packets=31, n_bytes=1622, priority=2, in_port=1 actions=output:2, CONTROLLER:65535
cookie=0x2b00000000000002, duration=43.529s, table=0, n_packets=2, n_bytes=196, priority=0 actions=drop
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
OFPT_FLOW reply (xid=0x2):
sdn@sdn:~$
```

Figure 1.18: OVS Standalone Mode and Switch Flow Entries with Stopped ODL Controller.

We also notice that with the HPE controller hybrid switch, the action for table miss entries during interruption will cause the switch to forward the packets to the traditional mechanism, while after changing it to pure-OF, this action changes to the controller which effectively will result in the packets not being delivered for new flow entries.

In case flow entries are removed due to timeouts, the switch will not know where to

forward packets and it will simply drop them. Also, if there would be a new flow entry where the switch doesn't know what to do with it, it will also drop them. In OVS if the connection to the configured controller gets interrupted, no new connection will be configured, and no packets will pass through the switch. When the controller operates in standalone mode the switch will take responsibility for flows configuration after inactivity timeout and it will act as an ordinary MAC learning switch, while in secure mode it will not set up any new flows.

1.3.7. Real World

According to Fortier (2017), even VMware moves to SDN with native vSwitch products and support for OVS to simplify the platform which will result in a reduction of upgrade and deployment times. This means that it will no longer be possible to run Cisco Nexus 1000V in ESXi.

There is no requirement to change most of the devices to support OF as vendors such as HP (HPE Support, 2012) and Cisco (Cisco Support, 2017) provide firmware upgrades. This way in core or access layer we can use hybrid switches or OVS in Mininet and if OF isn't configured to operate with the controller, traditional routing and forwarding mechanisms will be used.

For example, we can use STP OF capabilities in the features message with VLANs in conjunction with non-OF switches to gradually implement the protocol within any network. OF switches might not support loop prevention, but this can be implemented either in the controller or app. We can also use multiple controllers to manage parts of the network independently from each other.

It's even possible to build and test a real physical network with a budget below 1,000 Euro using OF-only Zodiac FX switches (Kickstarter, 2015), Flow Maker Deluxe (Northbound Networks, 2016) and SDN controllers such as ODL, Floodlight, HPE Aruba VAN or Ryu before deployment in live Datacentre environment.

1.3.8. Message Types

The OpenFlow version 1.3.2 (OF13) specification distinguishes three types of communication (Open Networking Foundation, 2013), as seen in *Figure 1.19*.

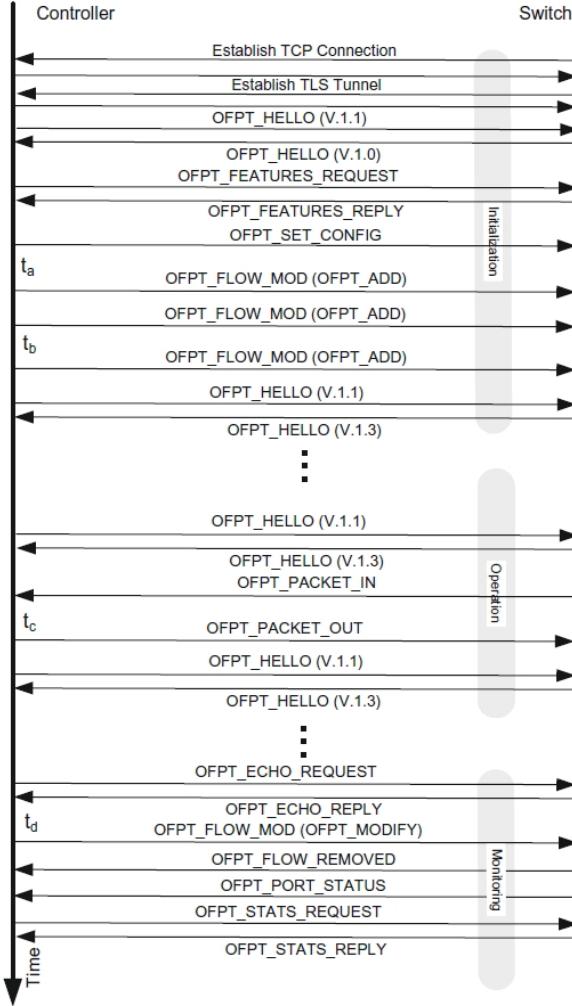


Figure 1.19: OpenFlow Controller to Switch Session (Goransson and Black, 2014).

In controller-to-switch communication, the controller sends a message to the switch and possibly receives a reply. These messages are used to manage the switch. The protocol supports eight main types of messages:

- Features used for request of supported features during session creation.
- Configuration used for configuration parameters.
- Modify-State is used for modification and reading of the connection state, adding/removing/altering flows/groups in OF tables.
- Read-State is used to get statistics.
- Packet-Out sends a packet to a switch.
- A barrier is used to confirm the packet delivery which ensures that dependencies between controller and switch are met.
- Role-Request is used to configure the roles for OF channel.

- Asynchronous Configuration sets a filter for asynchronous communication for the OF channel.

Asynchronous messages are initiated by switches and there are four basic types of messages: a packet receipt(Packet-In), a removal of an entry from the flow table (Flow-Removed), a request for port status (Port-Status) and an error message (Error).

Symmetric communication can be initiated by any of the sides and messages sent in this way are: Hello messages between switch and controller, Echo verifies the link and can be used to measure the latency as well as a vendor-specific message reserved to be used in the future (Experimenter).

OVS (Mininet VM) sends a Hello message to the ODL controller with an IP address of 192.16.20.32 on port 6653 and receives the highest version of supported OF protocol back in the Features packet with the same version of OpenFlow, as seen in *Figure 1.20*.

The screenshot shows two Wireshark captures, one for 'openflow_v4' and one for 'openflow_v4'. Both captures show a sequence of OpenFlow messages exchanged between two hosts at 192.16.20.30 and 192.16.20.32. The first host (192.16.20.30) initiates the negotiation with a 'OFPT_HELLO' message (Frame 646). The second host (192.16.20.32) responds with an 'OFPT_FEATURES_REQUEST' message (Frame 654). Subsequent frames show 'OFPT_FEATURES_REPLY', 'OFPT_BARRIER_REQUEST', 'OFPT_BARRIER_REPLY', 'OFPT_MULTIPART_REQUEST', 'OFPT_MULTIPART_REPLY', and 'OFPT_ROLE_REQUEST' messages. The 'OFPT_HELLO' frame is highlighted with a red border in both captures, and its details pane shows it is a Version 1.3 message with a Transaction ID of 21. The 'OFPT_FEATURES_REQUEST' frame is also highlighted with a red border in the second capture, and its details pane shows it is an Element of type 'OFPHEt_VERSIONBITMAP' (1).

No.	Time	Source	Destination	Protocol	Length	Info
646	189.001898046	192.16.20.30	192.16.20.32	OpenFlow	82	Type: OFPT_HELLO
654	189.164382979	192.16.20.32	192.16.20.30	OpenFlow	90	Type: OFPT_FEATURES_REQUEST
656	189.164817879	192.16.20.30	192.16.20.32	OpenFlow	98	Type: OFPT_FEATURES_REPLY
658	189.213772519	192.16.20.32	192.16.20.30	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
659	189.213904019	192.16.20.30	192.16.20.32	OpenFlow	74	Type: OFPT_BARRIER_REPLY
661	189.479324336	192.16.20.32	192.16.20.30	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, OFP
662	189.479756136	192.16.20.30	192.16.20.32	OpenFlow	1138	Type: OFPT_MULTIPART_REPLY, OFP
672	190.803635317	192.16.20.32	192.16.20.30	OpenFlow	114	Type: OFPT_MULTIPART_REQUEST, OFP
673	190.803663717	192.16.20.32	192.16.20.30	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
674	190.803775918	192.16.20.30	192.16.20.32	OpenFlow	98	Type: OFPT_MULTIPART_REPLY, OFP
675	190.803914618	192.16.20.30	192.16.20.32	OpenFlow	338	Type: OFPT_BARRIER_REPLY
684	191.130511984	192.16.20.32	192.16.20.30	OpenFlow	90	Type: OFPT_ROLE_REQUEST

No.	Time	Source	Destination	Protocol	Length	Info
646	189.001898046	192.16.20.30	192.16.20.32	OpenFlow	82	Type: OFPT_HELLO
654	189.164382979	192.16.20.32	192.16.20.30	OpenFlow	90	Type: OFPT_FEATURES_REQUEST
656	189.164817879	192.16.20.30	192.16.20.32	OpenFlow	98	Type: OFPT_FEATURES_REPLY
658	189.213772519	192.16.20.32	192.16.20.30	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
659	189.213904019	192.16.20.30	192.16.20.32	OpenFlow	74	Type: OFPT_BARRIER_REPLY
661	189.479324336	192.16.20.32	192.16.20.30	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, OFP
662	189.479756136	192.16.20.30	192.16.20.32	OpenFlow	1138	Type: OFPT_MULTIPART_REPLY, OFP
672	190.803635317	192.16.20.32	192.16.20.30	OpenFlow	114	Type: OFPT_MULTIPART_REQUEST, OFP
673	190.803663717	192.16.20.32	192.16.20.30	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
674	190.803775918	192.16.20.30	192.16.20.32	OpenFlow	98	Type: OFPT_MULTIPART_REPLY, OFP
675	190.803914618	192.16.20.30	192.16.20.32	OpenFlow	338	Type: OFPT_BARRIER_REPLY
684	191.130511984	192.16.20.32	192.16.20.30	OpenFlow	90	Type: OFPT_ROLE_REQUEST

Figure 1.20: Negotiation of OpenFlow Version.

When we try to use an unsupported OF version, then the controller will respond with an error message as well as information about what version is supported, as seen in *Figure 1.21*.

No.	Time	Source	Destination	Protocol	Length	Info
136	11.732693441	192.16.20.32	192.16.20.30	OpenFlow	82	Type: OFPT_HELLO
139	11.733882243	192.16.20.30	192.16.20.32	OpenFlow	155	Type: OFPT_ERROR

```

▶ Frame 139: 155 bytes on wire (1240 bits), 155 bytes captured (1240 bits) on interface 0
▶ Ethernet II, Src: Microsoft_02:64:55 (00:15:5d:02:64:55), Dst: Microsoft_02:64:80 (00:15:5d:02:64:80)
▶ Internet Protocol Version 4, Src: 192.16.20.30, Dst: 192.16.20.32
▶ Transmission Control Protocol, Src Port: 56828, Dst Port: 6653, Seq: 17, Ack: 17, Len: 89
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_ERROR (1)
  Length: 89
  Transaction ID: 0
  Type: OFPET_HELLO_FAILED (0)
  Code: OFPHFC_INCOMPATIBLE (0)
  Data: We support version 0x06, you support versions 0x01, 0x04, no common versions.

```

Figure 1.21: OF Version Negotiation Failure.

Traffic to the OF channel doesn't use an OF pipeline, but standard TLS or TCP connection methods, so the switch must distinguish ingress packets as local before matching them against flow tables.

After the controller receives a reply message to the features request from the switch it will identify the DPID of the device where the lower 48 bits are MAC address and 16 bits are defined by implementation such as the VLAN ID as well as a number of supported flow tables and its capabilities, as seen in *Figure 1.22*.

No.	Time	Source	Destination	Protocol	Length	Info
42	3.192506383	192.16.20.32	192.16.20.30	OpenFlow	82	Type: OFPT_HELLO
44	3.194541385	192.16.20.30	192.16.20.32	OpenFlow	82	Type: OFPT_HELLO
46	3.195870287	192.16.20.32	192.16.20.30	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
47	3.196113087	192.16.20.30	192.16.20.32	OpenFlow	98	Type: OFPT_FEATURES_REPLY
48	3.197294088	192.16.20.32	192.16.20.30	OpenFlow	74	Type: OFPT_BARRIER_REQUEST

```

▶ Transmission Control Protocol, Src Port: 58474, Dst Port: 6653, Seq: 17, Ack: 25, Len: 32
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_FEATURES_REPLY (6)
  Length: 32
  Transaction ID: 805
  datapath_id: 0x0000000000000001
  n_buffers: 256
  n_tables: 254
  auxiliary_id: 0
  Pad: 0
  capabilities: 0x0000004f
    .....1..... = OFPC_FLOW_STATS: True
    .....1..... = OFPC_TABLE_STATS: True
    .....1..... = OFPC_PORT_STATS: True
    .....1.... = OFPC_GROUP_STATS: True
    .....0.... = OFPC_IP_REASM: False
    .....1.... = OFPC_QUEUE_STATS: True
    .....0.... = OFPC_PORT_BLOCKED: False
  Reserved: 0x00000000

```

Figure 1.22: Features Reply Message from the Switch.

The switch next sends a multipart request back to the ODL containing information about metering for statistics, port groups, and port description, as seen in *Figure 1.23*.

No.	Time	Source	Destination	Protocol	Length	Info
63	6.345540491	192.16.20.30	192.16.20.32	OpenFlow	1138	Type: OFPT_MULTIPART_REPLY, OFPMP_DESC
64	6.365464407	192.16.20.32	192.16.20.30	OpenFlow	114	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
65	6.365598507	192.16.20.30	192.16.20.32	OpenFlow	98	Type: OFPT_MULTIPART_REPLY, OFPMP_METER_FEATURES
66	6.365720107	192.16.20.30	192.16.20.32	OpenFlow	330	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
► Ethernet II, Src: Microsoft_02:64:80 (00:15:5d:02:64:80), Dst: Microsoft_02:64:55 (00:15:5d:02:64:55)						
► Internet Protocol Version 4, Src: 192.16.20.32, Dst: 192.16.20.30						
► Transmission Control Protocol, Src Port: 6653, Dst Port: 58518, Seq: 49, Ack: 1129, Len: 48						
▼ OpenFlow 1.3						
Version: 1.3 (0x04)						
Type: OFPT_MULTIPART_REQUEST (18)						
Length: 16						
Transaction ID: 1						
Type: OFPMP_METER_FEATURES (11)						
► Flags: 0x0000						
Pad: 00000000						
▼ OpenFlow 1.3						
Version: 1.3 (0x04)						
Type: OFPT_MULTIPART_REQUEST (18)						
Length: 16						
Transaction ID: 2						
Type: OFPMP_GROUP_FEATURES (8)						
► Flags: 0x0000						
Pad: 00000000						
▼ OpenFlow 1.3						
Version: 1.3 (0x04)						
Type: OFPT_MULTIPART_REQUEST (18)						
Length: 16						
Transaction ID: 3						
Type: OFPMP_PORT_DESC (13)						

Figure 1.23: Multipart Reply Message from the Switch.

These multipart messages are used to encapsulate reply or requests as packets, are limited to 64 bytes in size. In port description multipart reply message to OVS, we can distinguish port names, MAC addresses, current link bandwidth and its state which is set to *False* as we didn't send any traffic between the hosts yet, as seen in Figure 1.24.

No.	Time	Source	Destination	Protocol	Length	Info
64	6.365464407	192.16.20.32	192.16.20.30	OpenFlow	114	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
65	6.365598507	192.16.20.30	192.16.20.32	OpenFlow	98	Type: OFPT_MULTIPART_REPLY, OFPMP_METER_FEATURES
66	6.365720107	192.16.20.30	192.16.20.32	OpenFlow	330	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
68	6.371896912	192.16.20.32	192.16.20.30	OpenFlow	90	Type: OFPT_ROLE_REQUEST
Type: OFPT_MULTIPART_REPLY (19)						
Length: 208						
Transaction ID: 3						
Type: OFPMP_PORT_DESC (13)						
► Flags: 0x0000						
Pad: 00000000						
► Port						
▼ Port						
Port no: 1						
Pad: 00000000						
Hw addr: fe:dd:29:17:8c:9a (fe:dd:29:17:8c:9a)						
Pad: 0000						
Name: s1-eth1						
► Config: 0x00000000						
▼ State: 0x00000000						
.....0 = OFPPS_LINK_DOWN: False						
.....0 = OFPPS_BLOCKED: False						
.....0.. = OFPPS_LIVE: False						
▼ Current: 0x000000840						
.....0 = OFPPF_10MB_FD: False						
.....0.. = OFPPF_10MB_HD: False						
.....0... = OFPPF_100MB_FD: False						
.....0... = OFPPF_100MB_HD: False						
.....0.... = OFPPF_1GB_FD: False						
.....0.... = OFPPF_1GB_HD: False						
.....1.... = OFPPF_10_GB_FD: True						
.....0.... = OFPPF_40GB_FD: False						

Figure 1.24: Multipart Port Description Reply Message from the Switch.

Next, when we send a Packet-In message to the controller it compares flow entries starting from *Flow Table 0* and then it sends back a Packet-Out message to the switch after learning

all the MAC addresses with ARP as well as the IP addresses of the hosts in the topology, as seen in *Figure 1.25*.

No.	Time	Source	Destination	Protocol	Length	Info
194	14.248933...	192.16.20.30	192.16.20.32	OpenFlow	74	Type: OFPT_BARRIER_REPLY
192	14.248727...	192.16.20.32	192.16.20.30	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
190	14.024514...	192.16.20.30	192.16.20.32	OpenFlow	150	Type: OFPT_PACKET_IN
188	13.744705...	192.16.20.32	192.16.20.30	OpenFlow	316	Type: OFPT_PACKET_OUT

Reason: OFPP_ACTION (1)

Table ID: 0

Cookie: 0x2b00000000000013

Match

- Type: OFPMT_OXM (1)
- Length: 12
- OXM field
 - Class: OFPXMC_OPENFLOW_BASIC (0x8000)
 - 0000 000 = Field: OFPXMT_OFB_IN_PORT (0)
 -0 = Has mask: False
 - Length: 4
 - Value: 1
 - Pad: 00000000
 - Pad: 0000

Data

- Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
- Address Resolution Protocol (request)
 - Hardware type: Ethernet (1)
 - Protocol type: IPv4 (0x0800)
 - Hardware size: 6
 - Protocol size: 4
 - Opcode: request (1)
 - Sender MAC address: 00:00:00_00:00:01 (00:00:00:00:00:01)
 - Sender IP address: 192.16.20.1
 - Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
 - Target IP address: 192.16.20.2

Figure 1.25: Packet-In Message from Switch to the ODL.

When the switch receives back the Packet-In the form of a Packet-Out message, it will know the protocol used to discover the links was LLDP and that traffic coming in via Port1 from Host1 should be outputted to Port2 as a unicast packet, as seen in *Figure 1.26*.

No.	Time	Source	Destination	Protocol	Length	Info
194	14.248933...	192.16.20.30	192.16.20.32	OpenFlow	74	Type: OFPT_BARRIER_REPLY
192	14.248727...	192.16.20.32	192.16.20.30	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
190	14.024514...	192.16.20.30	192.16.20.32	OpenFlow	150	Type: OFPT_PACKET_IN
188	13.744705...	192.16.20.32	192.16.20.30	OpenFlow	316	Type: OFPT_PACKET_OUT

Length: 125

Transaction ID: 40

Buffer ID: OFP_NO_BUFFER (0xffffffff)

In port: OFPP_CONTROLLER (0xfffffff0)

Actions length: 16

Pad: 000000000000

Action

- Ethernet II, Src: 12:a1:20:7b:4f:fc (12:a1:20:7b:4f:fc), Dst: CayeeCom_00:00:01 (01:23:00:00:00:01)
 - Destination: CayeeCom_00:00:01 (01:23:00:00:00:01)
 - Address: CayeeCom_00:00:01 (01:23:00:00:00:01)
 -0 = LG bit: Globally unique address (factory default)
 -1 = TG bit: Group address (multicast/broadcast)
 - Source: 12:a1:20:7b:4f:fc (12:a1:20:7b:4f:fc)
 - Address: 12:a1:20:7b:4f:fc (12:a1:20:7b:4f:fc)
 -1 = LG bit: Locally administered address (this is NOT the factory default)
 -0 = IG bit: Individual address (unicast)
- Type: 802.1 Link Layer Discovery Protocol (LLDP) (0x88cc)

Link Layer Discovery Protocol

- > Chassis Subtype = MAC address, Id: 00:00:00:00:00:01
- > Port Subtype = Locally assigned, Id: 2
- > Time To Live = 4919 sec
- > System Name = openflow:1
- > Stanford - Unknown (0)
- > Stanford - Unknown (1)
- > End of LLDPDU

Figure 1.26: Packet-Out Message from ODL to the Switch.

1.4. SDN

1.4.1. SDN Considerations

Every vendor has a different definition while Sherwood (2014) from Big Switch explains that network provisioning since 1996 evolved from Telnet to SHH. According to Cisco, SDN is the network programmability and automation (Hardesty, 2017). ONF's definition of SDN states that it's a separation of the network control plane from the forwarding plane where devices can be controlled via the control plane (ONF, 2017).

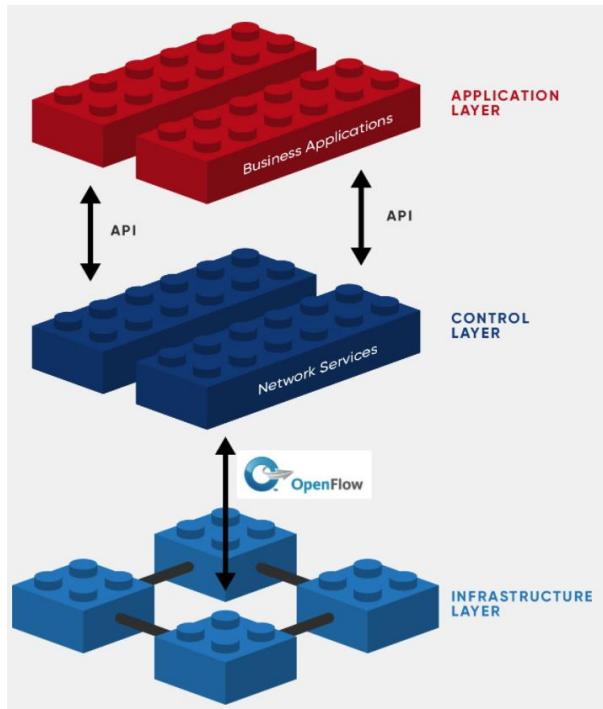


Figure 1.27: SDN Architecture (ONF, 2017).

Application, control and infrastructure layers, where switches reside in infrastructure layer, controllers such as OpenFlow, OpenDaylight (ODL), Ryu, ONOS, Floodlight, POX or HPE's Aruba Virtual Application Networks (VAN) and applications which talk to the controller via Northbound Interface (NBI).

OpenFlow itself isn't an SDN, it's a protocol used in the Southbound Interface (SBI) between the controller and switches. This protocol is only used by some vendors as Cisco, for example, talks about others such as NETCONF, Border Gateway Protocol Link-State (BGP-LS), SNMP or CLI (Hill and Voit, 2015).

VMware (2017) for example, has a product called VMware NSX which overlays a virtual network across the physical network. It allows to quickly deploy networking infrastructure and caters for ease of management, as seen in *Figure 1.28*.

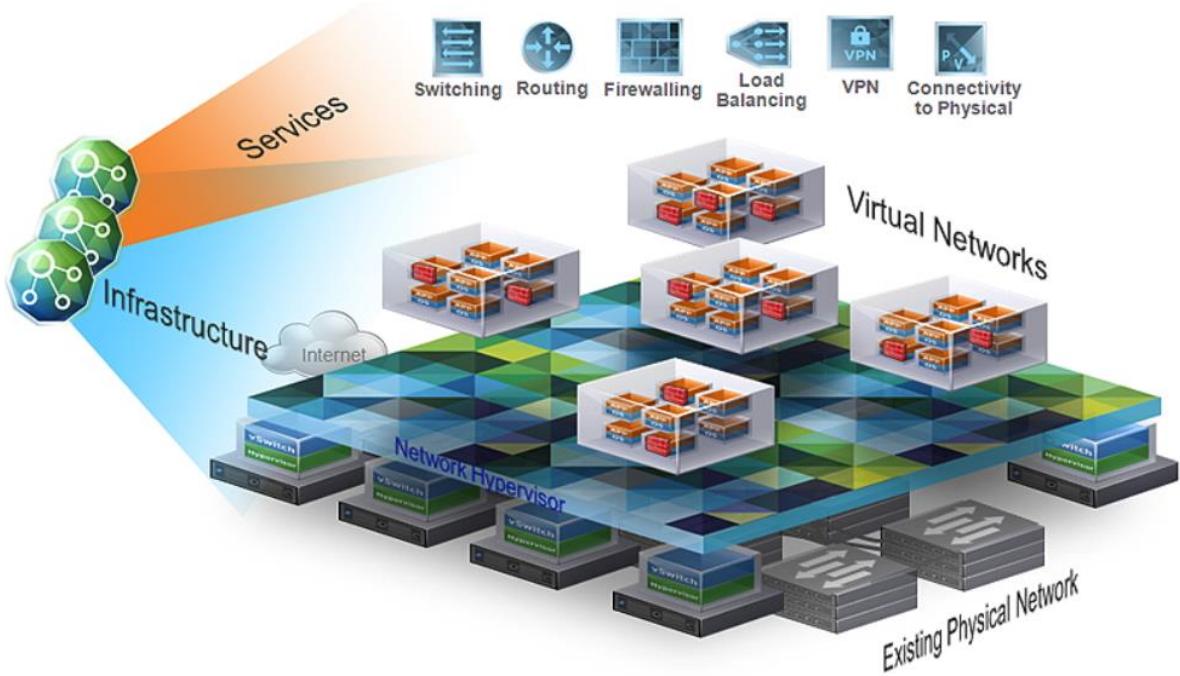


Figure 1.28: VMware NSX (VMware, 2017).

Cisco defines SDN as Software Defined Wide Area Network (SD-WAN) where traffic, rather than use MPLS, would use the centralized controller to send traffic in an intelligent manner to better utilize the links (Miller, 2017) while Cumulus Networks (2017) talk about bare-metal switches and ability to run Linux.

Other people might think that SDN is the open switches, such as produced by Facebook (2016) which allow running open source software.

They are many definitions of SDN, but one important factor is that these terms correlated with programmability and automation of networking infrastructure rather than on hardware itself.

Google (2017) already has implemented SDN as an extension of the public internet to provide performance higher than any router-centric protocol in the same learning from packets streams to provide QoS.

1.4.2. NBI vs SBI

We can notice that ONF (*Figure 1.27*), ODL (*Figure 1.29*) or Cisco's Application Policy Infrastructure Controller Enterprise Module (APIC-EN) (*Figure 1.30*) SDN infrastructures have three things in common.



Carbon: Proliferating Use Cases

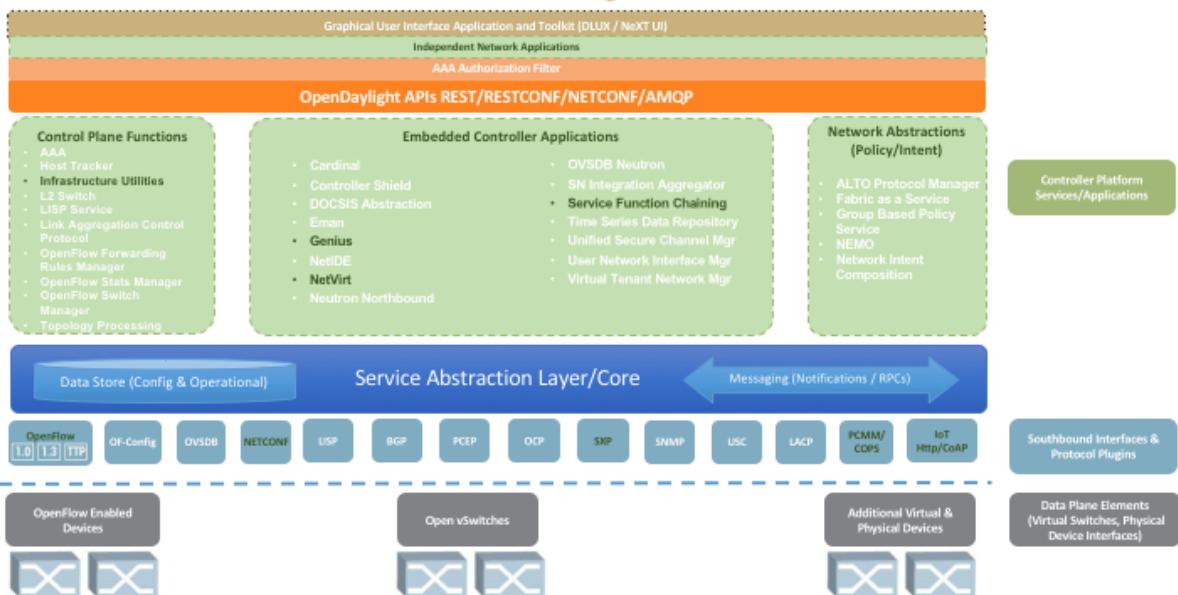


Figure 1.29: ODL Controller Architecture (ODL, 2017).

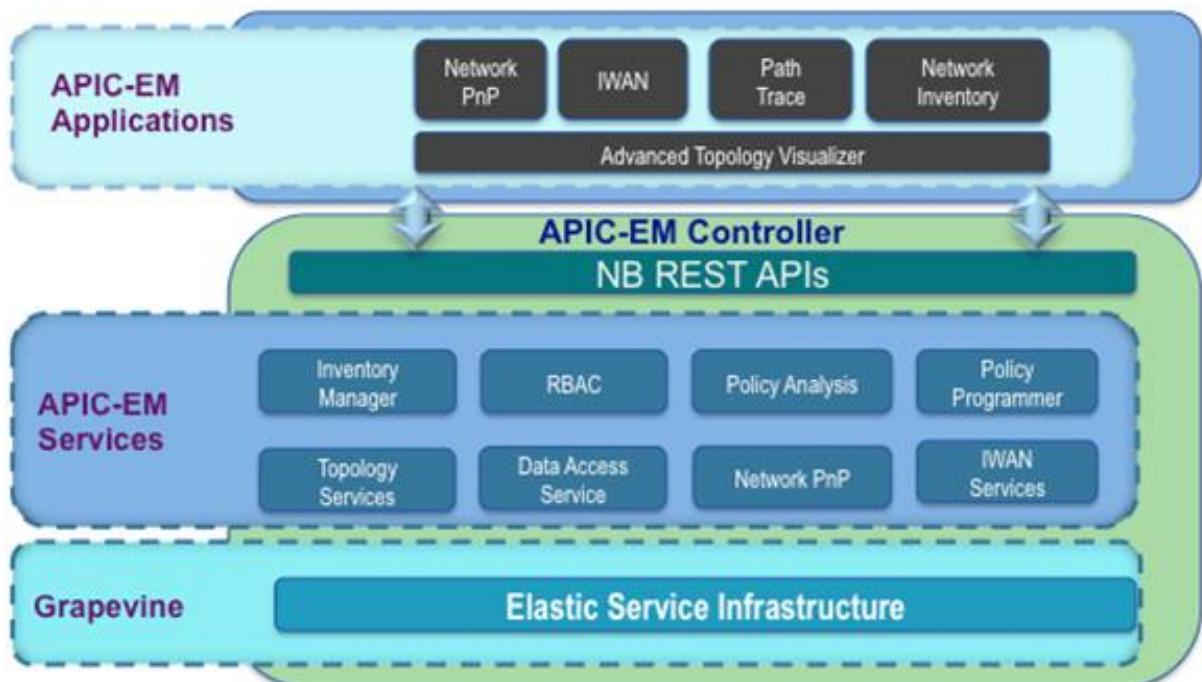


Figure 1.30: Cisco APIC-EN Controller Architecture (Cisco DevNet, 2017)

In Cisco, we can distinguish applications, controllers or services and southbound devices such as network equipment, wherein ODL we can see applications using NBI to communicate with the controller and SBI which talks to network devices. In ONF, applications use NBI to talk to the controller wherein the abstraction layer, our SDN, in turn, communicates to multiple devices via SBI with OpenFlow protocol or any other protocol as discussed by Cisco or ODL.

Cisco uses Representational State Transfer (REST) on Northbound Application Programming Interface (API) to use multiple protocols on SBI where controller abstracts low-level details from the applications, so business apps can be written in high-level languages such as Python to use high-level APIs such as REST to the controller. This hides low-level details from the developer where the user doesn't know the intent. A protocol such as OpenFlow used by ONF only changes the forwarding plane of the network devices, it doesn't configure them. Cisco doesn't even use OpenFlow, however, it was widely developed since the appearance of SDN. SDN can use NETCONF, OpenFlow Configuration (OF-Config) or OVSDB to program the changes in the network device configuration rather than changing the forwarding of the traffic through the device.

In general, SDN assumes the possibility of a global network managed by the logical centralization of control functions, allowing to control many network devices as if they were one element of infrastructure. Flows are controlled at the abstract level of the global network, not associated with individual devices, most often though OpenFlow.

SDN centralized control allows to configure infrastructure from a single management interface and automatically perform reconfiguration to match the changing demands of distributed network applications. Centralized management also allows for more comprehensive and more convenient network monitoring than ever before, despite the increased scale of changes introduced to meet the needs of new services and interconnections.

Two interfaces are required for SDN (Russello, 2016). The NBI allows individual components of the network to communicate with higher level components and vice versa. This interface describes the area of communication between hardware controllers and applications as well as higher layer systems. Its functions focus mainly on the management of automation and the interchange of data between systems.

The SBI is implemented, for example, through OpenFlow. Its main function is to support communications between the SDN controller and network nodes, both physical as well as virtual. It's also responsible for the integration of a distributed network environment. With this interface, devices can discover network Topology (Topo), define network flows and implement API to forward requests from the northbound interface.

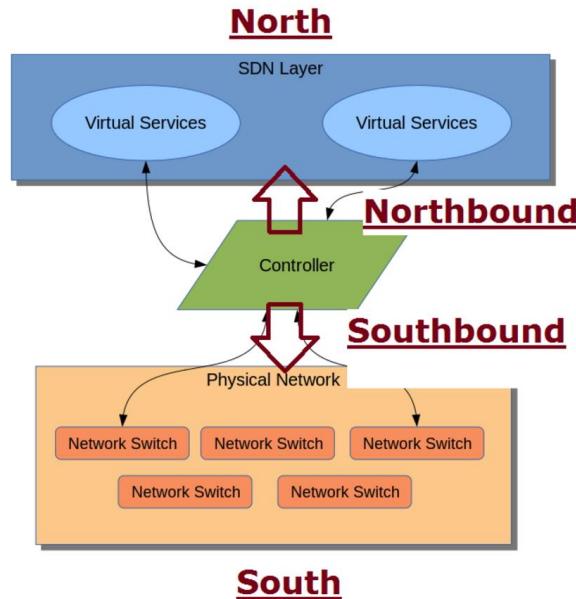


Figure 1.31: SDN Northbound and Southbound Interfaces (Hong, 2014).

In general, SDN separates the control plane from the data plane and provides interfaces and APIs for centralized network management rather than configuring individual distributed devices. There is no exact definition of SDN, so in practice, we are dealing with different types and models for implementation. Depending on the requirements of the case, the choice of architecture may be slightly different.

1.4.3. NFV

According to Chiosi et al. (2012), Network Functions Virtualization (NFV) are functions which are typically available on the HW but deployed as Software (SW) running in a virtual environment.

In the past, we had to purchase a device to deploy an email, database server or firewall, but with hypervisors, we can virtualize the functions on Virtual Machines (VMs). With routers, we can simply implement a Virtual Appliance (VA) to run a router SW rather than to purchase HW router.

Basically, we can run multiple virtual routers such as *Cisco Cloud Services Router (CSR) 1000V* in the same way that we would run multiple virtual servers (Cisco, 2017).

1.4.4. CORD

Central Office Re-architected as a Datacentre (CORD), so rather than the use of traditional office approach where we use NFV and SDN to deploy VMs and VAs in the cloud with an agile approach to provide higher efficiency (OpenCORD, 2017).

This approach was already tested with SDN and ONOS controller which resulted in a

range of advantages for the Service Provider (SP), Subscriber (S) and Third-Party Provider (TPT) as discussed by ONOS (2015).

1.4.5. Available Controllers

There are many Open Source controllers such as Floodlight, LOOM, OpenContrail, ODL, OpenMUL, ONOS, Ryu, POX and Trema. We can also list a major number of commercial controllers developed by Hewlett Packard Enterprise (HPE), Brocade, Dell, Big Switch and many more.

According to ONF ODL, Ryu and ONOS are ahead of the competition in the Open Source field, while in the field of commercial giants we should look at VMware NSX and Cisco APIC-EM (SDN Central, 2016)

In 2016, ONF released a report where many of the examples of OpenFlow deployment are discussed, such as Google, Cornell University, REANZZ, TouIX or Geant and their challenges such as vendor HW compatibility, scaling, incomplete support, inconsistent table access, limited pipeline management, performance impact across different HW vendors (ONF, 2016)

The most important factor is that the SDN controller software must include drivers that will allow controlling the functions of network devices running the system as only then can it act as a network management system. Network performance and reliability are monitored using Simple Network Management Protocol (SNMP) or other standard protocols.

OpenFlow compatible network devices allow to manage the configuration of network topologies, connections and data transfer paths as well as QoS.

It should be noted that most modern switches and routers are equipped with memory modules containing flow tables to control the flow of data at a performance close to the network interfaces. They are used to control packet forwarding in L2, L3, and L4. These tables have a different structure depending on the hardware manufacturer, but there is a basic set of features supported by all network devices as it uses the OpenFlow protocol. Based on this standard, a single controller caters for centralized programming of physical and Virtual Switch (vSwitch) functions on the network.

1.4.6. White-box Switching

In the traditional monolithic approach, we buy everything from one vendor such as Cisco, HP or Juniper, that locks us to the proprietary network stack.

However, now we get HW from one vendor, OS from a different one and apps from the third

vendor. Then we can use a computer environment to visualize OS and apps on any HW.

For example, Google has noticed that they have limitations in networking as they have scaled compute and storage, so they created their own switches and implemented OpenFlow as an SDN (Levy, 2012).

However, Facebook created their own Open Source switches where we can purchase a *Backpack* switch from Cumulus Networks (2017) with their own SDN flavor on Linux or install another Open Source SW. Other companies such as LinkedIn have also created their own switch, called *Pigeon* (Kahn, 2016). Microsoft even offers Linux rather than Windows to visualize network routers and switches in Open Source model called *SONiC* (SONiC Azure GitHub, 2017).

We can also use *Open Network Linux (ONL)* to run on an Open Source switch (Big Switch Networks, 2017) or use *Snappy Ubuntu Core* (Callway, 2015) with *Penguin Arctica* network switches (Penguin Computing, 2017).

This is disaggregation of Network Operating System (NOS), in the past switch and OS were proprietary and integrated, but now we have moved to white-box switches running OS and apps on the top such as: Quagga (2017), SnapRoute (2017), FBOSS (Simpkins, 2015), ICOS (BW-Switch, 2016) or we can get Linux OS from Cumulus Networks, Pica8 (2017) or Big Switch Networks.

For example, we could do something similar to NGI in Italy, who have built their own switch from proprietary HW components with Linux 3.10.55 and SW components such as: OpenVSwitch 2.10, Quagga/Zebra for BGP and OSPF, PPP daemon, DHCP replay, Bidirectional Forwarding Detection (BFD) to ensure bidirectional flow of data on links as well as Nodejs (Bernardi, 2014)

Their vendors were not able to exploit multipath-load balancing and scale beyond 10.000 routers in one *OSPF Area 0* which would force NGI to segment the network into multiple regions which could lead to communications issues for Fixed Wireless Access (FWA).

Nowadays, we have Open Source HW and SW, so we can install NOS on Open HW such as OpenSwitch (2017), Pica8's PicOS or Arista's EOS (Arista, 2017) to swap vendor OS on the switch or run them as VA.

According to Salisbury (2013), many of the commercially available switches in L2 and L3 layers can function as so-called hybrid switches which support both classic switching and packet routing functions as well as commands issued by the OpenFlow controller. This only extends the firmware functionality to NOS (Egmond, 2007) such as Open Network Operating

System (ONOS) which adds an OpenFlow in the form of agent module (English, 2017).

1.4.7. Software Defined WAN

Cisco has a product called Intelligent WAN for Software Defined Wide Area Network which in the past was achieved with Frame Relay and MPLS (SDxCentral, 2017).

SD-WAN technologies are used to control the traffic sent via MPLS networks while at the same time dynamically sending parts of it via Internet cloud rather than using static VPNs and Policy Based Routing (PBR). This way the centralized controller can send low latency apps via MPLS domain where other packets can be sent via the Internet to dynamically forward traffic across different network segments, as seen in *Figure 1.32*.

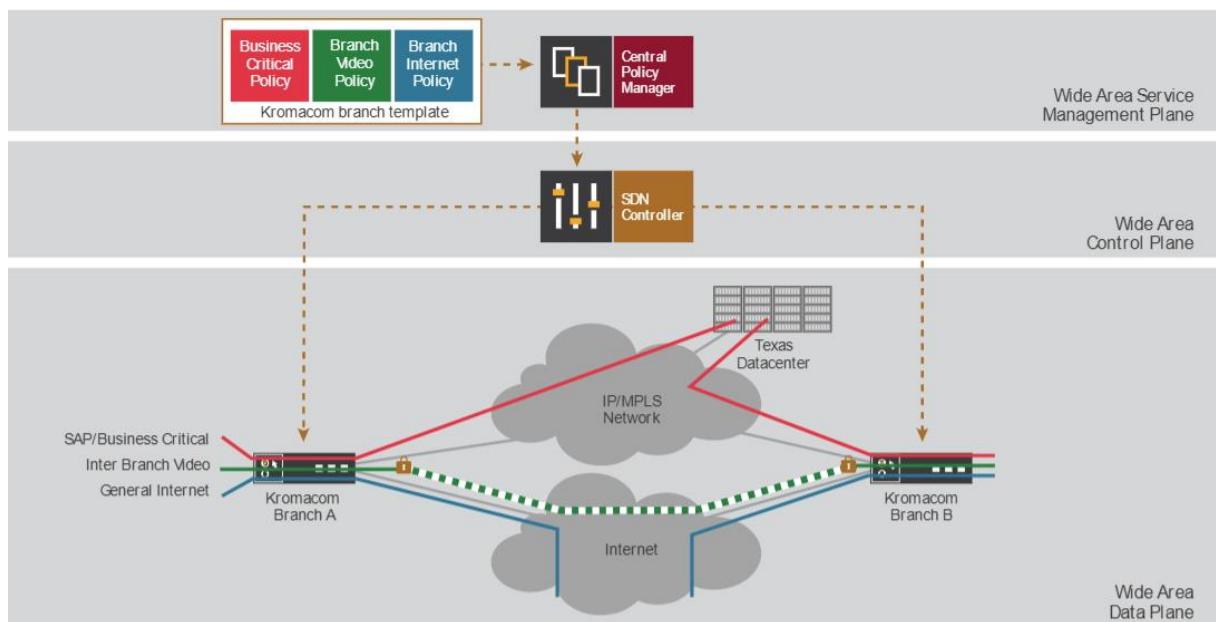


Figure 1.32: Intelligent Offloading of Traffic (Nuage Networks, 2015).

This allows us to more easily manage the forwarding of the traffic which can be controlled via GUI to avoid the use of expensive links.

1.4.8. Advantages and Disadvantages

Burgess (2014) states that centralization is one of the key determinants of the business success of the SDN, because it allows for significant reductions in Operating (Opex) and Capital Expenditure (Capex). In the meantime, the most significant issue of the SDN architecture is how centralized the control plane is and how efficiently each network device can work in this model without the additional control subsystem.

Unfortunately, according to Reber (2015), centralized control plane simplifies architecture, but this approach does not work well in the face of the need for high scalability in real applications. This is particularly important in the context of today's IP networks where the number of

network nodes and endpoints is steadily growing as well as the increase in traffic to be managed by a centralized system.

Therefore, the manufacturers themselves emphasize that in practice we will be dealing with a hybrid model where network topology will be centralized while management of individual data flows will take place inside the network (Hu, 2014)

Since the configurations of the individual flows are very detailed and may also contain parameters of the application layer what could be a potential security risk, any centralized system in a large network could be overloaded with the propagation of millions of these flows. We would have to process them and possibly change their parameters in the event of a connection or device failure. However, if we would centralize only simple control operations such as flow paths propagation between subnets, the SDN concept is likely to work in practice. In the case of server virtualization if we are talking about a small data centre with 100 VMs, the number of new streams to be serviced within the network will increase from 10 to at least 100 thousand. Although network devices can easily handle such quantities, it does involve investing in more efficient models. At the same time, in situations like network failures, the need for new paths and thus the number of streams can dramatically increase. In this situation, the distributed control layer located locally in each switch manages and scales better than a fully centralized system.

Another important factor to be considered before deciding to implement SDN is a delay in packet forwarding (O'Reilly, 2014). For 10 Gigabit per second (Gbps) links they cannot exceed nanoseconds if the performance of this link is to be maintained. After the transfer of flow control in real-time to a central point, the delay may rise to unacceptable values.

1.4.9. Deployment Approaches

According to Vissicchio et al. (2014) a more practical solution, but still allowing for a rather detailed level of control, is the hybrid approach. It assumes an indirect solution in which both the central control point and the local control plane are used in the switches.

The local, distributed control plane is responsible for network virtualization, failover mechanisms and provisioning of new flows. However, some flows are subjected to more thorough analysis of the central point and its reconfiguration. The results are then returned to the switches and subsequent updates are made to them.

Another indirect solution is to use more than one SDN controller depending on network size. In this way, the controllers can be placed closer to the devices that they manage. This leads to shorter delays and allows more efficient control of the work of the switches while transferring

requests to the central control plane.

McNickle (2014) states that SDN can also be deployed using common protocols and interfaces such as Border Gateway Protocol (BGP), Network Configuration Protocol (NETCONF), Extensible Messaging and Presence Protocol (XMPP), Open Virtual Switch Database Management Protocol (OVSDB) and Multiprotocol Label Switching Transport Profile (MPLS-TP) as well as with Command Line Interface (CLI) or SNMP. Below is an overview of each protocol and API used in SDN implementations:

- BGP: used in control plane together with NETCONF for management purpose to interact with devices (Johnson, 2013).
- NETCONF: this protocol is used in routing solutions and it is suitable for large service provider networks (Enns et al., 2011).
- XMPP: this solution has been developed to allow the exchange of nearly real-time structured data between two or more network devices with a focus on availability management.
- MPLS-TP: an extension to MPLS in data plane where control plane is based on the SDN and OpenFlow.
- OVSDB: a protocol where control cluster with managers and controllers which propagate the configuration to the multiple switch databases (Rouse, 2013).
- CLI: each manufacturer has its own implementation of the command line. Most of them have a lot of different CLI interfaces because of acquisitions and mergers between companies. There are tools on the market that create an abstraction layer for CLIs from different manufacturers, but they are very expensive and are only applicable to large service provider networks (Lawson, 2013).
- SNMP: in this case, we are faced with similar challenges as at the command line. In practice, most manufacturers use SNMP only to monitor network devices, not to configure and allocate resources (Mitchell, 2014). This protocol carries various issues around visibility, security and compatibility of Management Information Base (MIB) modules discussed in the by Schoenwaelder (2003) on Internet Architecture Board (IAB) Network Management Workshop.

SDN uses User Datagram Protocol (UDP) tunnels which are very similar to Generic Routing Encapsulation (GRE) tunnels, except that they can be dynamically switched on and off. According to Wang et al. (2017) the effect of using tunnelling is the lack of transparency of

network traffic which entails significant consequences such as serious difficulties in the troubleshooting of network problems. As an example, let's take the user complaining about slow access to the database. In a traditional network, the administrator can quickly determine what is causing it as possible cause might be multiple backup routines executed at the same time. The solution is to set these tasks to execute at different times.

In the SDN environment, the administrator can locate the source and endpoint of a UDP tunnel but will not see key information such as what data is in this tunnel. It will not be able to determine whether the root problem is associated with the replication process, accounting system or mail server. The actual datagrams will be hidden within the UDP tunnel. This means that it is not easy to determine what is causing the problem when users start complaining about slow network performance.

Being aware of this situation, administrators can prepare for it by using network performance management tools that provide information on how the packets physically flow and what rules govern the traffic. There are several solutions developed by companies such as SevOne, Paessler, and ScienceLogic that allow an administrator to keep track of what is happening in the physical network as well as in the SDN tunnels and detect a sudden increase in traffic.

1.4.10. Controller Security

While most SDN systems are relatively new and the technology is at an early stage of development we can be sure that with an increasing number of implementations it will become a target for cybercriminals. We can distinguish several directions of attacks on SDN systems, but the most frequently discussed security issues concern the different layers of the architecture of these systems (Millman, 2015).

Theoretically, a hacker could gain unauthorized physical or virtual network access or break security on an endpoint device connected to the SDN and then try to escalate the attack to destabilize other network elements. This may be, for example, a type of Denial of Service (DoS) attack.

Attackers can also use underlying protocols to add new entries to flow tables by modifying new flows to allow a specific type of network communication that was previously excluded from the network. In this way it's possible to initiate a flow that bypasses the traffic control mechanisms to allow the network communication through the firewall and if it is possible to manage the traffic so that it will be able to pass through preferred network links, then this can be used to capture network traffic and perform Man in the Middle (MITM) attacks. A hacker can also eavesdrop on communication between the controller and network devices to see what

kind of transmission takes place and what kind of traffic is allowed on the network to use this information as reconnaissance.

According to Millman (2015), we should use TLS to authenticate and encrypt communications between network devices and the controller. Using TLS helps to authenticate the controller and network devices which prevents eavesdropping or spoofing of legitimate communications. Depending on the protocols used, there are different ways to protect our data links as some protocols may work inside a TLS session while other may use shared or one-time passwords to prevent attacks. For example, protocols such as SNMPv3 offer a higher level of security than SNMPv2c and Secure Shell (SSH) is much better than Telnet.

Distributed Denial of Service (DDoS) attacks on a controller may cause the device to stop working correctly where exhaustion of the hardware resources available to the controller may influence its operation by extending the response time to incoming packets or failure of permanent response due to device crash.

Hogg (2014) stated that hackers could also run their own controller which network devices can take as a legitimate controller. In this way, an attacker could change the entries in the flow tables of the various network devices and network administrators will no longer have access to these flows from the production perspective. In this case, the attacker will have complete control over the network.

Millman (2015) claims that to avoid unauthorized access to the controller layer we should use mechanisms to implement secure and authenticated administrator access as well as Role-Based Access Control (RBAC) policies and logs to audit changes made by both administrators and unauthorized personnel.

Hogg (2014) also stated that attacks on SDN-specific protocols are another vector of attack due to APIs such as Python, Java, C, REST, XML and JSON which hackers can potentially exploit in terms of the vulnerabilities, and then take control of the SDN via the controller. If the controller does not have any security measures implemented against attacks on APIs, then there is the possibility to create its own SDN rules and thus take control of the SDN environment. Often administrators leave default passwords for Representational State Transfer (REST) APIs that are easy to determine and if during the deployment of SDN these passwords are not changed the attacker is given the ability to send packets to the controller's management interface which in turn enables reading and modifying the SDN configuration.

We can implement measures by using out-of-band security to secure protocols used to manage the controller as well as TLS or SSH encryption to protect communication with the controller

and make sure that data from the application layer to the controller will be encrypted and authenticated (Millman, 2015).

Pickett (2015) talked about Floodlight and ODL controllers' weaknesses on “*DefCon 22 Hacking Conference*” in relation to the use of OpenFlow 1.0 with SDN Toolkit he had developed which exploits security risks (SourceForge, 2016). This, however, doesn't work with the newest releases as they were extensively developed since then to overcome similar issues that was proven by the tests below when we tried to retrieve flows out of controllers with the *of-map.py* app.

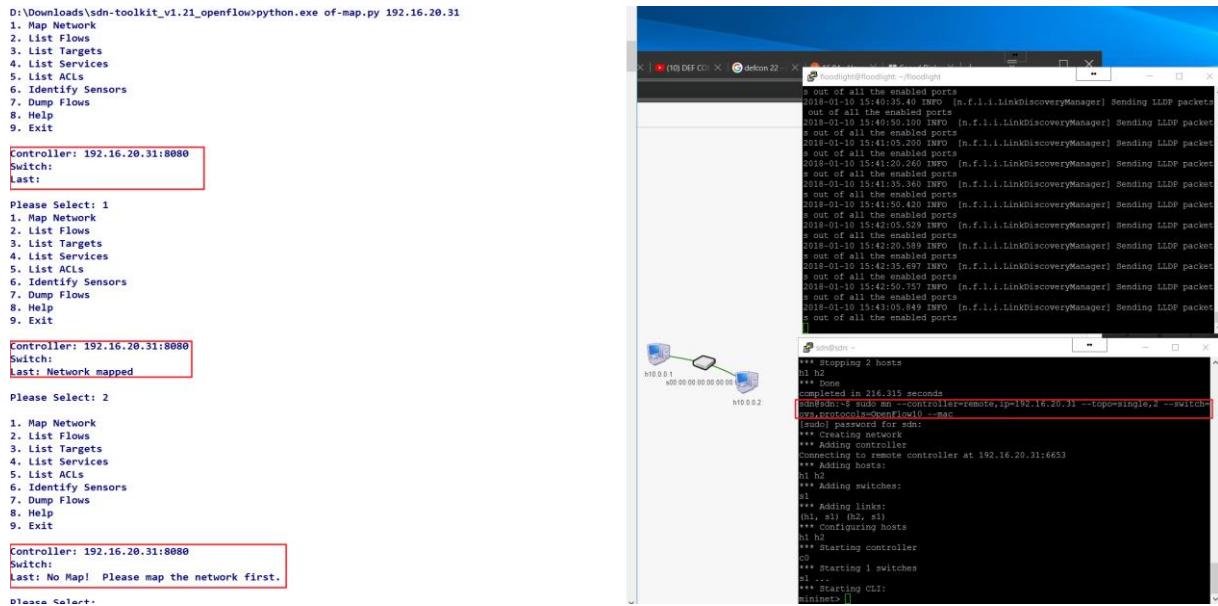


Figure 1.33: SDN-Toolkit Attempt to Download Flows from Floodlight Controller.

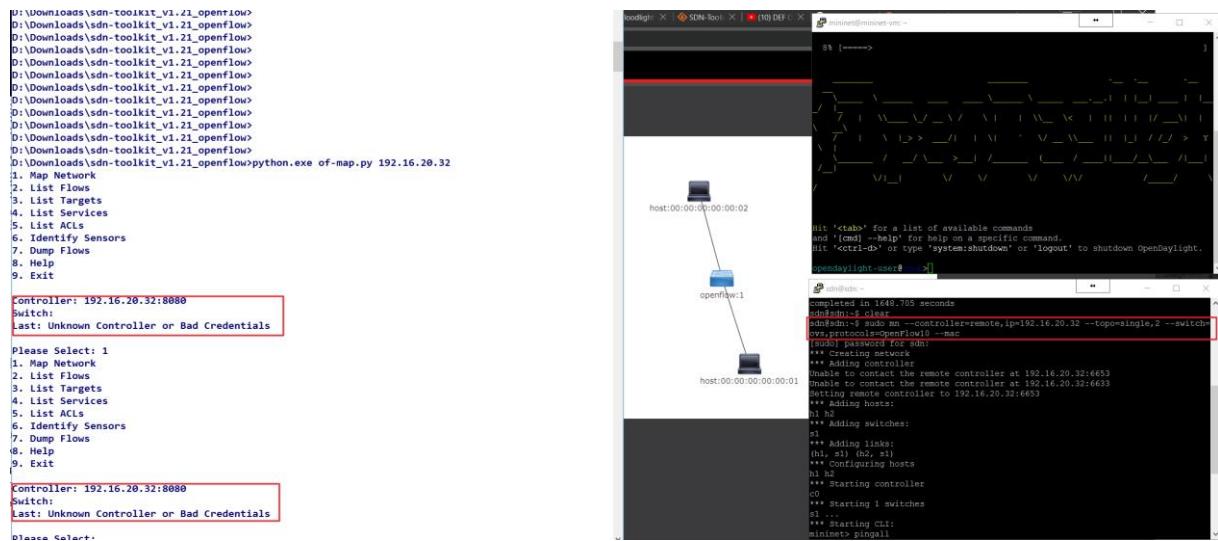


Figure 1.34: SDN-Toolkit Attempt to Download Flows from ODL Controller.

Another example is HPE Aruba VAN controller where we tried to access the GUI which uses a self-signed certificate for TLS with an authentication token which will expire in the

cookie after 24 hours, as seen in *Figure 1.35*.

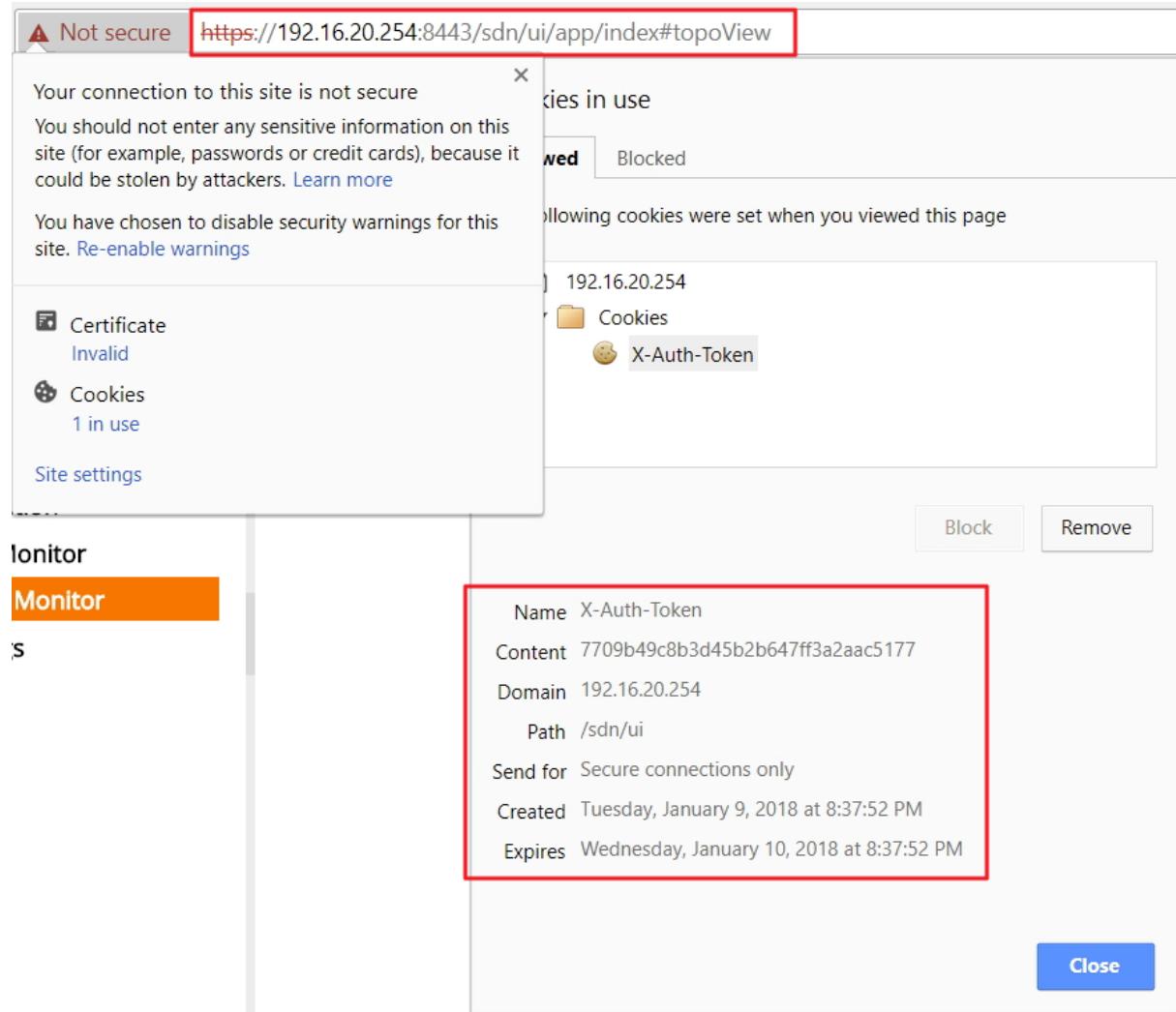


Figure 1.35: Authentication Token with HPE Aruba VAN Controller.

When we tried to access the DPID information via API it fails as we didn't authenticate to the controller, as seen in *Figure 1.36*.

Implementation Notes
List all datapaths that are managed by this controller.

Parameters

Parameter	Value
Try it out!	Hide Response

Request URL
<https://192.16.20.254:8443/sdn/v2.0/of/datapaths>

Response Body

```
{
  "error": "com.hp.api.auth.AuthenticationException",
  "message": "Authentication required"
}
```

Response Code
401

Response Headers

```
date: Tue, 09 Jan 2018 20:51:52 GMT
server: Apache-Coyote/1.1
x-frame-options: deny
access-control-allow-methods: GET, POST, PUT, HEAD, PATCH
content-type: application/json
access-control-allow-origin: *
transfer-encoding: chunked
access-control-allow-headers: Content-Type, Accept, X-Auth-Token
```

Figure 1.36: Failed Authentication with HPE Aruba VAN Controller.

Before authentication to the controller we must obtain the token, so we need to send a username and password via post method to API, as seen in *Figure 1.37*.

login

```
{"login": {"user": "sdn", "password": "skyline"}}
```

Request URL
<https://192.16.20.254:8443/sdn/v2.0/auth>

Response Body

```
{
  "record": {
    "token": "78e372967e7f44bcb89f7c9124afff2f",
    "expiration": 1515535283000,
    "expirationDate": "2018-01-09 22-01-23 +0000",
    "userId": "92a58a84e311491594da55d910843bfc",
    "userName": "sdn",
    "domainId": "6f889c729f094c3d8d1c230ec5af6833",
    "domainName": "sdn",
    "roles": [
      "sdn-user",
      "sdn-admin"
    ]
  }
}
```

Response Code
200

Headers

- Status Code: 200 OK
- Remote Address: 192.16.20.254:8443
- Referrer Policy: no-referrer-when-downgrade

Response Headers

- Access-Control-Allow-Headers: Content-Type, Accept, X-Auth-Token
- Access-Control-Allow-Methods: GET, POST, PUT, HEAD, PATCH
- Access-Control-Allow-Origin: *
- Cache-Control: no-cache, no-store, no-transform, must-revalidate
- Content-Type: application/json
- Date: Tue, 09 Jan 2018 21:14:30 GMT
- Expires: Tue, 09 Jan 2018 21:14:30 GMT
- Server: Apache-Coyote/1.1
- Transfer-Encoding: chunked
- X-FRAME-OPTIONS: deny

Request Headers

- Accept: application/json, text/javascript, */*; q=0.01
- Accept-Encoding: gzip, deflate, br
- Accept-Language: en-US,en;q=0.9
- Connection: keep-alive
- Host: 192.16.20.254:8443
- Referer: https://192.16.20.254:8443/api/
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
- X-Auth-Token: 78e372967e7f44bcb89f7c9124afff2f
- X-Requested-With: XMLHttpRequest

Figure 1.37: Successful Authentication with HPE Aruba VAN Controller.

Since we have authenticated to the controller we are able to retrieve DPID information as well

as other switch specific data such as version, port, and its IP address as well as a number of supported flow tables, as seen in *Figure 1.38*.

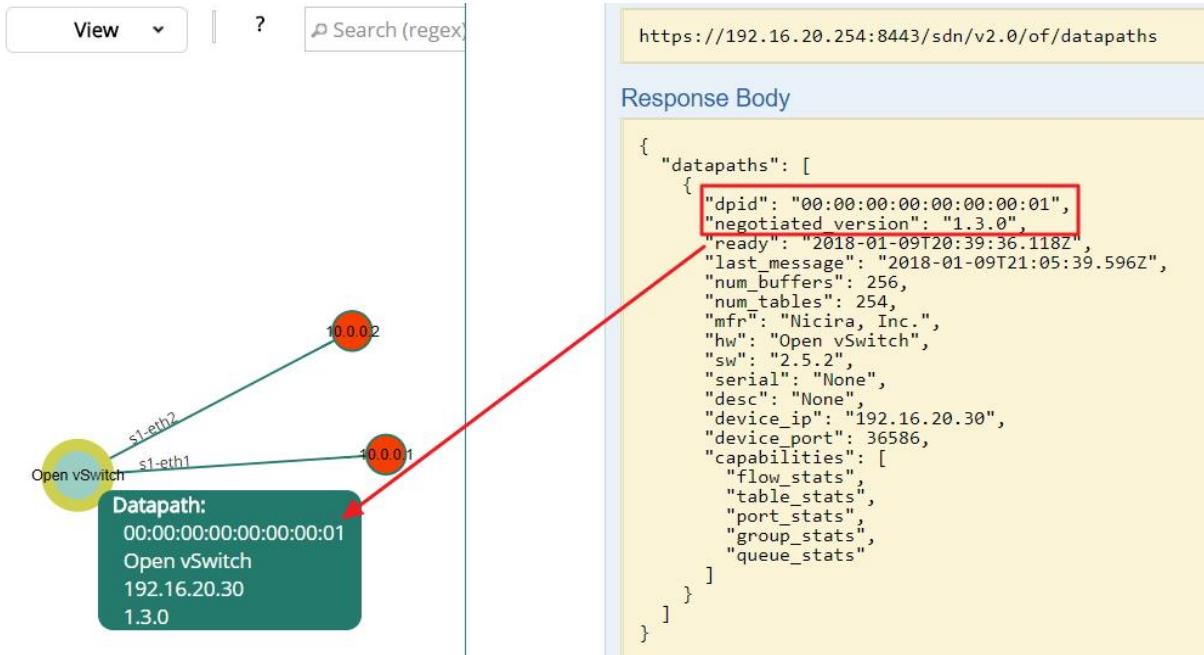


Figure 1.38: DPID Information Retrieval via API with HPE Aruba VAN Controller.

Implementation of TLS is however left to the developer of the controller and as discussed, a use case would be preferable with the use of tokens to access API and Secure Socket Layer (SSL) to encrypt the connection. This is because REST uses HTTP methods to perform create, retrieve, update, delete (CRUD) operations that would allow us not only to retrieve the information but also manipulate the configuration via APIs.

1.4.11. Traditional vs OF Forwarding

In a traditional environment, each switch has its own MAC Address Table, so if PC1 with MAC address *00:00:00:00:00:01* wants to send a packet to PC2 with MAC address *00:00:00:00:00:02* it must send it first to the first switch which will look at the MAC Address Table and if not found an association to PC2 it will flood this through all other ports. In case there are other switches a route to PC2, then another switch will enter the address for PC1 in the MAC Address Table and it will either forward this further or if an association is found to PC2 it will simply forward the packet to a destination, as seen in *Figure 1.39*.

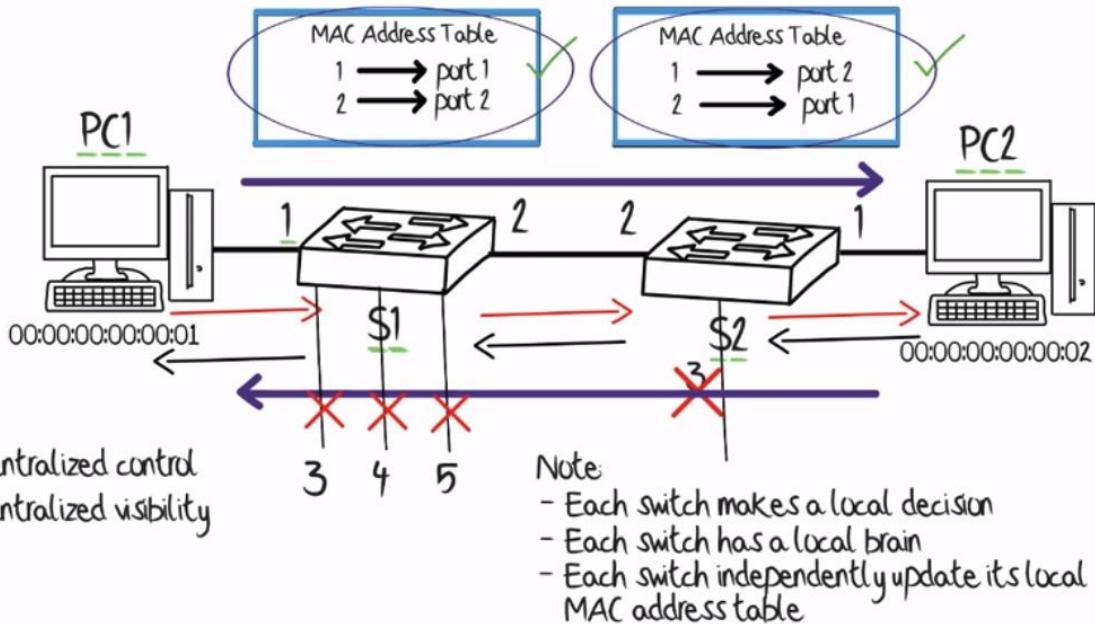


Figure 1.39: Traditional Forwarding (Bombal, 2017).

This will not take into consideration other ports after tables are populated as all decisions are made on each switch independently.

However, in OpenFlow when a packet arrives on the port the switch sends it to the controller as per its Flow Table, the controller, in turn, runs an app which learns all of the MAC addresses and where to forward the specific MAC addresses. The controller would have a table which will contain all list of MAC addresses and their requests from ports coming in. In a situation when the controller doesn't know the location of MAC address of the port to PC2, then it will send a message to the switch to send an ARP request to the remaining ports. When the other switch receives the ARP request it will also forward the frame to the controller and since it will know to which port it belongs, it will send the packet out back to the switch to flood the packet out via remaining ports. PC2 now will know the MAC address of PC1, so it will send a unicast packet to the switch and since OpenFlow messages must be configured in both directions it will send the packet back to the controller, but now it will know the MAC of PC1, so it will update MAC entries and it will update the Flow Table on both switches, as seen in *Figure 1.40*.

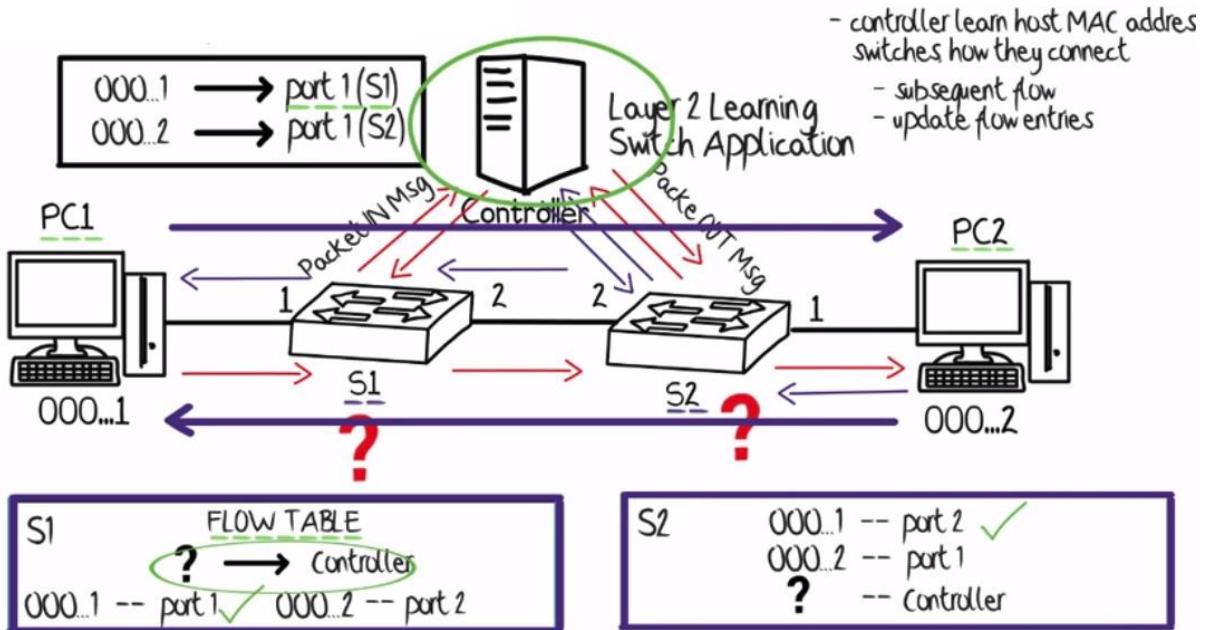


Figure 1.40: OpenFlow Forwarding (Bombal, 2017).

All learning happens on the controller, but as soon as it knows where the devices are, it will update the Flow Tables on the switches, so they will forward the packets and learn independently while only unknown traffic will be sent to the controller.

1.4.12. Reactive vs Proactive

In a reactive flow, the entry controller learns the port and MAC address after an event such as a Packet-In message from the switch while sending a request, as seen in *Figure 1.41*.

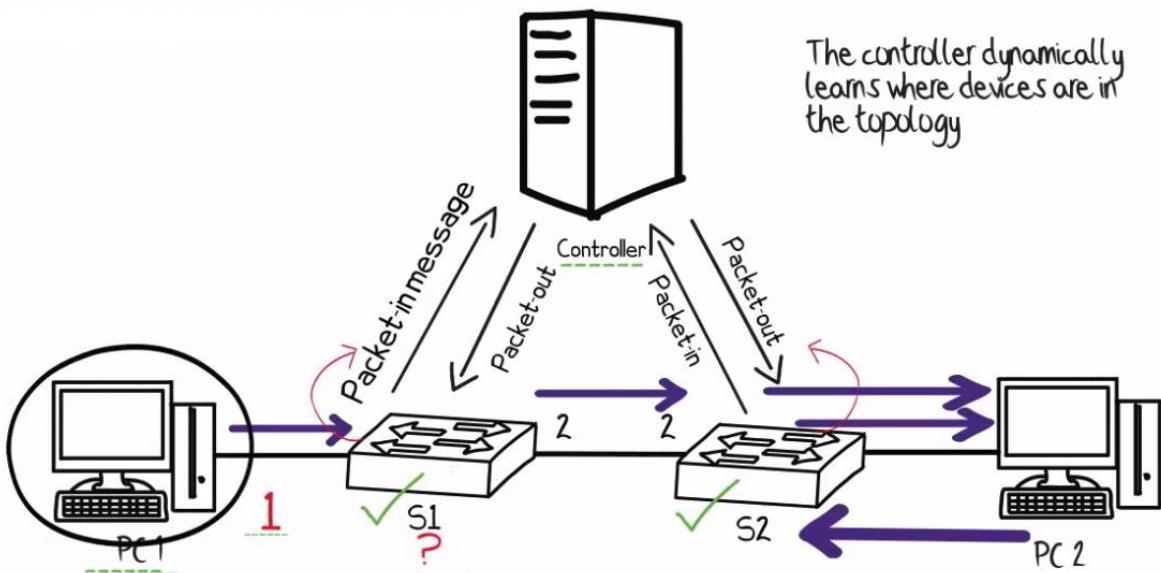


Figure 1.41: Reactive Flow Entry (Bombal, 2017).

Proactive flow entries are pre-programmed rules which exist before any request will be sent as the controller tells the switch what to do, e.g. to drop packets sent by the individual host

or use DSCP markings for QoS for specific devices. Flow entries would contain matches to devices, their actions, and instructions, as seen in *Figure 1.42*.

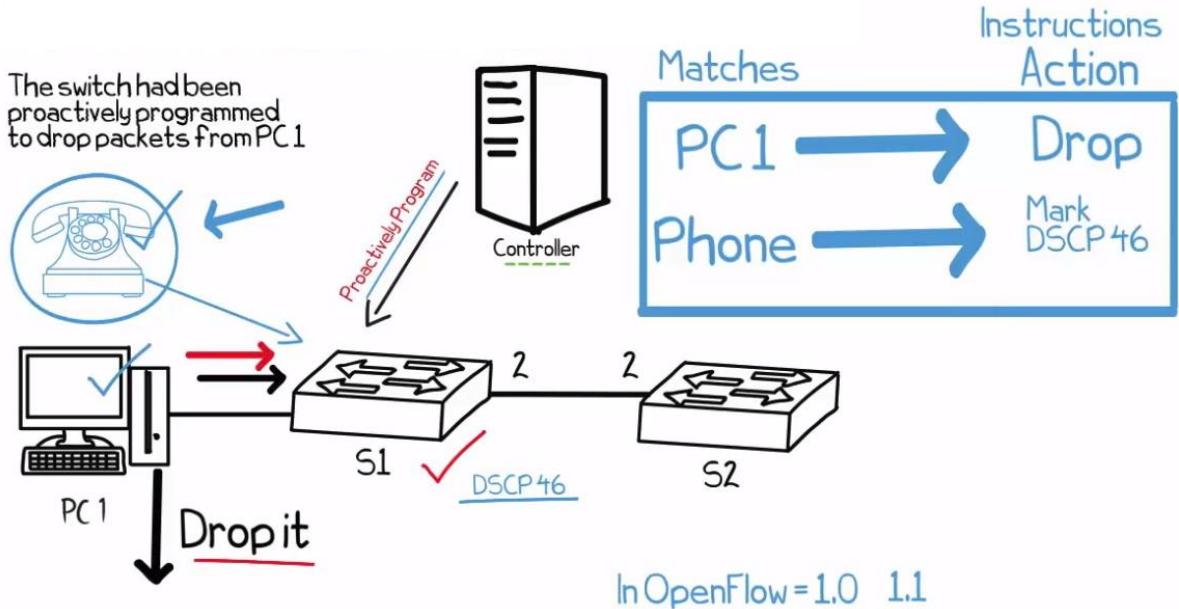


Figure 1.42: Proactive Flow Entry (Bombal, 2017).

We can also program a flow path on the Controller between devices without its presence in the local forwarding of the traffic with the use of Application Specific Integrated Circuits (ASICs) for HW switches or SW, as seen in *Figure 1.43*.

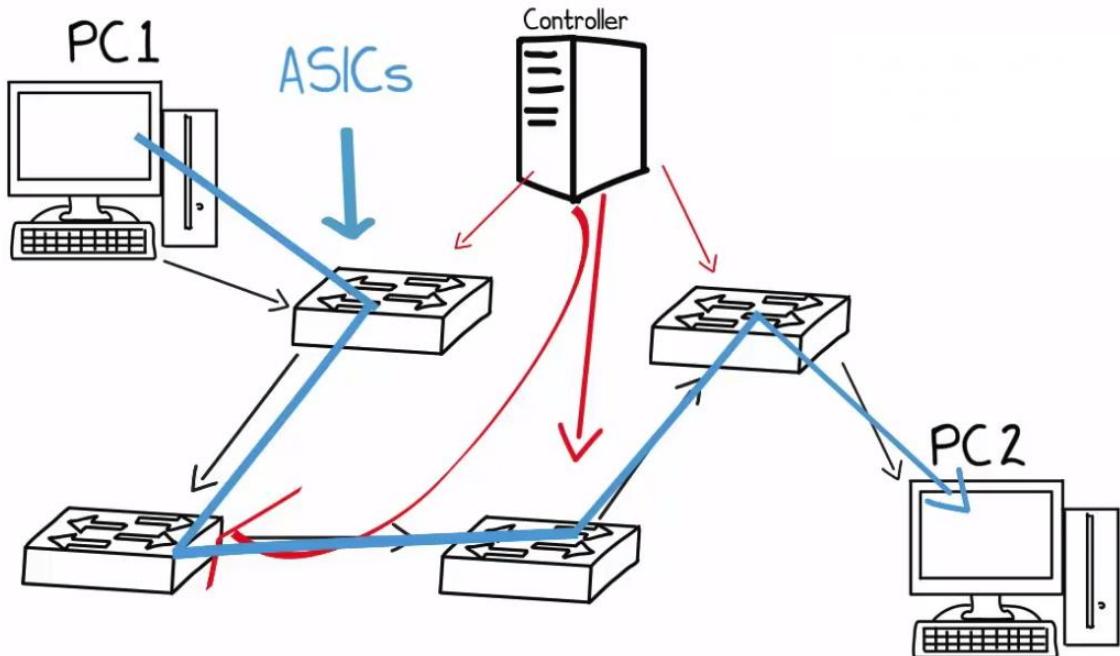


Figure 1.43: Proactive Flow Path (Bombal, 2017).

For reactive entries, the controller should be close to the switch due to decision making,

but for proactive flows, it can be far away as the controller doesn't participate in the forwarding as soon as flow entry is written to the Flow Table.

Chapter 1 introduces the limitations of the IP protocol which can be solved with MPLS and OpenFlow protocols described in the next sections, particularly implementations of MPLS in Linux and in Cisco routers. It also covers OpenFlow definitions and protocol operation which follow Software Defined Networking (SDN) terms and its advantages over traditional networking approaches.

2. Test Environment

All testing of performance, and compatibility of communication using the selected network protocols was performed using physical equipment, such as three Cisco 2801 routers and Hyper-V Virtual Machines (VMs) as guests with Ubuntu OS as well as with the use of a Mininet environment on a Windows Server 2016 host, with 32 Gigabytes (GBs) of Random Access Memory (RAM), an AMD FX 9590 Central Processing Unit (CPU) with 8 cores and four Gigabit Network Interface Cards (NICs).

2.1. Cisco 2801

The Cisco 2801 router is a 2800 series device produced with four types of chassis. The equipment is characterized by high flexibility in a configuration which allows for the delivery of a wide range of services that are commonly used in business solutions. The equipment which we are going to use has three Local Area Network (LAN) ports working in the Fast Ethernet standard and several card slot interfaces allowing to attach additional modules that meet the individual needs of users (Cisco, 2016).

2.1.1. Hardware Routers

For our three “*test-bed*” routers, we have installed 1-Port Fast Ethernet layer 3 (HWIC-1FE) cards and 2-Port Serial Wide Area Network (WAN) interface (HWIC-2T) cards. Routers also can be mounted in the rack as a 1U size unit, because its height is 1.75 inches, as seen in *Figure 2.1*.

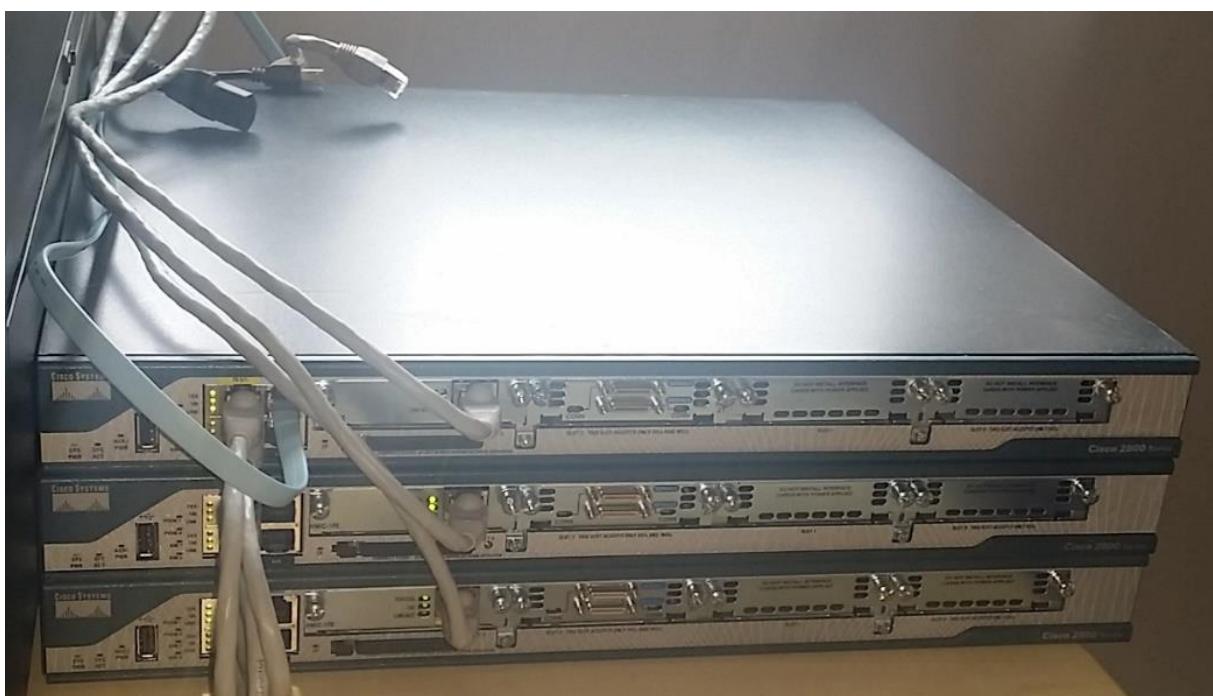


Figure 2.1: Cisco 2800 Series Routers.

These routers have a maximum allowable 384 Megabytes (MB) of Dynamic Random-Access Memory (DRAM) and Compact Flash (CF) memory of 128 MB which allowed us to load IOS released at October 2016 named *2801-adventerprisek9-mz.151-4.M12a* with MPLS support, as seen in *Figure 2.2*.

```
System image file is "flash:c2801-adventerprisek9-mz.151-4.M12a.bin"
Last reload type: Normal Reload

This product contains cryptographic features and is subject to United
States and local country laws governing import, export, transfer and
use. Delivery of Cisco cryptographic products does not imply
third-party authority to import, export, distribute or use encryption.
Importers, exporters, distributors and users are responsible for
compliance with U.S. and local country laws. By using this product you
agree to comply with applicable laws and regulations. If you are unable
to comply with U.S. and local laws, return this product immediately.

Router#show mpls ?
  discovery           Information about LSP discovery
  forwarding-table    Show the Label Forwarding Table
  interfaces          Per-interface MPLS forwarding information
  ip                  MPLS IP information
  l2transport         MPLS circuit transport info
  label               Label information
  ldp                Label Distribution Protocol information
  memory              Memory usage information
  oam                OAM information
  static              Show MPLS static information
  traffic-eng        Traffic engineering information
```

Figure 2.2: Cisco 2801 Router IOS Version.

Nomenclature of IOS version is quite intricate and complicated which is due to many features and packages that create the operating system. *Advanced Enterprise Services* is the full version of the OS containing all available functions which I would require for the configuration of the environment (Cisco, 2017).

2.1.2. Hardware Configuration

To configure the routers, we will use the 64bit Windows Putty client version 0.69 to connect to the IOS via console port (OmniSecu, 2017).

This will require us to use a console cable which can translate instructions via DLL function calls through Universal Serial Bus (USB) rather than an RS-232 connector and to RJ45 port on the console port (eBay, 2017). However, to execute these calls we will also require additional OS drivers for 64bit Windows in version 2.12.26 (FTDI Chip, 2017).

2.2. Traffic Generators

The main scope of this thesis is to check the effectiveness of OpenFlow and MPLS. To investigate how the performance of the network changes within example topologies we will need to generate artificial traffic. For this purpose, we need to use the software traffic generators such as MGEN and iPerf which are made available as freeware software.

2.2.1. MGEN

Multi-Generator (MGEN) is a traffic generator software created and developed by the U.S. authorities called the Naval Research Laboratory (NRL). It is used for testing the performance of computer networks. This tool can generate TCP and UDP traffic, and then allows to save the relevant statistics (U.S. Naval Research Laboratory, 2017). MGEN provides a very wide range of possibilities when it comes to the type of traffic generated, with which we will be able to plan test scenarios that conform to actual conditions in the working environment.

We will use MGEN5 which can be run on Linux, Windows, and MacOS. The program can be downloaded from the archives of the website of the Naval Research Laboratory and it is also available as a package in Ubuntu repositories.

2.2.2. iPerf

This is a tool for testing computer networks created by a group of Distributed Applications Support Team (DAST) of the National Laboratory for Applied Network Research (NLANR). The program is written in C++ and it works in a client-server architecture. For our performance studies, we are going to use generated streams of TCP and UDP traffic to observe the network throughput, jitter and the number of lost packets (iPerf, 2017).

We are going to use version iPerf3 for TCP related tests, iPerf2 for most of UDP scenarios and default buffer sizes. The reason for that is it will support multiple client connections (iPerf, 2017) in conjunction with jPerf2. Both applications are available in the Ubuntu repositories as a package to download and install. There are also versions that work on Linux, Windows, and MacOS. Except for iPerf tools, we are also going to install the previously mentioned Graphical User Interface (GUI) for iPerf2 called jPerf2 as per instructions by Rao (2016).

2.3. Hyper-V

We will not use the bare type 1 hypervisor, but the software where one of the elements is the type 2 hypervisor called Virtual Machine Monitor (VMM) used to run VMs. Hyper-V software which will be hosting the physical machines in the virtual environment. It will allow us to run different operating systems at the same time on one physical server without interference with the existing OS or the need to create independent partitions on a physical disk. Each virtual system has its own virtual hardware and resource allocations.

We must note that the standalone version of Hyper-V Server is completely free, besides the scenario where it is installed as a feature of OS, where it's limited to a number of Virtual Operating System Environments (VOSE) running at the same time on the host (Microsoft,

2017).

We are going to use Windows Server 2016 and Hyper-V version 10.0.14393 as this will allow the building of an unlimited number of VOSE.

2.4. Mininet

Mininet is a system for vitalizing computer networks on a PC (Lantz et al., 2010). Creating and testing of an environment can be done from the CLI. It also has a convenient API which can be explored in detail on GitHub (2017) repository and there is documentation on the Mininet (2017) website. Virtual computers can run any application installed on the system where we can easily start up a Web server in one place and download data from another. This program is intended for the Linux family, but we can also run it on other systems using a VM. A ready to download image is available on the Mininet GitHub (2017) repository that can be started, for example, in VirtualBox (VirtualBox, 2017) which is the easiest and recommended method of installation. The program is also available in the package repositories for Ubuntu distribution (Ubuntu, 2017).

A Mininet uses a lightweight process-based virtualization. For this purpose, network namespaces are used in the Linux kernel since version 2.2.26. They allow us to create logical copies of a network stack that are independent of each other. Each space has its own devices, addresses, routing tables etc. A Mininet for each network element will create a network space, add the appropriate number of Virtual Ethernet (VEth) interfaces and configure them. To enable virtual communication, the devices must be paired and the appropriate namespaces software switches such as Open vSwitch must be started. The Mininet CLI will allow us to execute commands in network spaces that will emulate physical hardware. An identical test environment that the Mininet program will create can be manually compiled using only the CLI through the tedious series of commands. However, with a Mininet this process is significantly accelerated and automated. In addition, it provides an API that facilitates further automation capabilities.

A Mininet is an emulator that is great for exploring and testing the capabilities of the SDN architecture. It features good scalability and quick set up where small networks can start up in seconds. It will allow us to create an “*easy-to-play*” and duplicate environment which is important for the automation of tests when working in larger teams. It works on the actual Linux kernel network stack, so it can be connected easily to physical networks. It is obvious that it does not accurately reproduce the production environment where the migration of the created solution is likely to cause unforeseen problems. Their source may include differences in

supported functionality by a switch that is marked as optional in the OpenFlow standard.

It will help us to create virtual networks via *sudo mn* and to test OpenFlow and SDN. In Mininet, we can use the local OVS controller or connect to the remote controller via *sudo --controller=remote,ip=IP address*. However, if we want to see all of the options we just need to type *sudo mn -h / more*. Since the environment creates phantom processes, sometimes when we build the topology there might be some issues due to an improperly closed process, so it would be advisable to run *sudo mn -c* to remove them.

We can also display devices and their links by typing *net* and *nodes* in Mininet, while *dump* will provide us more information about the devices. To see OVS information we need to execute *sudo ovs-vsctl show* in the Linux box. Ports on the switch can be displayed via *sudo ovs-ofctl -O OpenFlow10* (OF version) *dump-ports s1* (switch number). We can see flow entries on the OVS with the use of *sudo ovs-ofctl -O OpenFlow10* (OF version) *dump-flows s1* (switch number) and to see the flow tables we would need to execute *sudo ovs-ofctl -O OpenFlow10* (OF version) *dump-tables s1* (switch number).

Hosts in Mininet are running Linux so we can use standard Linux CLI, e.g. *h1 ifconfig* to display information about network interfaces. We can also use internal commands such as *link s2 h2 down* to shut down the link between Host1 and Switch1. It's also possible to start a DNS or Web server on a host via *h2 python -m SimpleHTTPServer 80 &*, so we could execute get a request from another host via *h1 wget -O -h1*.

Nodes in the topologies use MAC addresses which are constantly changing so we would normally use *--mac* to keep it simple to read MAC addresses which would correspond to host numbers. There are various topology types available which we are going to discuss in *Chapter 2.6* while testing remote controllers. To see the GUI of apps during SSH terminal session we would need to enable X11 Forwarding and install Xming Server as per instructions by Loria (2010). To change the subnet range *--ipbase* command must be used, e.g. *sudo mn --mac --topo=single,2 --mac --ipbase=192.16.20.0/24*. It's also possible to connect the environment to the outside world via a bridge by adding it to the Linux box which hosts Mininet.

For example, we will add additional vNIC to Mininet VM which is bound to NIC0 of our Hyper-V Host located in the 192.168.2.0/24 network and then create a bridge on OVS to use this interface to connect to the Python Web server running on Host2, see *Appendix 8* for detailed configuration.

From *Figure 2.3* below, we can see the IP addresses of NATSwitch mapped to eth0 and NIC0 to eth1 in Linux VM.

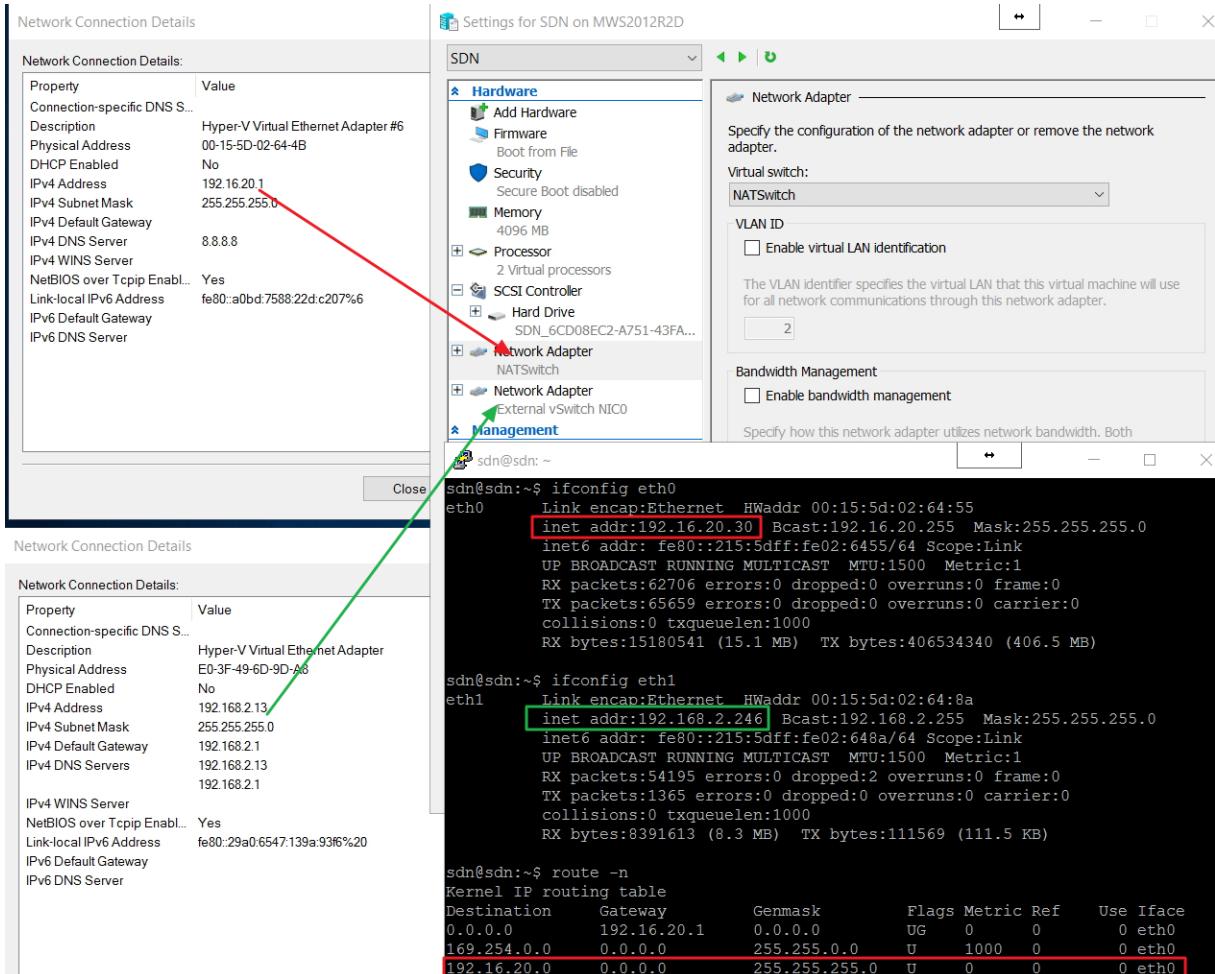


Figure 2.3: Port Mapping between Hyper-V NICs and Mininet VM.

After adding in the bridge to OVS and reconfiguring h2-eth1 IP address to 192.168.2.0/24 network we can get responses to ICMP requests from Host2 in Mininet and from Hyper-V Host back to Mininet as well as the ability to display the web page from a Web server running on that host, as seen in *Figure 2.4*.

Figure 2.4: Communication between Host2 and Hyper-V Host.

Since in our test case we have used an HPE Aruba VAN SDN controller we are able to see other devices in the GUI connected to Switch1 via a bridged eth1 interface which connects to the physical network 192.168.20.0/24, as seen in *Figure 2.5*.

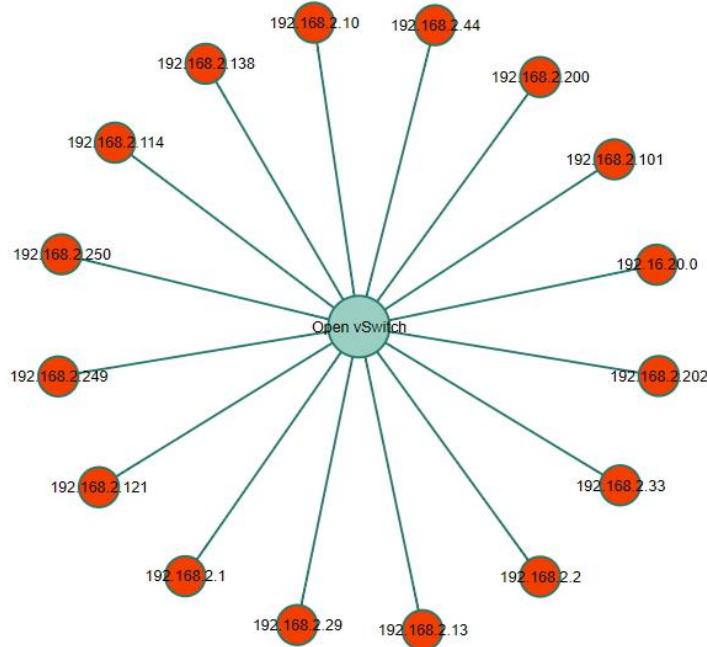


Figure 2.5: View of Devices in External Network via GUI with HPE Controller.

The mininet environment will be used in experiments with OF which will allow us to load specific topologies and connect them to the physical NIC interface via a bridge and

therefore allow access to physical network resources or Hyper-V VMs.

2.5. Wireshark

Wireshark is a graphical network traffic analyser, also known as “*network sniffer*”, which allows capturing packets transmitted over specific network interfaces within ISO OSI layers 2-7 with data frame protocols (Wireshark, 2017).

In addition, it also provides detailed visualization of captured data of many network protocols and it works passively. This means that it does not send any packets, but only intercepts the data on the network interface. It also does not affect the operation of the applications which do send data over the network in any way.

2.6. Controllers with Mininet

In this section, we will investigate different remote controllers and their integration with Mininet via OF13 (Open Networking Foundation, 2013) and OpenFlow version 1.0 (OF10) as well as various Topos and terminology covered in *Chapter 1.3*. The main purpose of this section is to compare various SDN controllers and their features as well as different topologies which can be used to create OF network. Detailed instructions for how to install specific controller distribution can be found in *Appendix 8*.

2.6.1. OpenDaylight

First, we are going to explore the ODL Carbon SR2 release with two switches in the topology, by connecting our Mininet environment where we are going to use MAC for addressing the hosts.

After executing *pingall* in Mininet we can see all the hosts connected to the switches, as seen in *Figure 2.6*.

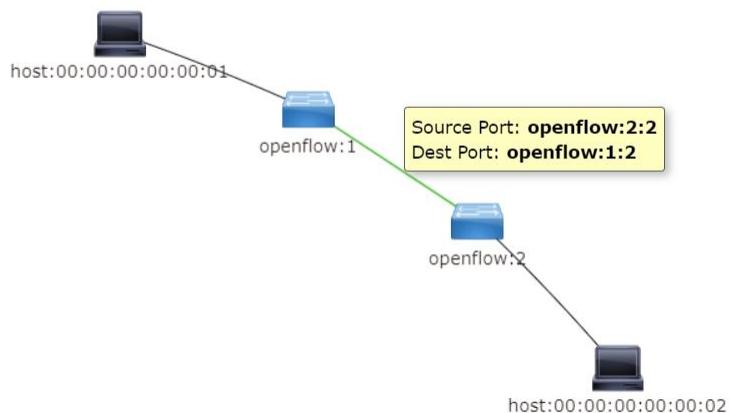


Figure 2.6: Linear Topo with ODL Controller.

When we execute *sudo ovs-vsctl show*, we can see the Flow Tables where switches are

bridged to the remote controller via port 6653 with two Virtual Ports (vPorts) on each switch, as seen in *Figure 2.7*.

Node Id	Node Name	Node Connectors
openflow:1	None	3
openflow:2	None	3

```

sdn@sdn: ~
121 packages can be updated.
4 updates are security updates.

Last login: Wed Dec 27 17:45:17 2017 from 192.16.20.1
sdn@sdn:~$ sudo ovs-vsctl show
[sudo] password for sdn:
f2408541-6908-48b7-bcdd-2843daca80e6
    Bridge "s1"
        Controller "ptcp:6654"
        Controller "tcp:192.16.20.32:6653"
            is_connected: true
            fail mode: secure
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1-eth2"
            Interface "s1-eth2"
        Port "s1"
            Interface "s1"
            type: internal
    Bridge "s2"
        Controller "ptcp:6655"
        Controller "tcp:192.16.20.32:6653"
            is_connected: true
            fail mode: secure
        Port "s2-eth1"
            Interface "s2-eth1"
        Port "s2"
            Interface "s2"
            type: internal
        Port "s2-eth2"
            Interface "s2-eth2"
ovs_version: "2.5.2"

```

Figure 2.7: Flow Table with ODL Controller.

Now, to see the port entries we need to execute `sudo ovs-ofctl -O OpenFlow13 dump-ports s1` or open *Node Connectors* in the GUI which would give us traffic sent on each port of Switch1, as seen in *Figure 2.8*.

Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx
openflow:1:2	421	422	36486	36556	0	0	0
openflow:1:1	19	437	1486	37922	0	0	0
openflow:1:LOCAL	0	0	0	0	0	0	0

```

sdn@sdn: ~
Interface "s2"
    type: internal
Port "s2-eth2"
    Interface "s2-eth2"
    ovs_version: "2.5.2"
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow13 dump-ports s1
OFPST_PORT reply (OF1.3) (xid=0x2): 3 ports
    port LOCAL: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
        tx pkts=0, bytes=0, drop=0, errs=0, coll=0
        duration=2620.958s
    port  1: rx pkts=20, bytes=1556, drop=0, errs=0, frame=0, over=0, crc=0
        tx pkts=589, bytes=50841, drop=0, errs=0, coll=0
        duration=2620.977s
    port  2: rx pkts=571, bytes=49228, drop=0, errs=0, frame=0, over=0, crc=0
        tx pkts=571, bytes=49228, drop=0, errs=0, coll=0
        duration=2620.978s

```

Figure 2.8: Port Entries with ODL Controller of Switch1.

To display flow entries, we would use *sudo ovs-ofctl -O OpenFlow13 dump-flows s1* on the Mininet VM, as seen in *Figure 2.9*.

```

sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x2b00000000000001, duration=2989.310s, table=0, n_packets=599, n_bytes=50915, priority=100, dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x2b00000000000000, duration=2985.577s, table=0, n_packets=29, n_bytes=2363, priority=2, in_port=2 actions=output:1
cookie=0x2b00000000000001, duration=2985.568s, table=0, n_packets=12, n_bytes=840, priority=2, in_port=1 actions=output:2,CONTROLLER:65535
cookie=0x2b00000000000000, duration=2989.310s, table=0, n_packets=5, n_bytes=641, priority=0 actions=drop

```

Figure 2.9: Flow Entries with ODL Controller of Switch1.

From *Figure 2.9*, we can see when looking from the highest to a lower priority that we have four flow entries. First flow entry traffic for *0x88cc* is Link Layer Discovery Protocol (LLDP) which will be sent to the controller, in the second it will be received on Port2 and sent out on Port1, in the third it will arrive on Port1 and then it will be sent to Port2 as well as the controller, while all other traffic by default will be dropped (Network Sorcery, 2012).

We have also installed Cisco OpenFlow Manager (OFM) which we used to input a flow entry on Switch1 which has blocked packets coming out on Port2 with a timeout of 60 seconds, as seen in *Figure 2.10*.

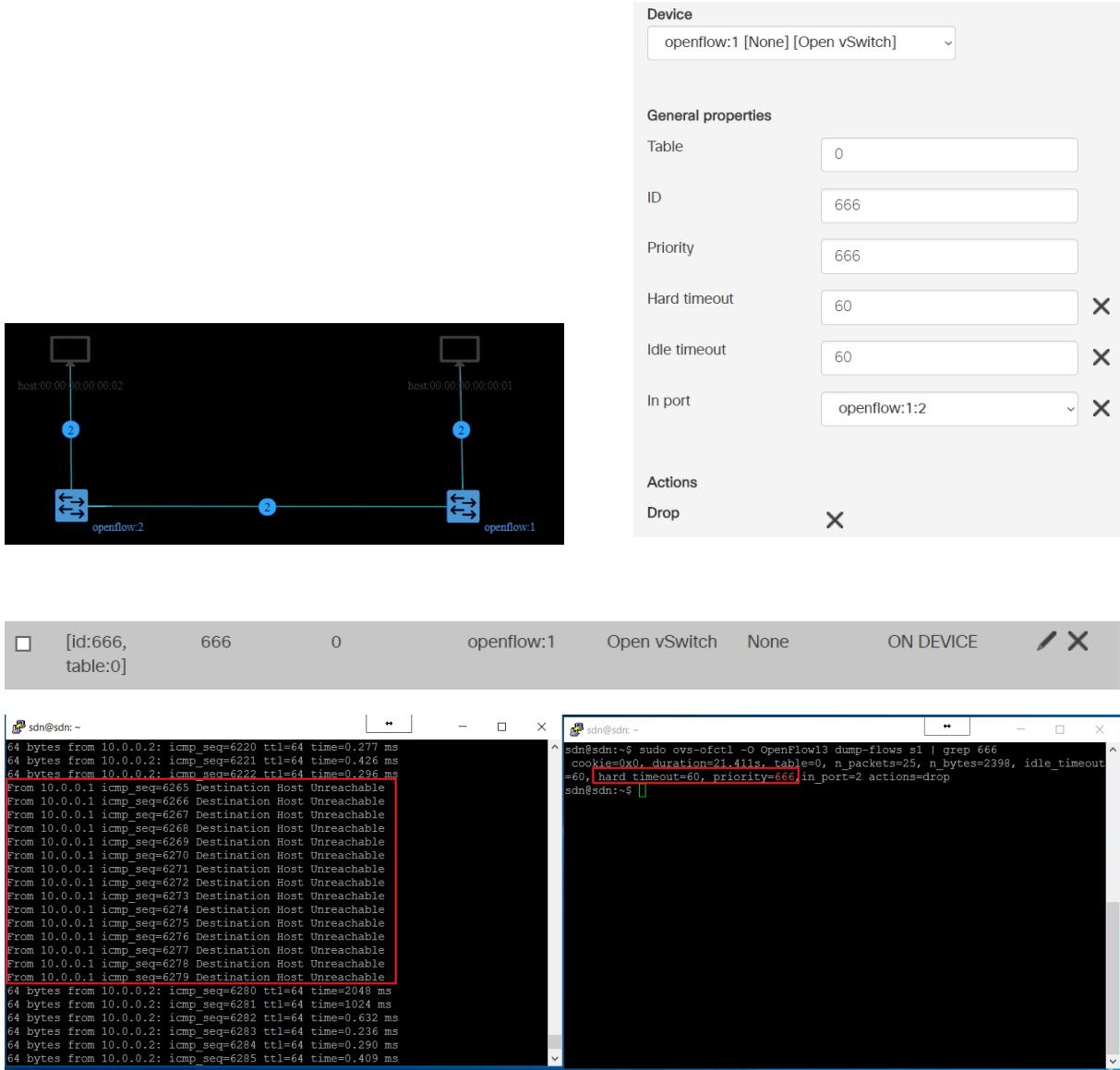


Figure 2.10: OFM Flow Entry to Block Traffic on Switch1.

In the above test case, OFM in the application layer communicates with the controller via the Model-Driven Service Adoption Layer (MD-SAL) to use Extensible Markup Language (XML) YANG models to generate REST APIs, so-called RESTCONF on NBI (Bierman et al., 2014). This, in turn, sends commands to OVS Switch1 with OF13 on the SBI.

2.6.2. Floodlight

A second controller which we are going to look at is the Floodlight version 1.2 where we have decided to build a single switch and three host Topos, as seen in *Figure 2.11*.

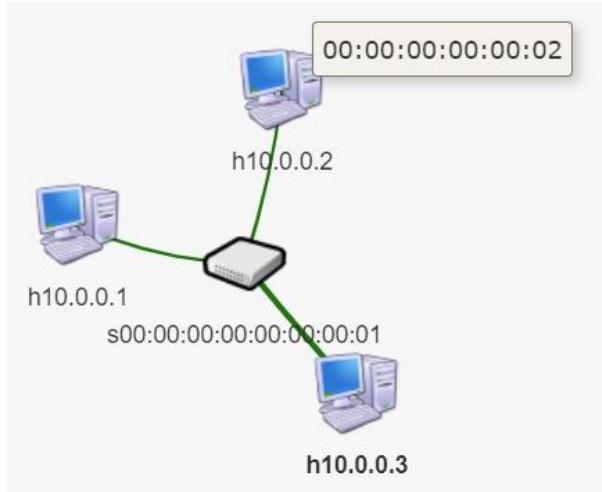


Figure 2.11: Single Topo with Floodlight Controller.

In OVS and GUI we can see that this Topo consists of three vPorts and that communication between Switch1 and the controller also uses port 6653, as seen in *Figure 2.12*.

Hosts

Hosts Connected					
MAC	IPv4 Address	IPv6 Address	Switch	Port	
00:00:00:00:00:01	10.0.0.1	fe80::200:ff:fe00:1	00:00:00:00:00:01	1	
00:00:00:00:00:02	10.0.0.2	fe80::200:ff:fe00:2	00:00:00:00:00:01	2	
00:00:00:00:00:03	10.0.0.3	fe80::200:ff:fe00:3	00:00:00:00:00:01	3	

Showing 1 to 3 of 3 entries

```

sdn@sdn: ~
121 packages can be updated.
4 updates are security updates.

Last login: Thu Dec 28 09:06:44 2017 from 192.16.20.1
sdn@sdn:~$ sudo ovs-vsctl show
[sudo] password for sdn:
f2408541-6908-48b7-bcdd-2843daca80e6
  Bridge "s1"
    Controller "ptcp:6654"
    Controller "tcp:192.16.20.31:6653"
      is_connected: true
      fail_mode: secure
      Port "s1"
        Interface "s1"
          type: internal
      Port "s1-eth2"
        Interface "s1-eth2"
      Port "s1-eth3"
        Interface "s1-eth3"
      Port "s1-eth1"
        Interface "s1-eth1"
  ovs_version: "2.5.2"
sdn@sdn:~$ 

```

Figure 2.12: Flow Table with Floodlight Controller.

However, when we execute *pingall* and view the flow entries of Flow Table 0 we can see that the controller has learnt that if traffic comes from *h10.0.0.1* with MAC address

00:00:00:00:00:01 it will send it to destination MAC address *00:00:00:00:00:02* which is *h10.0.0.2* via Port2, while reversed communication will happen via Port1. Other requests from hosts not in the Topo will be sent to the controller which will learn their MAC and IP address source/destination matches as well as port actions, as seen in *Figure 2.13*

Flow Table										
Show 10 entries										
Table No	Pkt.Count	Byte	Duration(s)	Priority	IdleTimeoutSec	HardTimeoutSec	Flags	Instructions		
0x0	78	7644	79	1	5	0		output=2		
0x0	78	7644	79	1	5	0		output=1		
0x0	33	2342	86	0	0	0		output=controller		


```
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
Open vSwitch (OVS) version 2.9.9-dev (v2.9.9-dev:54495:20200702200000:0:0)
cookie=0x20000028000000, duration=3.475s, table=0, n_packets=1, n_bytes=90, idle_timeout=5, priority=1,ip,in_port=>dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:2
cookie=0x20000028000000, duration=3.475s, table=0, n_packets=1, n_bytes=90, idle_timeout=5, priority=1,ip,in_port=>dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:1
cookie=0x20000028000000, duration=3.485s, table=0, n_packets=1, n_bytes=90, idle_timeout=5, priority=1,ip,in_port=>dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=output:3
cookie=0x20000028000000, duration=3.485s, table=0, n_packets=1, n_bytes=90, idle_timeout=5, priority=1,ip,in_port=>dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=output:1
cookie=0x20000028000000, duration=3.479s, table=0, n_packets=1, n_bytes=90, idle_timeout=5, priority=1,ip,in_port=>dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=output:3
cookie=0x20000028000000, duration=3.479s, table=0, n_packets=1, n_bytes=90, idle_timeout=5, priority=1,ip,in_port=>dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=output:2
sdn@sdn:~$
```

Match	Action
0x0	actions=output:2
0x0	actions=output:1
0x0	actions=output:controller

Figure 2.13: Flow Entries with Floodlight Controller of Switch1.

Normally, OF13 drops all traffic without associated flow entries in the controller, but Floodlight by default forwards all traffic if there is no explicit entry in the Flow Table as in OF10.

2.6.3. HPE Aruba VAN

A third controller which we are going to use was developed by HPE called Aruba VAN SDN in version 2.8.8.0366. This time, to build our Topo with three switches and four hosts, we have used OF10, as seen in *Figure 2.14*.

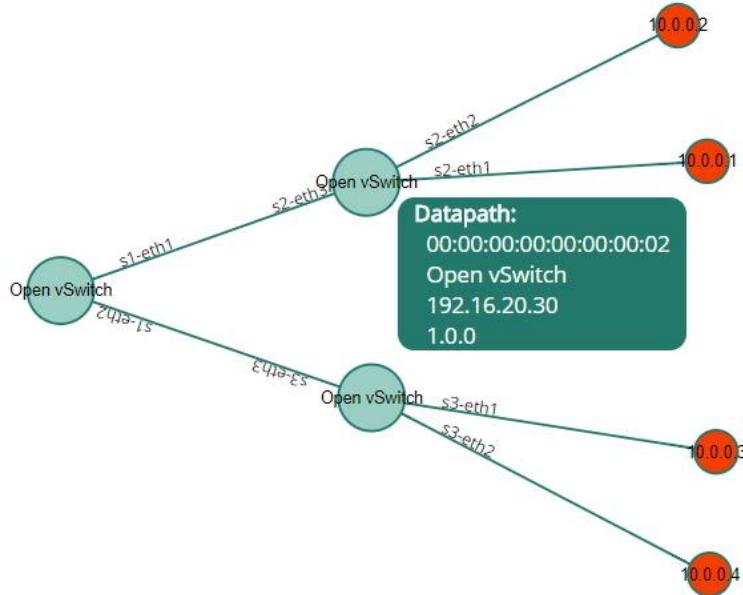


Figure 2.14: Tree Topo with HPE Controller.

The above Topo uses depth of two which means that two links will be connected to the tree

starting at Switch1 and a fanout of two which will add two hosts on each switch at the end of each leaf.

We can also see that Switch1 has two vPorts and other two switches have three vPorts each while communicating. The controller uses port 6633 and Spanning Tree Protocol (STP) on the ports which are working in listening mode, which forwards packets directly to the switch, as seen in *Figure 2.15*.

Ports for Data Path ID: 00:00:00:00:00:00:01				
Port ID	Port Name	H/W Address	State	Current Features
1	s1-eth1	6e:86:75:4d:41:a9	stp_listen	rate_10gb_fd, copper
2	s1-eth2	aa:b1:fe:77:9d:31	stp_listen	rate_10gb_fd, copper
LOCAL	s1	9a:29:4c:c8:4d:c1	link_down, stp_listen	

```

sdn@sdn: ~
f2408541-6908-4b7-bcdd-2843daca80e6
Bridge "s3"
    Controller "tcp:192.16.20.254:6633"
        is_connected: true
    Controller "ptcp:6656"
    fail_mode: secure
    Port "s3-eth3"
        Interface "s3-eth3"
    Port "s3-eth1"
        Interface "s3-eth1"
    Port "s3-eth2"
        Interface "s3-eth2"
    Port "s3"
        Interface "s3"
        type: internal
Bridge "s1"
    Controller "tcp:192.16.20.254:6633"
        is_connected: true
    Controller "ptcp:6654"
    fail_mode: secure
    Port "s1-eth2"
        Interface "s1-eth2"
    Port "s1"
        Interface "s1"
        type: internal
    Port "s1-eth1"
        Interface "s1-eth1"
Bridge "s2"
    Controller "tcp:192.16.20.254:6633"
        is_connected: true
    Controller "ptcp:6655"
    fail_mode: secure
    Port "s2-eth1"
        Interface "s2-eth1"
    Port "s2-eth3"
        Interface "s2-eth3"
    Port "s2-eth2"
        Interface "s2-eth2"
    Port "s2"
        Interface "s2"
        type: internal
ovs version: "2.5.2"

```

Figure 2.15: Port Entries with HPE Controller of Switch1.

During ICMP request we can observe that table miss-matches were using *Table ID 0* which means that packets are forwarded in the traditional way via a MAC Address Table, as seen in *Figure 2.16*.

Table ID	Flow Count	Table Name	Priority	Packets	Bytes	Match	Actions/Instructions	Flow Class ID
0	6		60000	2	142	eth_type: bddp in_port: 3	output: CONTROLLER output: NORMAL	com.hp.sdn.bddp.steal com.hp.sdn.infra.filter
			34000	9	378	eth_type: arp		
			31500	0	0	eth_type: ipv4 ip_proto: udp udp_src: 67 udp_dst: 68	output: CONTROLLER output: NORMAL	com.hp.sdn.dhcp.copy
			31500	0	0	eth_type: ipv4 ip_proto: udp udp_src: 68 udp_dst: 67	output: CONTROLLER output: NORMAL	com.hp.sdn.dhcp.copy
			31000	12	504	eth_type: arp	output: CONTROLLER output: NORMAL	com.hp.sdn.arp.copy
			0	135	13056		output: NORMAL	com.hp.sdn.normal

```

sdn@sdn: ~
sdn@sdn-8 sudo ovs-ofctl -O OpenFlow10 dump-flows s1
[sudo] password for sdn:
NXST_FLOW reply (xid=0x4):
cookie=0xffff0000faaded000, duration=2196.301s, table=0, n_packets=4, n_bytes=284, idle_age=2194, priority=60000, dl_type=0x8999 actions=CONTROLLER:65535
cookie=0xffffd000babadada, duration=2196.301s, table=0, n_packets=0, n_bytes=0, idle_age=2196, priority=31000, arp actions=CONTROLLER:65535, NORMAL
cookie=0xffff900000000000, duration=2194.381s, table=0, n_packets=9, n_bytes=378, idle_age=2188, priority=34000, arp,in_port=1 actions=NORMAL
cookie=0xffff900000000000, duration=2194.379s, table=0, n_packets=9, n_bytes=378, idle_age=2188, priority=34000, arp,in_port=2 actions=NORMAL
cookie=0xffffc000babadada, duration=2196.301s, table=0, n_packets=0, n_bytes=0, idle_age=2196, priority=31500, udp,tp_src=67, tp_dst=68 actions=CONTROLLER:65535,NORMAL
cookie=0xffffc0000babadada, duration=2196.301s, table=0, n_packets=0, n_bytes=0, idle_age=2196, priority=31500, udp,tp_src=68, tp_dst=67 actions=CONTROLLER:65535,NORMAL
cookie=0xfffff000000000000, duration=2196.301s, table=0, n_packets=136, n_bytes=13108, idle_age=147, priority=0 actions=NORMAL
sdn@sdn: ~

```

Figure 2.16: Flow Entries with HPE Controller of Switch1.

Since these matches have the lowest priority of 0, they will be used for all unknown traffic while highest the priority of 60000 is the Broadcast Domain Discovery Protocol (BDDP) which together with LLDP allows discovering the links between devices which do not support OF (Yazici, 2013).

2.6.4. ONOS

Another controller worth of looking at is ONOS Magpie release in version 1.12.0 where we have built a linear Topo of three switches connected to each other in one line and three hosts linked to them individually, as seen in *Figure 2.17*.

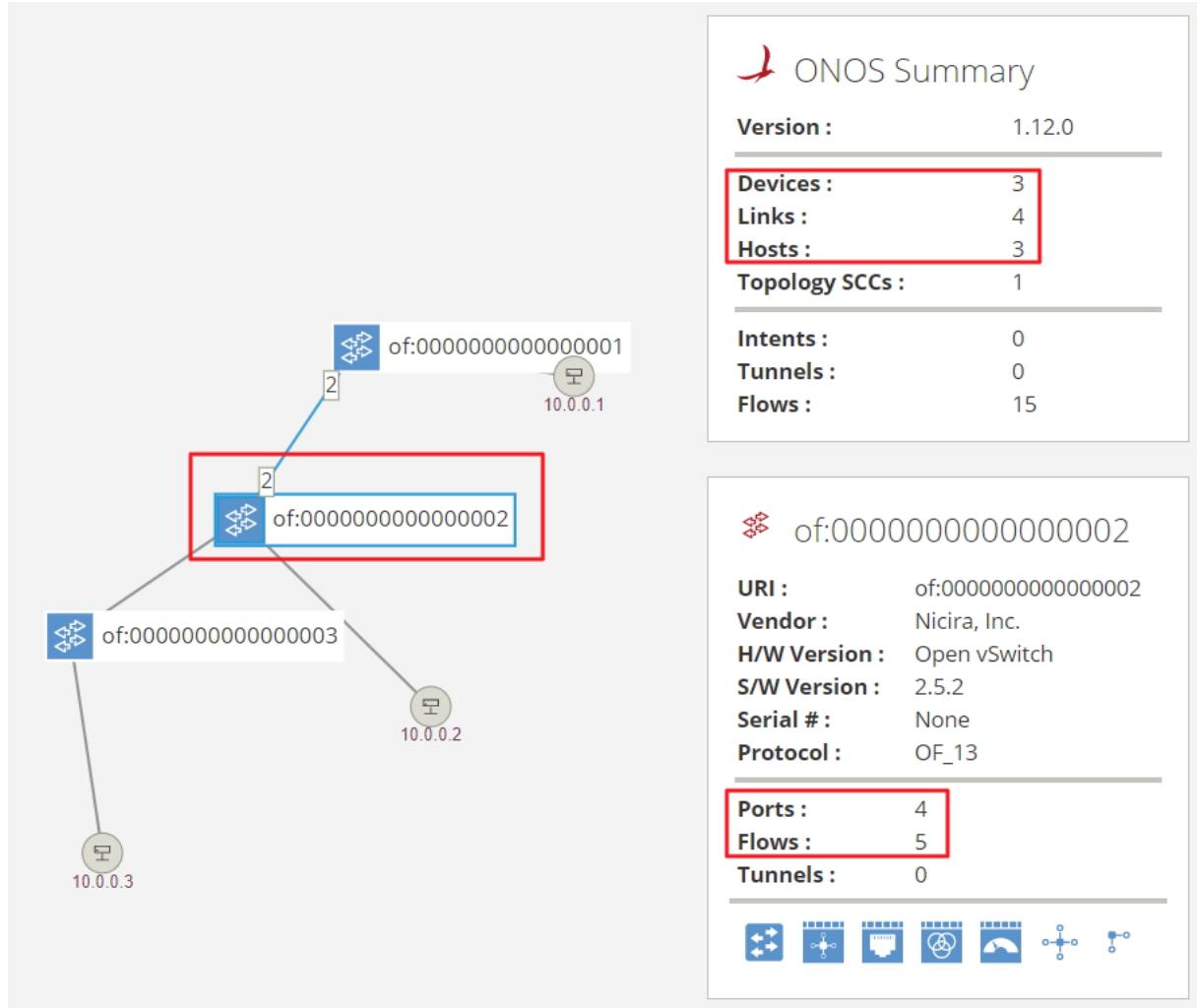


Figure 2.17: Linear Topo with ONOS Controller.

From *Figure 2.17* we can see the Switch2 Datapath ID (DPID), ports and flows as well as details of the whole Topo. Switch2 has four vPorts where one is the local management port (Lo0) which is using OF13 connecting to our OVS controller version 2.5.2 via port 6653, as seen in *Figure 2.18*.

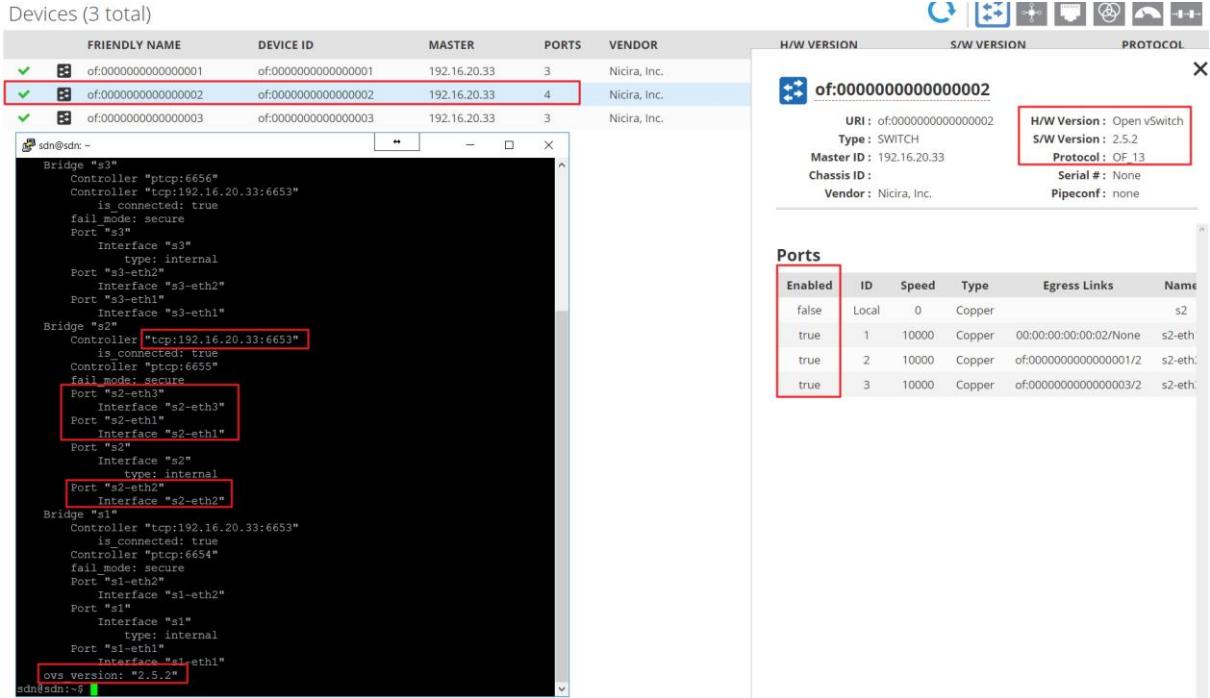


Figure 2.18: Port Entries with ONOS Controller of Switch2.

When we execute a ping between Host1 and Host2 in Mininet via *h1 ping h2* we can see that the forwarding app running in the NBI allows the traffic to go through Port1 of Switch1 and comes out on Port2, then it's forwarded to Port2 of Switch2 to arrive on Port1 before reaching the destination, as seen in *Figure 2.19*.

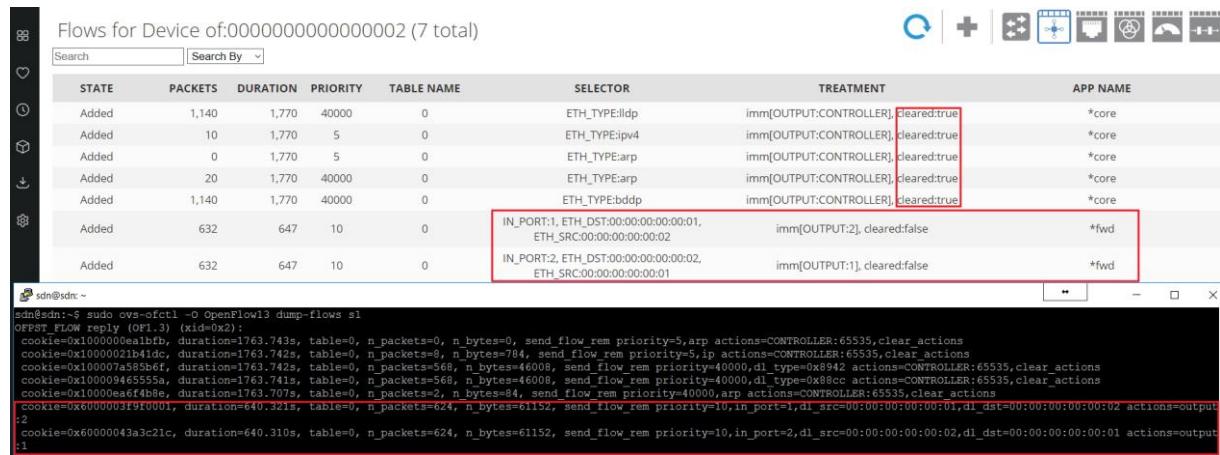


Figure 2.19: Flow Entries with HPE Controller of Switch2.

We can also see that other protocol matches in the Flow Table are set to *clear actions* which means that the controller will refresh any requests directed with: ARP, BDDP, LLDP or IPv4 rather than use the learned flow entries for DPIDs. These actions or packet treatments can be used to clear entries immediately, write or update them to the controller, drop or modify packets, point to another Flow Table or to timeout the flow entries (Open vSwitch, 2016).

2.6.5. POX

POX Eel release in version 0.5.0 only supports OF10 and has no GUI, therefore, we have decided to build a basic Topo with one switch and two hosts with an L2 learning component where the switch learns MAC addresses and installs flows which only match the Flow Table 0. Except for the above component, we also loaded components responsible for readability of the log displayed to the user via `./pox.py log.level --DEBUG samples.pretty_log`.

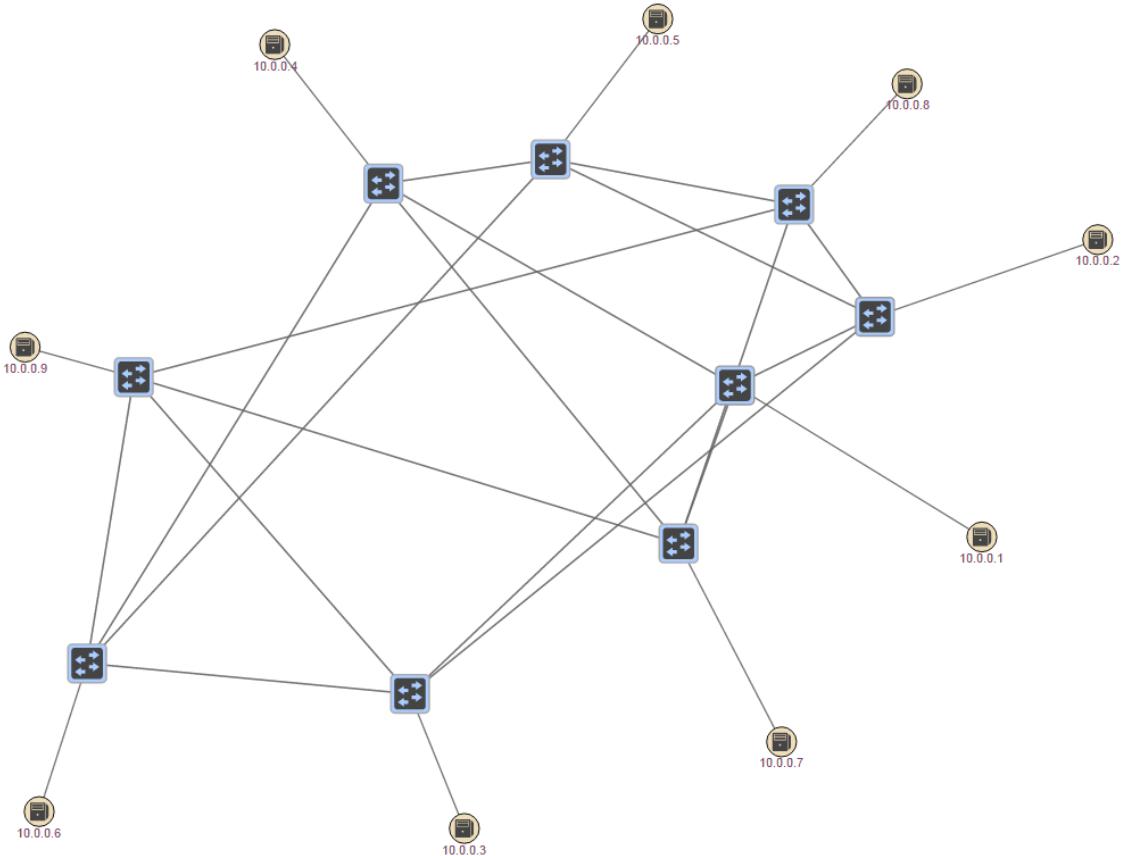
The screenshot shows three terminal windows:

- Terminal 1 (Left):** Shows the command `sudo ovs-ofctl -O OpenFlow10 dump-flows s1` output. It lists several flow entries, all with the same priority (65535), idle timeout (30), and hard timeout (30). The flows are for ARP and ICMP types.
- Terminal 2 (Middle):** Shows the command `mininet> pingall`. It displays ping results between hosts h1, h2, and s1, indicating 0% dropped packets.
- Terminal 3 (Right):** Shows the command `./pox.py log.level --DEBUG samples.pretty_log` running on a host named "POXandRyu". The log output shows the POX version (0.5.0), the platform (Ubuntu 14.04), and various log messages from the learning module as it installs flows for ARP and ICMP requests.

Figure 2.20: Flow Entries with POX Controller of Switch1.

We can see that all priorities and timeout intervals are all the same. This is because they're inserted into the Flow Table 0 via learning module which uses ARP to discover MAC addresses on the two hosts and their ports, then it inserts the flows to forward ICMP packets.

Now we are going to test STP with a mesh topology where each switch will have four links, three to other switches and one to the host. Our IP address range will be 192.16.20.128/25 network, not standard in 10.0.0.0/24, as seen in *Figure 2.21*.



```
sudo mn --controller=remote,ip=192.16.20.34 --
ipbase=192.16.20.128/25 --topo=torus,4,4 --
switch=ovsk,protocols=OpenFlow10 --mac
```

Figure 2.21: Example Torus Topo View from ONOS Controller (Bombal, 2015).

To make sure that we will be able to achieve communication between hosts we must load STP and then disable flooding to prevent loops as well as enable LLDP and the ability of switches to learn from each other, as seen in *Figure 2.22*.

```
./pox.py log.level --DEBUG samples.pretty_log
forwarding.l2_multi openflow.discovery openflow.spanning_tree -
-no-flood --hold-down
```

Figure 2.22: STP POX Controller Command for Torus Topo.

From *Figure 2.23* we can observe that the controller drops LLDP packets to prevent loops in the Topo when sending ICMP requests between h1x1 (192.16.20.129/25 - the first available IP address in the subnet) and h4x4 (192.16.20.144/25 - the last IP address of the host based on 4^n where $n = 4$).

```

sdn@sdn: ~
mininet> h4x1 ping h4x4
PING 192.16.20.144 (192.16.20.144) 56(84) bytes of data.
64 bytes from 192.16.20.144: icmp_seq=1 ttl=64 time=111 ms
64 bytes from 192.16.20.144: icmp_seq=2 ttl=64 time=0.349 ms
64 bytes from 192.16.20.144: icmp_seq=3 ttl=64 time=0.072 ms
^C
--- 192.16.20.144 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
rtt min/avg/max/mdev = 0.072/37.372/111.696/52.555 ms
mininet> h4x4 ping h4x1
PING 192.16.20.129 (192.16.20.129) 56(84) bytes of data.
64 bytes from 192.16.20.129: icmp_seq=1 ttl=64 time=127 ms
64 bytes from 192.16.20.129: icmp_seq=2 ttl=64 time=0.294 ms
64 bytes from 192.16.20.129: icmp_seq=3 ttl=64 time=0.061 ms
^C
--- 192.16.20.129 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2016ms
rtt min/avg/max/mdev = 0.061/42.783/127.995/60.254 ms
mininet>

```

POXandRyu

```

[openflow.discovery] Dropping LLDP packet 1580
[openflow.discovery] Dropping LLDP packet 1582
[openflow.discovery] Dropping LLDP packet 1584
[openflow.discovery] Dropping LLDP packet 1586
[openflow.discovery] Dropping LLDP packet 1579
[openflow.discovery] Dropping LLDP packet 1591
[openflow.discovery] Dropping LLDP packet 1603
[openflow.discovery] Dropping LLDP packet 1580
[openflow.discovery] Dropping LLDP packet 1612
[openflow.discovery] Dropping LLDP packet 1579
[openflow.discovery] Dropping LLDP packet 1580
[openflow.discovery] Dropping LLDP packet 1576
[openflow.discovery] Dropping LLDP packet 1580
[openflow.discovery] Dropping LLDP packet 1592
[openflow.discovery] Dropping LLDP packet 1581
[openflow.discovery] Dropping LLDP packet 1581
[openflow.discovery] Dropping LLDP packet 1590
[openflow.discovery] Dropping LLDP packet 1567
[openflow.discovery] Dropping LLDP packet 1577

```

Figure 2.23: ICMP Packets between Two Hosts with POX Controller for Torus Topo.

2.6.6. Ryu

Another Open Source controller which we will test is called Ryu and it's written in Python which also supports OF10 to OpenFlow version 1.5 (OF15). Other previously discussed controllers such as ODL, Floodlight, HPE Aruba VAN, and POX do not support OF15 or OpenFlow version 1.4 (OF14) except ONOS Magpie.

According to IT technical director Brian Larish from National Security Agency (NSA), OF and SDN operating in their networking infrastructure allows them to save costs for their datacentres and provide an element of predictability and an ease of operations with higher performance due to centralization (Duffy, 2015).

In the first test case, we have tested OF support with five switches and five hosts in a linear Topo, as seen in *Figure 2.24*.

Ryu Topology Viewer

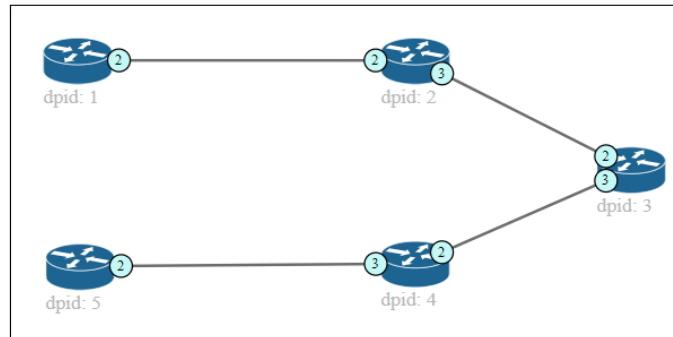


Figure 2.24: Linear Topo with Ryu Controller.

When we first send the ICMP requests via *pingall*, we might not get all the replies as packet sends from the host which are being broadcast out to all ports. This is because the “*Switching Hub*” app needs to learn all MAC addresses of the hosts to create the MAC Address Table later used to transfer packets via a specific port. From our experiments, we can see that Ryu effectively deals with OF10, OF13, and OF14 except for OF15 when we look at the flow

entries of Switch1, as seen in *Figure 2.25*.

```

ryu@Ryu:/usr/local/lib/python2.7/dist-packages/ryu/app
packet in 3 00:00:00:00:00:01 33:33:00:00:00:02 2
packet in 4 00:00:00:00:00:01 33:33:00:00:00:02 2
packet in 5 00:00:00:00:00:01 33:33:00:00:00:02 2
packet in 3 ee:f3:7a:6:b:bf:ab 33:33:00:00:00:02 2
packet in 4 ee:f3:7a:6:b:bf:ab 33:33:00:00:00:02 2
packet in 5 ee:f3:7a:6:b:bf:ab 33:33:00:00:00:02 2
packet in 4 00:00:00:00:00:05 33:33:00:00:00:02 3
packet in 3 00:00:00:00:00:05 33:33:00:00:00:02 3
packet in 2 00:00:00:00:00:05 33:33:00:00:00:02 3
packet in 1 00:00:00:00:00:05 33:33:00:00:00:02 2
[...]
OF10

ryu@Ryu:/usr/local/lib/python2.7/dist-packages/ryu/app
packet in 3 32:6a:0:f:d4:87:fe 33:33:00:00:00:02 3
packet in 2 06:1b:c1:ca:34:d7 33:33:00:00:00:02 2
packet in 2 00:00:00:00:00:01 33:33:00:00:00:02 2
packet in 2 32:6a:0:f:d4:87:fe 33:33:00:00:00:02 3
packet in 3 06:1b:c1:ca:34:d7 33:33:00:00:00:02 2
packet in 1 32:6a:0:f:d4:87:fe 33:33:00:00:00:02 2
packet in 4 06:1b:c1:ca:34:d7 33:33:00:00:00:02 2
packet in 5 06:1b:c1:ca:34:d7 33:33:00:00:00:02 2
packet in 5 00:00:00:00:00:01 33:33:00:00:00:02 2
[...]
OF13

sdn@sdn: ~
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X
h2 -> X h3 h4 h5
h3 -> X h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 40% dropped (12/20 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet> []
sdn@sdn: ~
mininet> reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow10 dump-flows s1
NXST_FLOW reply (xid=0x4):
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow10 dump-flows s1
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=13.875s, table=0, n_packets=15, n_bytes=900, idle_age=1, priority=65535, dl_dst=01:80:c2:00:00:0e, dl_type=0x88cc actions=CONTROLLER:60
cookie=0x0, duration=3.496s, table=0, n_packets=2, n_bytes=196, idle_age=3, in_port=2, dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:01 actions=output:1
cookie=0x0, duration=3.494s, table=0, n_packets=1, n_bytes=98, idle_age=3, in_port=1, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:02 actions=output:2
cookie=0x0, duration=3.472s, table=0, n_packets=2, n_bytes=196, idle_age=3, in_port=2, dl_src=00:00:00:00:03, dl_dst=00:00:00:00:01 actions=output:1
cookie=0x0, duration=3.470s, table=0, n_packets=1, n_bytes=98, idle_age=3, in_port=1, dl_src=00:00:00:00:01, dl_dst=00:00:00:00:03 actions=output:2
cookie=0x0, duration=3.453s, table=0, n_packets=2, n_bytes=196, idle_age=3, in_port=2, dl_src=00:00:00:00:04, dl_dst=00:00:00:00:04 actions=output:1
cookie=0x0, duration=3.451s, table=0, n_packets=1, n_bytes=98, idle_age=3, in_port=1, dl_src=00:00:00:00:01, dl_dst=00:00:00:00:04 actions=output:2
cookie=0x0, duration=3.426s, n_packets=2, n_bytes=196, idle_age=3, in_port=2, dl_src=00:00:00:00:05, dl_dst=00:00:00:00:01 actions=output:1
cookie=0x0, duration=3.420s, table=0, n_packets=1, n_bytes=98, idle_age=3, in_port=1, dl_src=00:00:00:00:01, dl_dst=00:00:00:00:05 actions=output:2
[...]
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
OFPT_FLOW_REPLY(OFL1.3) (xid=0x2):
cookie=0x0, duration=22.449s, table=0, n_packets=14, n_bytes=840, priority=65535
5, dl_dst=01:80:c2:00:00:0e, dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=11.883s, table=0, n_packets=5, n_bytes=378, priority=1, in_port=1, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:04 actions=output:2
cookie=0x0, duration=11.869s, table=0, n_packets=4, n_bytes=336, priority=1, in_port=2, dl_src=00:00:00:00:00:04, dl_dst=00:00:00:00:01 actions=output:1
cookie=0x0, duration=11.814s, table=0, n_packets=5, n_bytes=378, priority=1, in_port=1, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:05 actions=output:2
cookie=0x0, duration=11.798s, table=0, n_packets=4, n_bytes=336, priority=1, in_port=2, dl_src=00:00:00:00:05, dl_dst=00:00:00:00:01 actions=output:1
cookie=0x0, duration=7.515s, table=0, n_packets=4, n_bytes=280, priority=1, in_port=2, dl_src=00:00:00:00:02, dl_dst=00:00:00:00:01 actions=output:1
cookie=0x0, duration=7.509s, table=0, n_packets=3, n_bytes=238, priority=1, in_port=1, dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:02 actions=output:2
cookie=0x0, duration=7.504s, table=0, n_packets=3, n_bytes=280, priority=1, in_port=1, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:03 actions=output:2
cookie=0x0, duration=7.496s, table=0, n_packets=4, n_bytes=280, priority=1, in_port=2, dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:01 actions=output:1
cookie=0x0, duration=7.490s, table=0, n_packets=5, n_bytes=336, priority=1, in_port=2, dl_src=00:00:00:00:00:03, dl_dst=00:00:00:00:01 actions=output:1
cookie=0x0, duration=22.452s, table=0, n_packets=21, n_bytes=1311, priority=0 actions=CONTROLLER:65535
sdn@sdn:~$ 
```



```

ryu@Ryu:/usr/local/lib/python2.7/dist-packages/ryu/app
packet in 5 00:00:00:00:05 ff:ff:ff:ff:ff:ff 1
packet in 4 72:af:b0:e5:4a:01 33:33:00:00:00:fb 3
packet in 2 36:39:41:0d:a9:07 33:33:00:00:00:fb 2
packet in 5 ee:bc:7a:47:60:38 33:33:00:00:00:fb 2
packet in 1 b6:05:79:a6:a8:f7 33:33:00:00:00:fb 2
packet in 3 a2:85:c1:31:16:ee 33:33:00:00:00:fb 2
packet in 2 76:95:e5:62:bd:23 33:33:00:00:00:fb 3
packet in 3 06:05:93:b9:bd:50 33:33:00:00:00:fb 3
packet in 5 00:00:00:00:05 ff:ff:ff:ff:ff:ff 1
packet in 4 e2:9b:a8:6e:e6:79 33:33:00:00:00:fb 2
packet in 3 a2:85:c1:31:16:ee 33:33:00:00:00:02 2
[...]
OF14

ryu@Ryu:/usr/local/lib/python2.7/dist-packages/ryu/app
packet in 5 00:00:00:00:05 ff:ff:ff:ff:ff:ff 1
packet in 4 72:af:b0:e5:4a:01 33:33:00:00:00:fb 3
packet in 2 36:39:41:0d:a9:07 33:33:00:00:00:fb 2
packet in 5 ee:bc:7a:47:60:38 33:33:00:00:00:fb 2
packet in 1 b6:05:79:a6:a8:f7 33:33:00:00:00:fb 2
packet in 3 a2:85:c1:31:16:ee 33:33:00:00:00:fb 2
packet in 2 76:95:e5:62:bd:23 33:33:00:00:00:fb 3
packet in 3 06:05:93:b9:bd:50 33:33:00:00:00:fb 3
packet in 5 00:00:00:00:05 ff:ff:ff:ff:ff:ff 1
packet in 4 e2:9b:a8:6e:e6:79 33:33:00:00:00:fb 2
packet in 3 a2:85:c1:31:16:ee 33:33:00:00:00:02 2
[...]
OF15

sdn@sdn: ~
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 X
h2 -> h1 h3 h4 X
h3 -> h1 h2 h4 X
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 10% dropped (18/20 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet> []
sdn@sdn: ~
mininet> reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow14 dump-flows s1
NXST_FLOW reply (xid=0x4):
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow14 dump-flows s1
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=58.001s, table=0, n_packets=65, n_bytes=3900, priority=65535
35, dl_dst=01:80:c2:00:00:0e, dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=21.564s, table=0, n_packets=5, n_bytes=434, priority=1, in_port=2, dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:01 actions=output:1
cookie=0x0, duration=21.562s, table=0, n_packets=4, n_bytes=336, priority=1, in_port=1, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:02 actions=output:2
cookie=0x0, duration=21.556s, table=0, n_packets=5, n_bytes=378, priority=1, in_port=1, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:03 actions=output:2
cookie=0x0, duration=21.544s, table=0, n_packets=5, n_bytes=378, priority=1, in_port=2, dl_src=00:00:00:00:00:03, dl_dst=00:00:00:00:01 actions=output:1
cookie=0x0, duration=19.497s, table=0, n_packets=5, n_bytes=434, priority=1, in_port=2, dl_src=00:00:00:00:00:04, dl_dst=00:00:00:00:01 actions=output:1
cookie=0x0, duration=19.495s, table=0, n_packets=4, n_bytes=336, priority=1, in_port=1, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:04 actions=output:2
cookie=0x0, duration=13.303s, table=0, n_packets=4, n_bytes=336, priority=1, in_port=1, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:05 actions=output:2
cookie=0x0, duration=13.281s, table=0, n_packets=3, n_bytes=238, priority=1, in_port=2, dl_src=00:00:00:00:00:05, dl_dst=00:00:00:00:01 actions=output:1
cookie=0x0, duration=58.003s, table=0, n_packets=46, n_bytes=2882, priority=0 actions=CONTROLLER:65535
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow15 dump-flows s1
OFPT_FLOW_REPLY(OFL1.5) (xid=0x2):
cookie=0x0, duration=88.349s, table=0, n_packets=31, n_bytes=1730, priority=0 actions=CONTROLLER:65535
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow15 dump-flows s1
OFPT_FLOW_REPLY(OFL1.5) (xid=0x2):
cookie=0x0, duration=88.349s, table=0, n_packets=31, n_bytes=1730, priority=0 actions=CONTROLLER:65535
sdn@sdn:~$ 
```

Figure 2.25: Flow Entries with Ryu Controller of Switch1 with different OF Versions.

According to Yusuke (2016), the reason behind all ICMP packets being dropped for OP15 is that the OVS structure of packets coming out is different than expected by the Ryu controller (Rohilla, 2016).

Figure 2.26: OF15 and OVS Packet-Out Message issue with Ryu Controller.

In a second test case, we have decided to build a datacentre Topo which will use OF13 for communication on SBI and the “*Spanning Tree*” app on NBI. This topology will consist of 16 hosts placed on 4 racks and 6 switches (2 in core and 1 on each rack) with multiple links for redundancy, as seen in *Figure 2.27*.

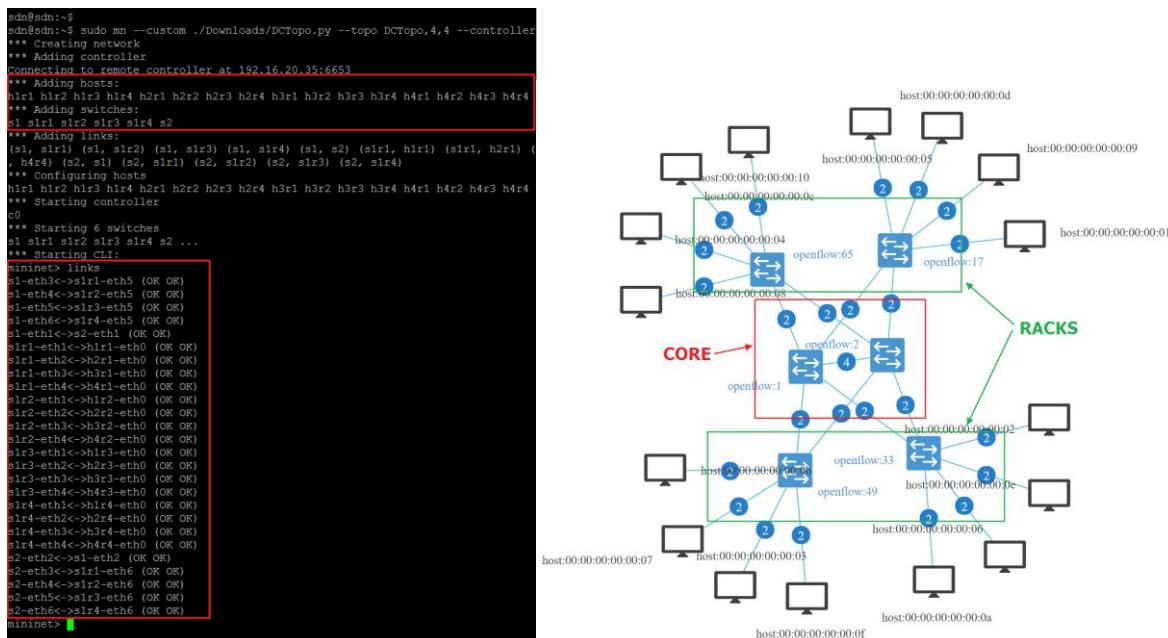


Figure 2.27: Datacentre Topo.4,4 in Mininet with Ryu Controller and in OFM.

We need to wait until the controller learns all of the topologies while Bridge Protocol Data Units (BPDUs) are exchanged between the switches to set up the operational mode of the ports, as seen in *Figure 2.28*.

```

[STP] [INFO] dpid=000000000000000041: [port=1] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000041: [port=2] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000041: [port=3] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000041: [port=4] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000041: [port=5] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000041: [port=6] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000011: [port=1] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000011: [port=2] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000011: [port=3] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000011: [port=4] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000011: [port=5] ROOT_PORT / FORWARD
[STP] [INFO] dpid=000000000000000011: [port=6] NON_DESIGNATED_PORT / BLOCK
[STP] [INFO] dpid=000000000000000002: [port=1] NON_DESIGNATED_PORT / BLOCK
[STP] [INFO] dpid=000000000000000002: [port=2] NON_DESIGNATED_PORT / BLOCK
[STP] [INFO] dpid=000000000000000002: [port=3] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000002: [port=4] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000002: [port=5] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000002: [port=6] ROOT_PORT / FORWARD
[STP] [INFO] dpid=000000000000000031: [port=1] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000031: [port=2] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000031: [port=3] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000031: [port=4] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000031: [port=5] ROOT_PORT / FORWARD
[STP] [INFO] dpid=000000000000000031: [port=6] NON_DESIGNATED_PORT / BLOCK
[STP] [INFO] dpid=000000000000000001: [port=1] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=2] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=3] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=4] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=5] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=6] ROOT_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=1] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=2] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=3] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=4] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=5] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=6] ROOT_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=1] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=2] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=3] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=4] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=5] DESIGNATED_PORT / FORWARD
[STP] [INFO] dpid=000000000000000001: [port=6] NON_DESIGNATED_PORT / BLOCK

```

Figure 2.28: DPIDs and BPDUs Exchange between Ryu Controller and Switches.

From *Figure 2.29* we can see that traffic for h1r1 (MAC address 00:00:00:00:00:01) is output via Port3 while for h1r2 (MAC address 00:00:00:00:00:02) via Port4, while there are three ingress port entries in the controller for each host connected to Switch1. This switch also has six links where Port1 must be in the disabled state as it's not in the Flow Table.

```

sdn@sdn:~$ sudo ovs-vsctl show | grep s1
[sudo] password for sdn:
    Bridge "s1"
        Port "s1"
            Interface "s1"
                Port "s1-eth1"
                    Interface "s1-eth1"
                Port "s1-eth4"
                    Interface "s1-eth4"
                Port "s1-eth5"
                    Interface "s1-eth5"
                Port "s1-eth6"
                    Interface "s1-eth6"
                Port "s1-eth3"
                    Interface "s1-eth3"
                Port "s1-eth2"
                    Interface "s1-eth2"

mininet> h1r1 ifconfig h1r1-eth0
h1r1-eth0 Link encap:Ethernet HWaddr 00:00:00:00:00:01
          inet addr:10.0.0.1 Bcast:10.255.255.255 Mask:255.0.0.0
          inet6 addr: fe80::200:ff:fe00:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:1042 errors:0 dropped:668 overruns:0 frame:0
          TX packets:76 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:66551 (66.5 KB) TX bytes:5496 (5.4 KB)

mininet> h1r2 ifconfig h1r2-eth0
h1r2-eth0 Link encap:Ethernet HWaddr 00:00:00:00:00:02
          inet addr:10.0.0.2 Bcast:10.255.255.255 Mask:255.0.0.0
          inet6 addr: fe80::200:ff:fe00:2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:1049 errors:0 dropped:675 overruns:0 frame:0
          TX packets:76 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:66971 (66.9 KB) TX bytes:5492 (5.4 KB)

sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1 | grep 00:00:00:00:00:01
cookie=0x0, duration=92.901s, table=0, n_packets=15, n_bytes=1078, priority=1, in_port=4, dl_dst=00:00:00:00:00:01 actions=output:3
cookie=0x0, duration=92.881s, table=0, n_packets=15, n_bytes=1078, priority=1, in_port=5, dl_dst=00:00:00:00:00:01 actions=output:3
cookie=0x0, duration=92.848s, table=0, n_packets=15, n_bytes=1078, priority=1, in_port=6, dl_dst=00:00:00:00:00:01 actions=output:3
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1 | grep 00:00:00:00:00:02
cookie=0x0, duration=114.359s, table=0, n_packets=14, n_bytes=980, priority=1, in_port=3, dl_dst=00:00:00:00:00:02 actions=output:4
cookie=0x0, duration=114.111s, table=0, n_packets=15, n_bytes=1078, priority=1, in_port=5, dl_dst=00:00:00:00:00:02 actions=output:4
cookie=0x0, duration=114.097s, table=0, n_packets=15, n_bytes=1078, priority=1, in_port=6, dl_dst=00:00:00:00:00:02 actions=output:4

```

Figure 2.29: Switch1 Flow Entries and Ports with Datacentre Topo and Ryu Controller.

The content of next section explains the test environment as well as the software which will be used during the experiments in *Chapter 3*. It also goes over SDN controllers, their use, operation, different topologies and OF versions.

2.7. Test Methodology

At the time of determining the subject matter and scope of this work, I have decided that any attempt to study effectiveness and possibilities of described network protocols will take place in a “*hybrid*” environment. We could create a solution completely based on the VMs that could be run on any computer without the need for any devices. This would be extremely convenient as we wouldn't have any problems with a lack of access to the physical hardware.

However, I decided to work on physical routers due to several reasons. First, the configuration and testing environment which will be used for the physical equipment is closer to what we can find in the actual network infrastructure. Running configuration can be a part of a larger segment of the network which makes the obtained results more easily applied in practice. Secondly, while examining the effectiveness of protocols, more reliable results can be obtained with real hardware than when emulating the operation of routers using a virtual environment. Thirdly, Cisco products are widely used in commercial solutions which is why it is worthwhile to check what we can gain by using them to build the network infrastructure.

The following sections will contain a detailed description of the preparation, installation, and configuration of the “*test-bed*” and the necessary software which will be used during the tests.

2.7.1. Software Routers

In this work, we have decided to install Linux kernel version 4.12 (Kernel Newbies, 2017) on Ubuntu version 16.04.2 (Ubuntu, 2017) which supports MPLS since Linux version 4.1 was released in June 2015. To prepare the OS for the new kernel we have used Ubuntu Kernel Update Utility (UKUU) version 17.2.3 installed from Personal Package Archive (PPA) developed by George (2015) as per instructions in Ji (2017). In this case, this was caused due to the ease of a kernel upgrade within the OS and the compatibility with embedded MPLS as seen in *Figure 2.30*.

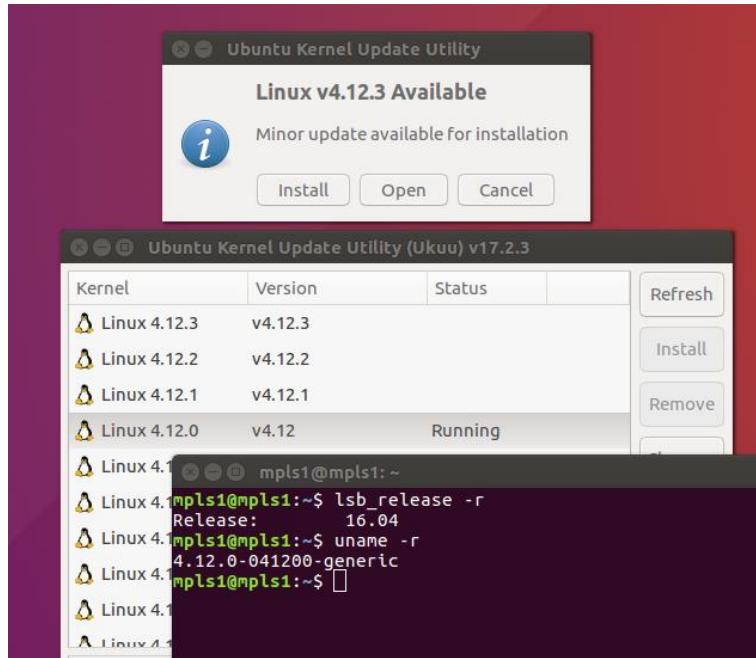


Figure 2.30: Kernel Upgrade in UKUU.

According to Russell (2015) and Prabhu (2016) we also must use a compatible IPRoute2 to add routing between devices, so we have decided to download Tape Archive (TAR) of IPRoute version 4.12.0 (Kernel, 2017) which in turn must be compiled before installation with the use of Bison parser and Flex generator (Wilson, 2016). Russell (2015) and Stack Overflow (2015) member called *user2798118* stated that MPLS is not enabled by default in Linux kernel. In this situation, we must load the module by executing *sudo modprobe mpls_router* as well as enabling the protocol for specific interfaces and assign labels to LSRs.

Since we are limited in terms of available HW resources we will also use VMs with *Cisco CSR 1000V Series IOS* version 3.12.2S (*csr1000v-universalk9.03.12.00.S.154-2.S-std*) installed with vendor minimal requirements of 3 GBs of RAM and 1 Virtual Central Processing Unit (vCPU) (Cisco, 2017). This will allow testing the scalability and interoperability of the protocols in the network which wouldn't be possible with the use of three physical routers.

A detailed configuration manual to implement the support of MPLS protocol is in *Appendix 1* and CSR nodes setup can be found in *Appendix 6*.

2.7.2. Hardware Routers

The Cisco 2801 is a device which provides functions which support MPLS. Necessary for this, we must have appropriate software. The network environment will be using *mpls static binding* to implement hop-by-hop forwarding for neighbours which do not use Label Distribution Protocol (LDP) (Cisco, 2007) or a dynamic method of distribution with OSPF

protocol which will assign labels to routes (Cisco, 2005).

To be able to use these features we will use IOS version 15.x *Advanced Enterprise Services* designed for the model 2801.

2.7.3. Software Switches

To use the PC as a set of switches with OpenFlow protocol we are going to also use Ubuntu OS version 16.04.2 LTS with the codename “*Xenial Xerus*” (Ubuntu, 2017) and Linux kernel version 4.12.

To start the preparation of our OpenFlow and Mininet VMs, we would need to install Git Source Code Management System (SCMS) from Ubuntu repositories by executing *sudo apt-get install git* (Git, 2017). Next, according to OpenFlow (2011), we will install OF10 via *clone* command from *git://gitosis.stanford.edu/openflow* and we also need to make sure that IP version 6 (IPv6), as well as Avahi mDNS/DNS-SD daemon, are turned off (SysTutorials, 2017). Since we will be emulating network behaviour and vSwitches as well as Virtual Hosts (vHosts) in Linux namespaces, we decided to perform a full installation of a Mininet environment in version 2.2.2 from a native source (Mininet, 2017) on Ubuntu 16.04.2 and Linux kernel 4.12, rather than using the pre-built VM of Ubuntu 14.04.04 (GitHub, 2017) with kernel version 4.2 as stated by Sneddon (2016).

We also installed Wireshark version 2.2.6 on all OpenFlow supported VMs from PPA (Ji, 2016) because it provides stable dissectors as per release of version 1.12.X (Wireshark, 2016). It, however, requires network privileges for *dumpcap* (Wireshark, 2014) as well as *universe* repository according to *Wireshark Developers* team (Launchpad, 2017) and user permissions to execute the packet capture on the interface which can be changed by executing *sudo dpkg-reconfigure wireshark-common* (AskUbuntu, 2013).

A detailed configuration manual to implement the support of OpenFlow protocol is in *Appendix 2*.

This chapter explains why this specific approach was decided for the test methodology as well as why a mixture of hardware and software nodes were implemented to build test cases in the following section.

3. Performance and Compatibility Tests

In this chapter we will describe in detail the tests performed which are supposed to show the advantages or disadvantages of the above-mentioned solutions. It will also contain obtained results from experiments which in further parts will be used to formulate adequate conclusions about the usefulness of these technologies. The hypothesis will verify and determine whether the MPLS or OpenFlow are able to efficiently manage network traffic without compromising the QoS or even improving it with use of TE.

The experiments can be broadly divided into six main groups:

1. Checking for MPLS interoperability between MPLS implementation for Linux and Cisco IOS Multiprotocol Label Switching.
2. Comparison of efficiency of the network computer based for standard IP routing solutions: MPLS in Linux, Cisco IOS MPLS, and OpenFlow.
3. Scaling up the network by adding additional nodes to MPLS and OF environments.
4. Tests of QoS approaches within various MPLS and OF topologies.
5. The use of TE with the use of the described technologies.
6. Explaining what the possibilities in response to the failure of certain parts of the network while using both protocols.

The next part of this chapter includes an accurate description of the issues mentioned in the paragraphs above.

3.1. Compatibility between MPLS in Linux and Cisco MPLS

In this topology, we will use the prepared earlier VMs some of which are software routers using MPLS in Linux kernel and the others are Cisco routers with MPLS support. Dublin node is the Cisco router which will act as LER or LSR depending of the topology, while Kildare and Laois are the Linux software routers with MPLS support, where VMs will act as end clients running in Hyper-V. We will check if there is a possibility of cooperation between them. To do this we will create a simple network consisting of five devices, three of which will act as routers and two which will be standard computers in form of VMs.

We can see that these devices are working in four different subnets. The aim is to provide communication using MPLS on the edge routers (Kildare, Laois) between two nodes which are VMs. This small part of the network topology is often found in real commercial solutions where the Internet Service Provider (ISP) utilizes MPLS label switching which from the end users point of view is completely invisible.

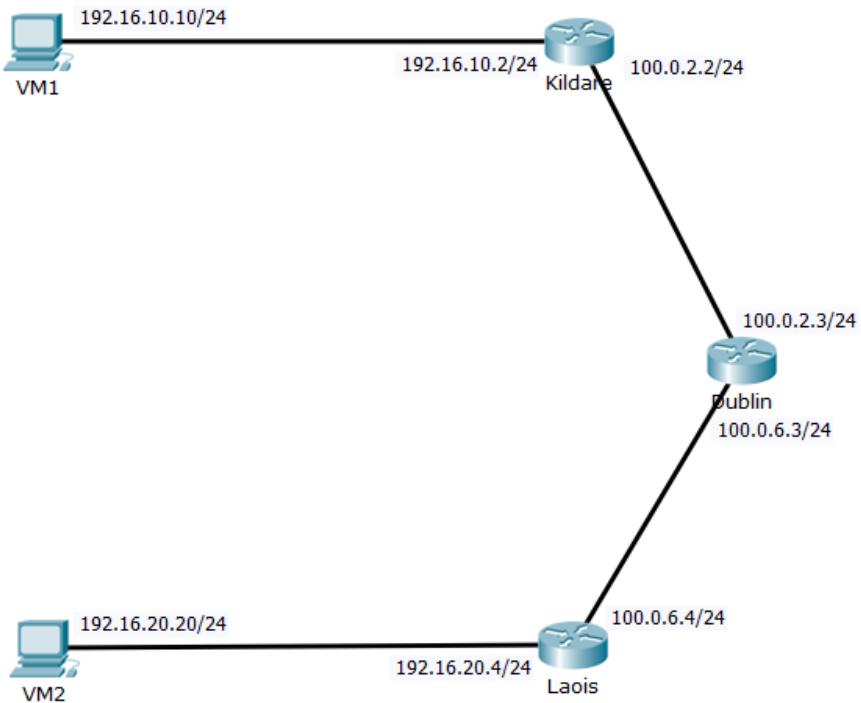


Figure 3.1: MPLS Compatibility Topology.

3.1.1. Configuration

In the above *Figure 3.1*, routers were assigned as Kildare, Dublin, and Laois which in turn are connected to corresponding Hyper-V vSwitches bound to specific NICs with their own vNICs where each is connected to the individual network port on the Dublin Cisco router as per below mappings (*Figure 3.2*):

- FastEthernet0/1 – NIC1 (eth1) to Kildare (MPLS1)
- FastEthernet0/0 – NIC3 (eth1) to Laois (MPLS2)

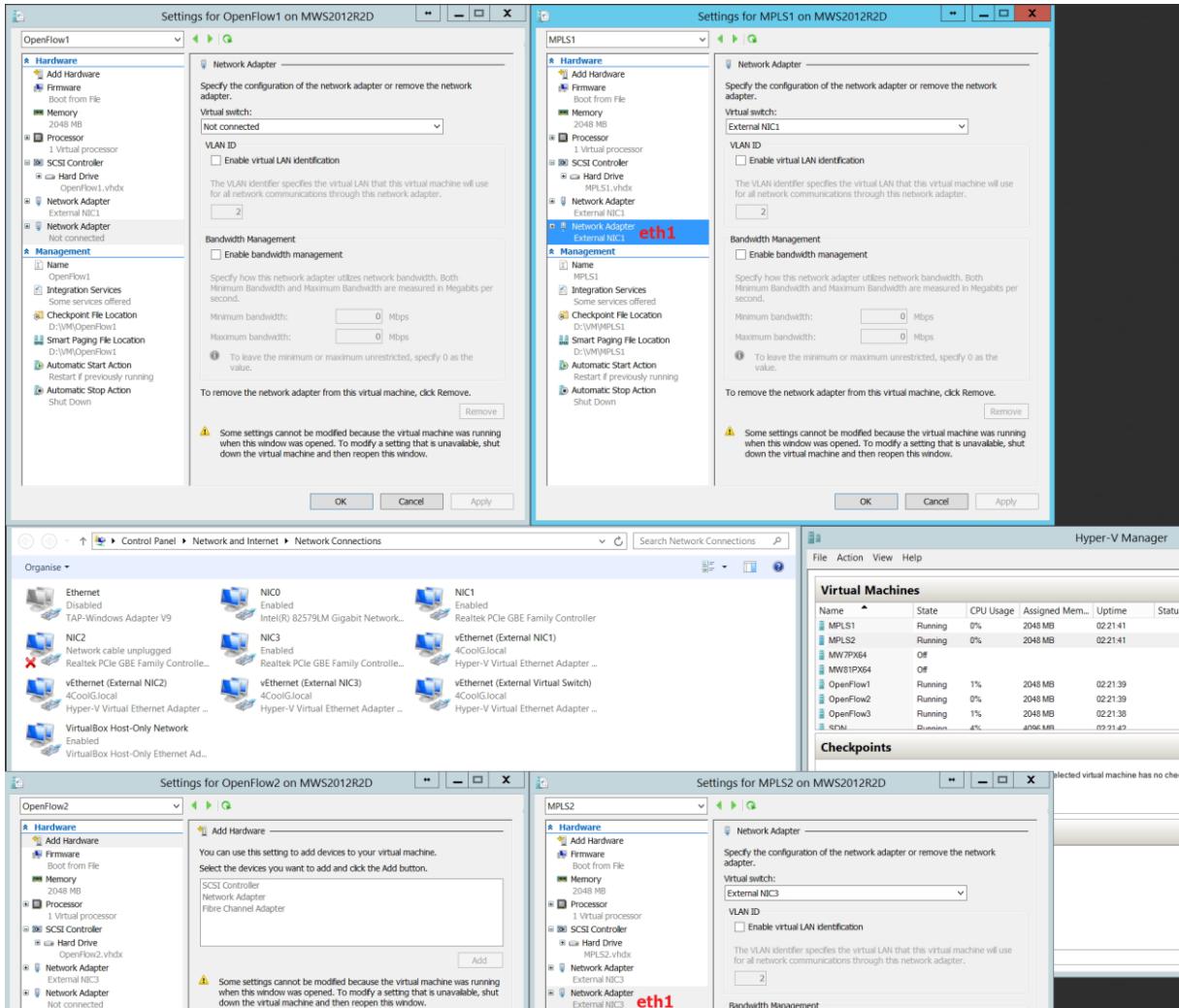


Figure 3.2: Virtual Interface Mapping to Kildare and Laois MPLS VMs.

To make sure that VM1 (OpenFlow1) and VM2 (OpenFlow2) are configured correctly we need to ensure that VM1 can reach the 192.16.20.0/24 network and VM2 can reach the 192.16.10.0/24 network. Commands specified below in *Figure 3.3* will set the correct IP addresses on computer network interfaces and it will also add static entries to the routing table as well as its default routes via Gateways (GWs). These rules ensure that the traffic from the source will reach the target as it will be routed to the devices which play the roles of the edge routers.

```

= OpenFlow1 =
sudo ifconfig eth0 192.16.10.10 netmask 255.255.255.0 up
sudo ip route add 192.16.20.0/24 via 192.16.10.2
sudo ip route add default via 192.16.10.2 dev eth0

= OpenFlow2 =
sudo ifconfig eth0 192.16.20.20 netmask 255.255.255.0 up
sudo ip route add 192.16.10.0/24 via 192.16.20.4
sudo ip route add default via 192.16.20.4 dev eth0

```

Figure 3.3: Static Routes on OpenFlow VMs to Kildare and Laois MPLS VMs.

On the boundary routers, described as Kildare and Laois, which act as LERs labels are attached or removed during packet transmission where Dublin is the LSR which replaces the labels between LSPs. After a packet is received on the LSR, it looks for the right rule in the label forwarding table and if the correct entry is found, it executes the action assigned to it. In this case, it converts the MPLS label and passes the packet to the next node where the label will be removed before forwarding the data to the destination.

During these tests, we have used one Cisco 2801 hardware router and two MPLS software routers. Network operation was checked both when the Cisco router worked as transit node (LSR) and when it was configured as ingress node (LER), as seen in *Figure 3.4*. Sample scripts for software routers and command sequences configuring Cisco devices with which we can reproduce any of the variants used during this test are available in *Appendix 3*.

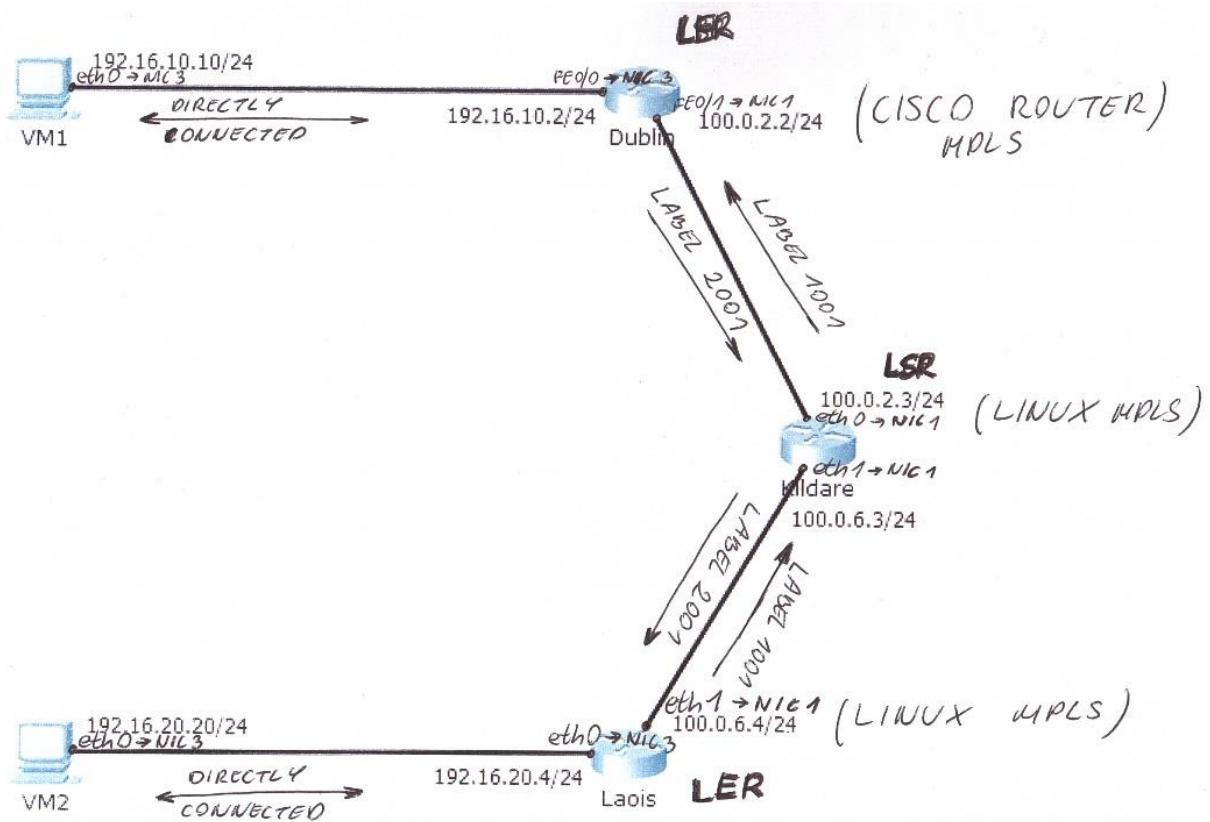


Figure 3.4: Dublin Cisco MPLS as LER.

3.1.2. Results

When VM1 sends the Internet Control Message Protocol (ICMP) request to VM2 via the Kildare router we can observe on eth1 MPLS label 2001 that when it receives the reply via Laois router on eth1 it has a label 1001 in the response to 192.16.10.10/24, as seen in *Figure 3.5*.

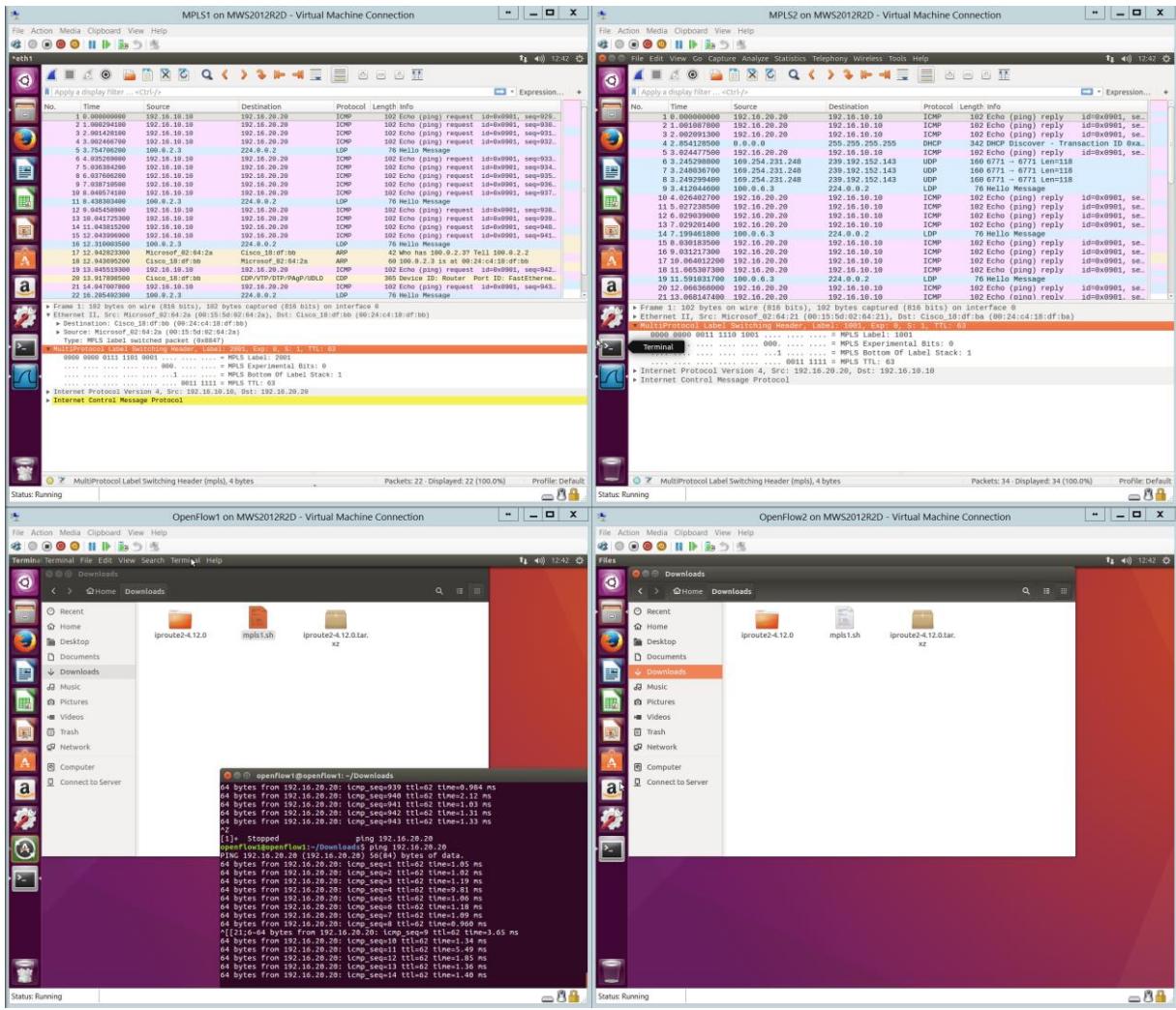


Figure 3.5: VM1 ICMP Request to VM2.

However, during the request from VM2 to 192.16.10.10/24, we can see that MPLS uses label 1001 and the reply from VM1 has the label 2001 in the IP packet header, as seen in *Figure 3.6*.

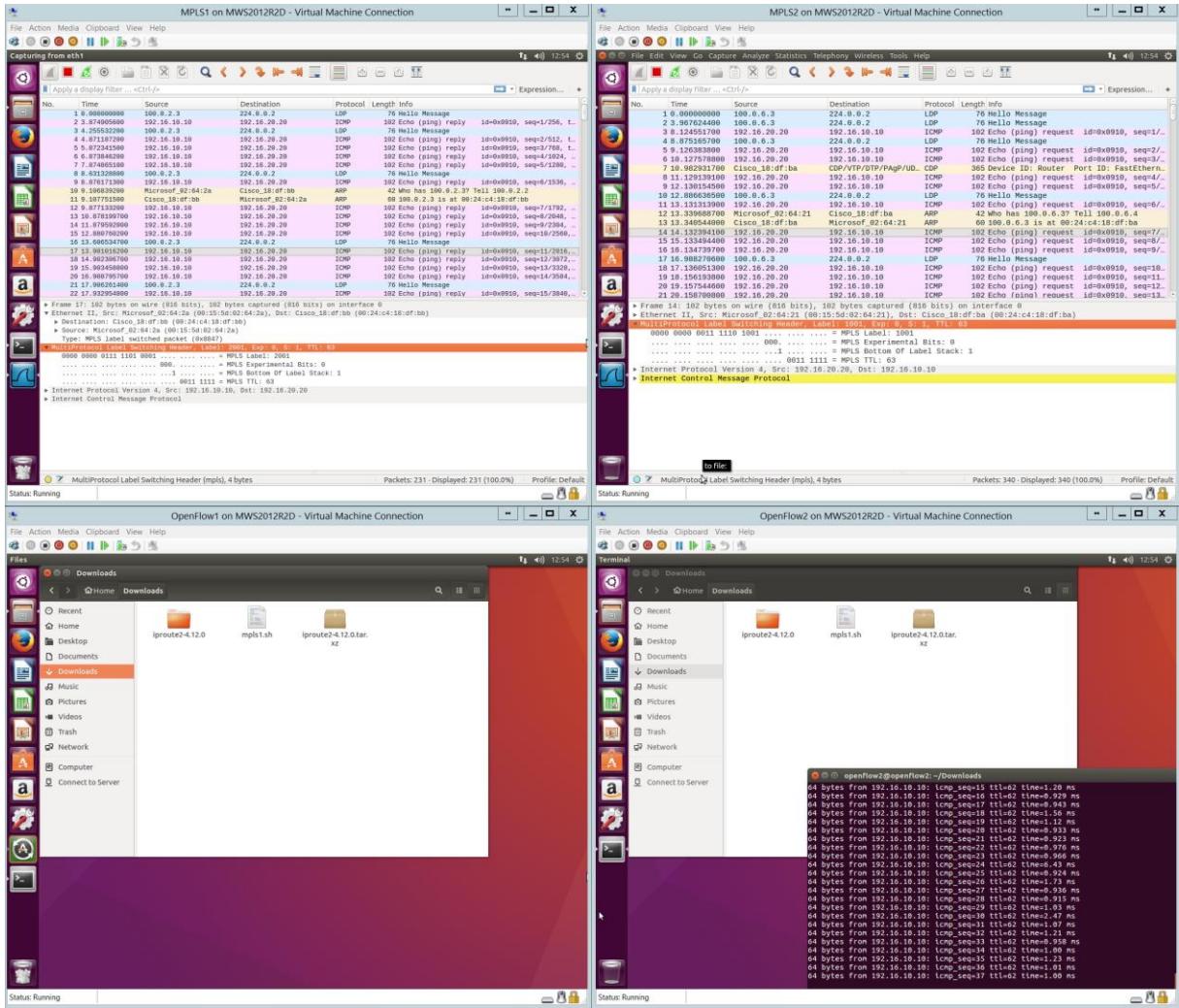
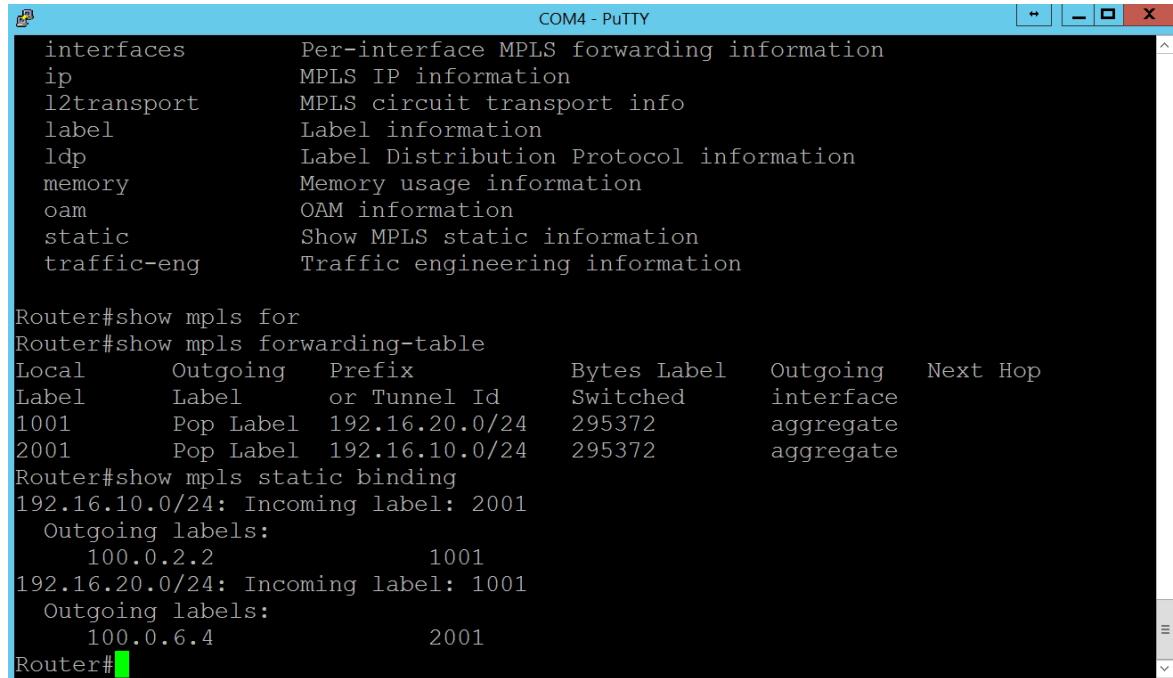


Figure 3.6: VM2 ICMP Request to VM1.

In the above scenario, the Dublin router was configured to act as the LSR responsible for the transition of packets between the LSPs with outgoing label 1001 to eth1 on the Kildare router for incoming label 2001 from 192.16.10.0/24 network and outgoing label 2001 to eth1 on the Laois router for incoming label 1001 from 192.16.20.0/24 network, as seen in *Figure 3.7*.



A screenshot of a PuTTY terminal window titled "COM4 - PuTTY". The window displays Cisco IOS command-line interface (CLI) output. The user has run several commands related to MPLS configuration:

```

interfaces          Per-interface MPLS forwarding information
ip                 MPLS IP information
l2transport        MPLS circuit transport info
label              Label information
ldp                Label Distribution Protocol information
memory             Memory usage information
oam                OAM information
static             Show MPLS static information
traffic-eng       Traffic engineering information

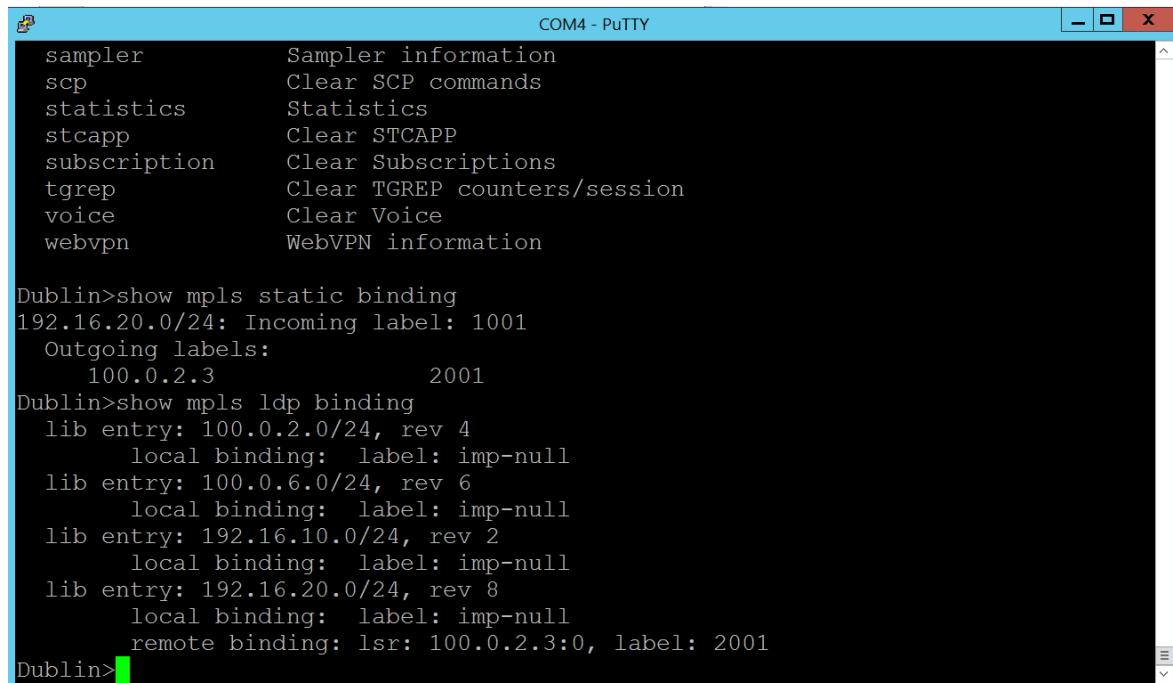
Router#show mpls for
Router#show mpls forwarding-table
Local      Outgoing   Prefix           Bytes Label    Outgoing   Next Hop
Label     Label      or Tunnel Id   Switched
1001      Pop Label  192.16.20.0/24  295372      interface
2001      Pop Label  192.16.10.0/24  295372      aggregate
Router#show mpls static binding
192.16.10.0/24: Incoming label: 2001
  Outgoing labels:
    100.0.2.2          1001
192.16.20.0/24: Incoming label: 1001
  Outgoing labels:
    100.0.6.4          2001
Router#

```

Figure 3.7: Cisco Router Outgoing and Incoming MPLS Labels as LSR.

This static MPLS binding allows using tunnels to pass packets with a specific label in the header and strip it to forward the data to the destination based on the endpoint interface assigned to the outgoing and incoming labels.

In the second scenario, the Dublin router was configured to act as the LER to accept incoming label 1001 and forward label 2001 to eth0 on Kildare router acting as the LSR, as seen in *Figure 3.8*.



A screenshot of a PuTTY terminal window titled "COM4 - PuTTY". The window displays Cisco IOS CLI output. The user has run several commands related to MPLS configuration:

```

sampler          Sampler information
scp              Clear SCP commands
statistics       Statistics
stcapp          Clear STCAPP
subscription    Clear Subscriptions
tgrep            Clear TGREP counters/session
voice            Clear Voice
webvpn          WebVPN information

Dublin>show mpls static binding
192.16.20.0/24: Incoming label: 1001
  Outgoing labels:
    100.0.2.3          2001
Dublin>show mpls ldp binding
  lib entry: 100.0.2.0/24, rev 4
    local binding: label: imp-null
  lib entry: 100.0.6.0/24, rev 6
    local binding: label: imp-null
  lib entry: 192.16.10.0/24, rev 2
    local binding: label: imp-null
  lib entry: 192.16.20.0/24, rev 8
    local binding: label: imp-null
    remote binding: lsr: 100.0.2.3:0, label: 2001
Dublin>

```

Figure 3.8: Cisco Router Outgoing and Incoming MPLS Labels as LER.

During the test, we managed to uncover that while both the Kildare (MPLS1) and Laois (MPLS2) routers can perform forwarding and label replacement, they cannot de-capsulate the packet to pop the outgoing label before forwarding it to the destination.

When we send an ICMP request from VM1 to eth0 on Kildare router we receive a reply with the label 1001 (*Figure 3.9*) and when from VM2 to eth1 the label 2001 is returned in the packet (*Figure 3.10*).

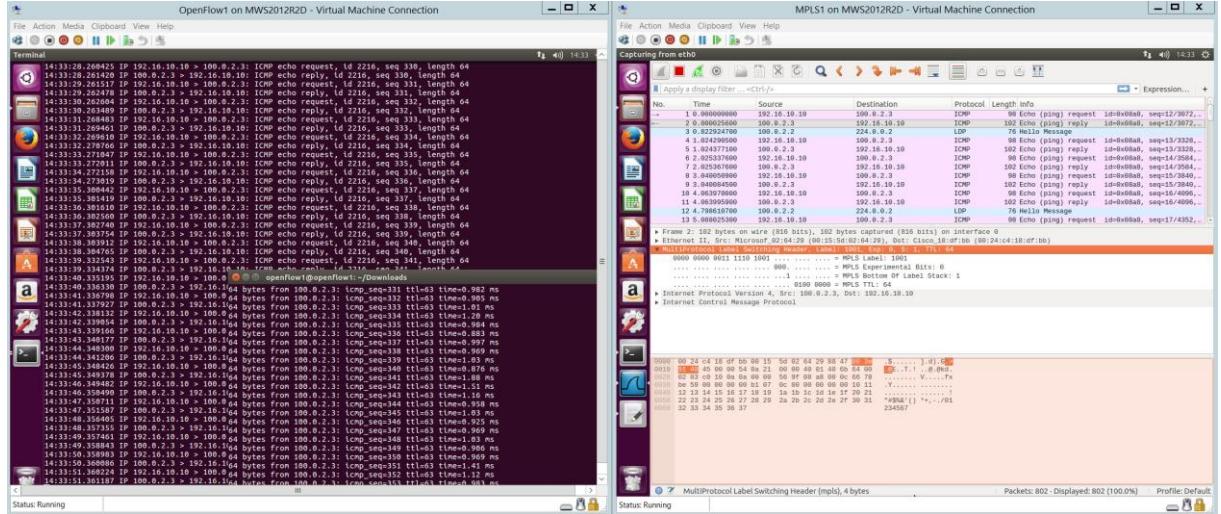


Figure 3.9: Kildare Router ICMP Reply to VM1.

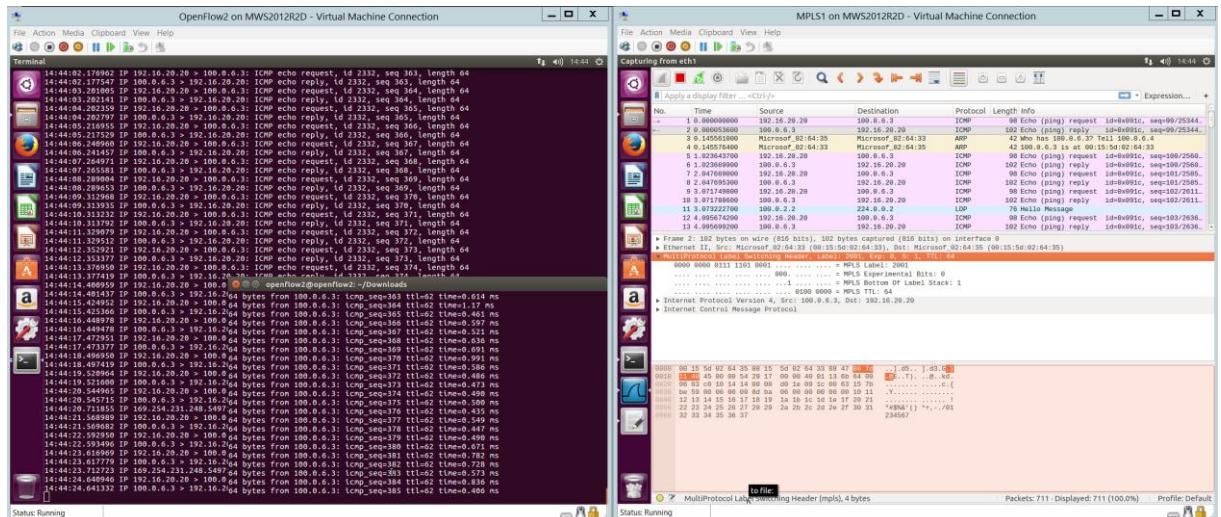


Figure 3.10: Kildare Router ICMP Reply to VM2.

However, from VM2 we were not able to receive a response from VM1's GW on the Dublin router with label 2001 even if it's correctly forwarded via the LSP during the request (*Figure 3.11*), but we managed to get a reply during the VM1 ICMP request to VM2's GW on the Laois router with the label 1001 (*Figure 3.12*).

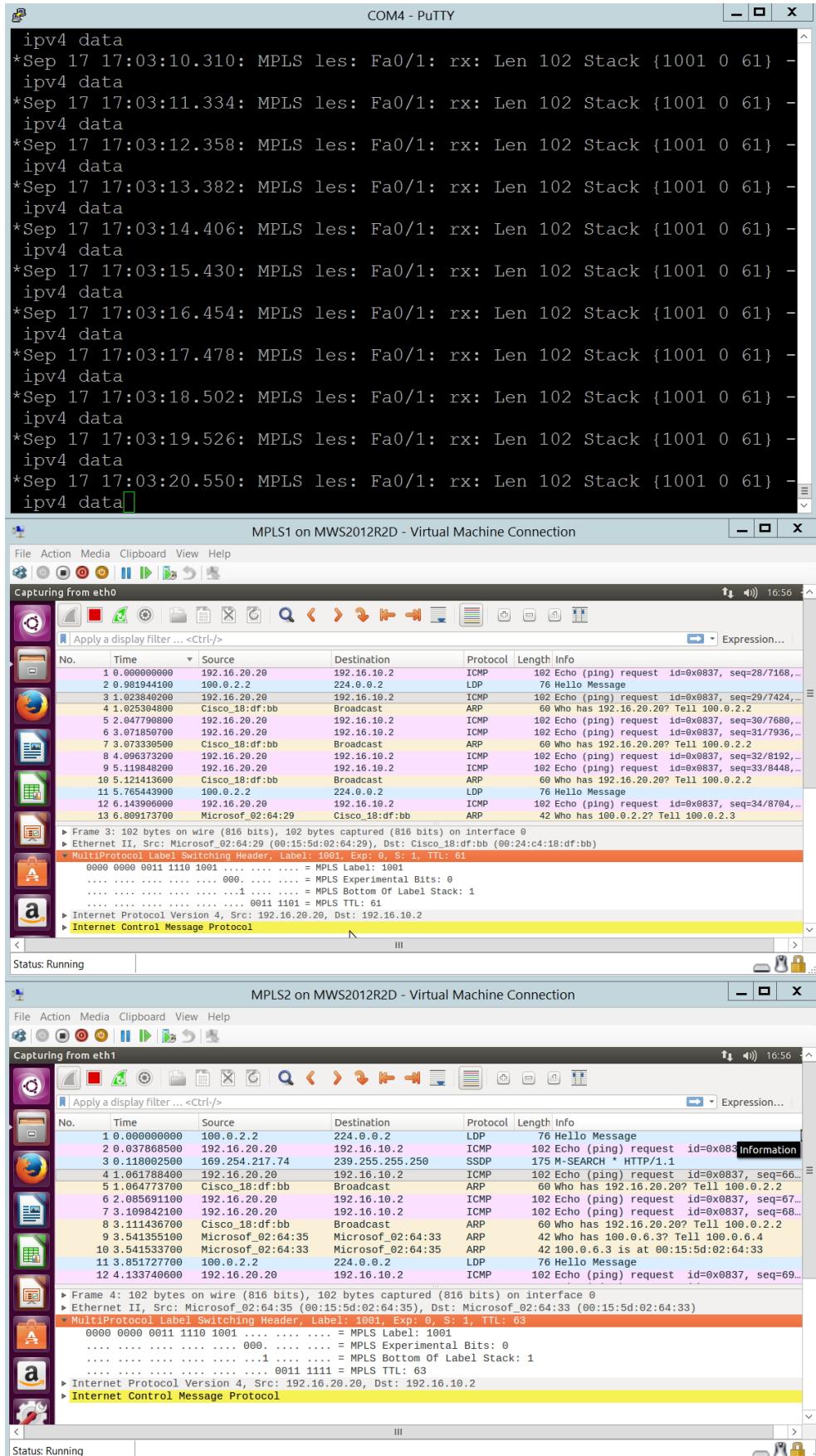


Figure 3.11: LSP from VM2 to VM1's GW between both LERs.

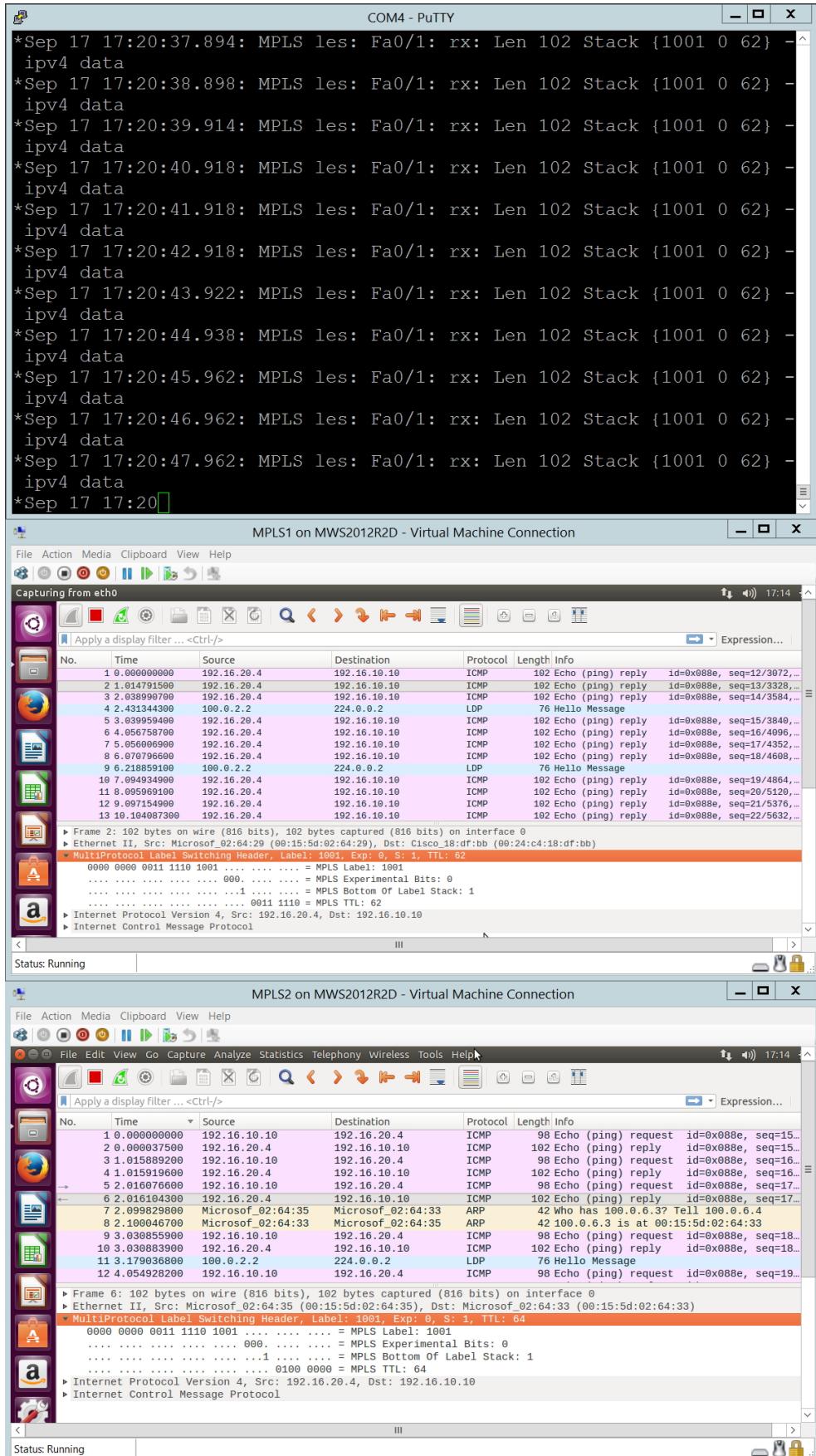


Figure 3.12: LSP from VM1 to VM2's GW between both LERs.

This test has proven that the Kildare router acting as the LSR isn't able to correctly pop the outgoing label while forwarding the packets on the LSP to VM1's GW and thus it will not be able to forward packets to Dublin's directly connected network 192.16.10.0/24.

It's hard to imagine that while MPLS increases its popularity as stated by 2Connect (2017) it's still not properly supported by the OS kernel which is widely implemented in Linux based solutions such as VyOS (VyOS, 2017). It proves that MPLS support in Linux isn't fully compatible as Linux software nodes acting as LSR cannot strip out the label before forwarding the packet to the next hop within the MPLS network.

Therefore, we have decided that in the remaining tests we are only going to investigate the scenario where the Dublin router will act as the LSR and other remaining Linux MPLS enabled routers will be configured as LERs.

3.2. IP Performance

It was already described in the previous chapters that one of the motivations for the use of MPLS protocols and OpenFlow is often the need to provide services with QoS. It is widely accepted that QoS is determined because of the following performance parameters (Rogier, 2016):

- Throughput of the network.
- Delay (jitter).
- Number of packets lost during transmission.

Except for the three above QoS parameters, we are also going to look at Round-Trip Time (RTT) which will show the response times between two endpoints with ICMP.

With the use of TE mechanisms which are available using the described technologies, we should be able to improve packet transmission performance in relation to TCP/IP. This chapter describes exactly how we have checked the impact of the use of MPLS and OpenFlow to describe the parameters for the quality of the network.

3.2.1. Configuration

The point of reference used to determine changes in productivity and quality of transmission using MPLS and OpenFlow was based on TCP/IP. Two VMs were acting as routers with Linux OS where IP forwarding was enabled, routing tables in all nodes of the network have been assigned in a static way and four networks served as the infrastructure backbone. We will still use the topology diagram shown in *Figure 3.1*, but attention should be paid to the configuration changes detailed in *Appendix 4*.

Taking into consideration the configuration described in the previous paragraph we will compare three solutions based on different technologies:

- Software routers using MPLS packet forwarding for Linux.
- Cisco hardware routers with the support of MPLS.
- Switches which use software and OpenFlow protocol.

Device settings for MPLS based routers are the same as for compatibility testing in *Appendix 3* described in *Chapter 3.1*.

It looks little different if we consider the situations in which we use the switches using OpenFlow protocol. The network equipment which acts as the data link layer of the ISO OSI forces the end nodes to be located within the same subnet. The switches do not only use routing based on the interface IP addresses but to transfer the traffic they use port names. Communication with the controller through a secure channel is possible because the switches listen on TCP port 6634. In our topology, the switches described as: S1, S2, S3 run in Mininet, where numbers beside them represent their ports, as seen in *Figure 3.13*. VM1 and VM2 are running in Hyper-V which have their default routes configured to physical NIC. This in turn acts as gateway for 192.16.20.0/24 network and thus it facilitates the communication between end clients VMs and Mininet via Linux *eth1* interface which is bound to the network adapter via *OpenFlow1.py* script.

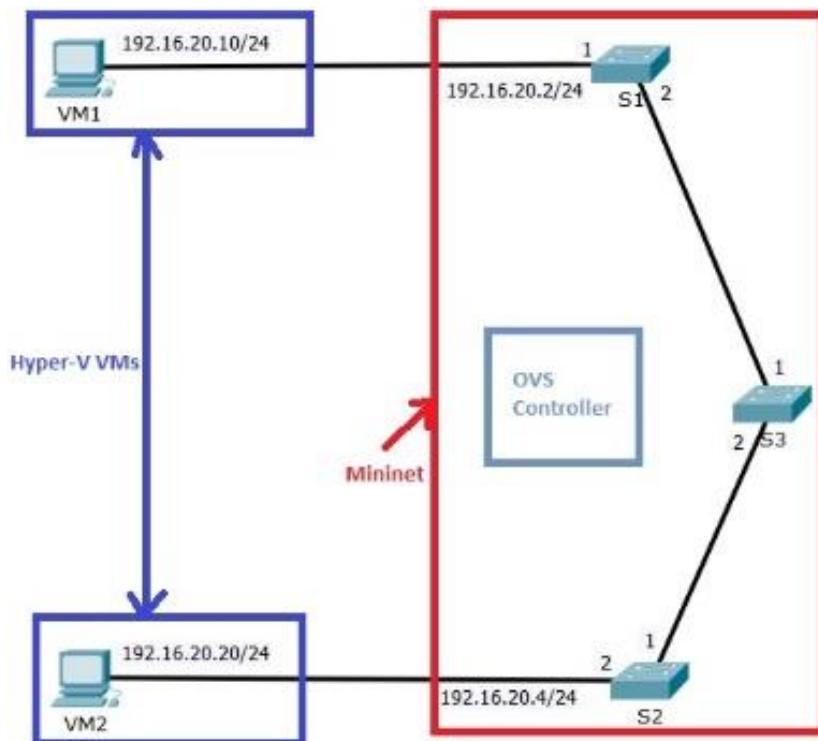
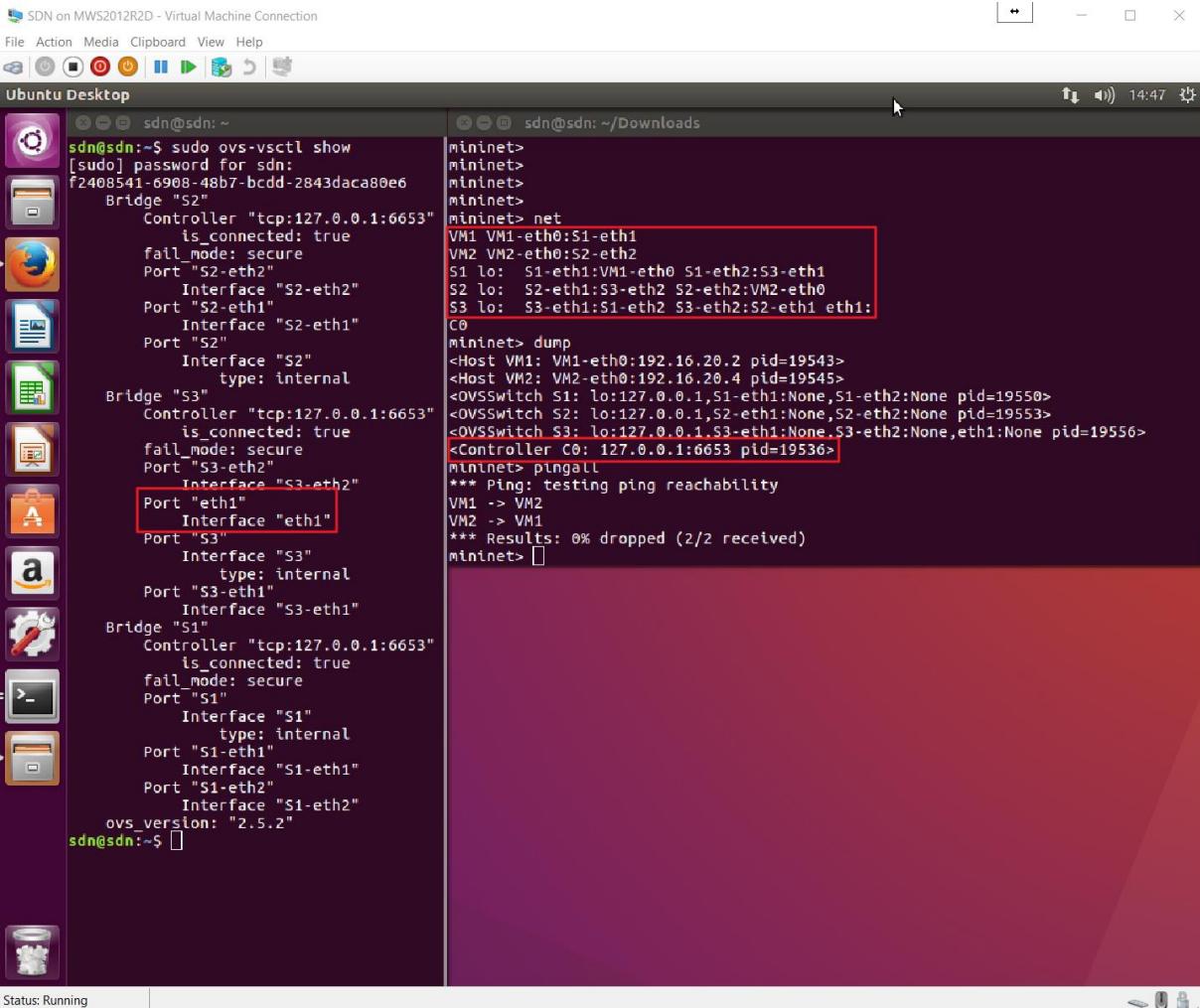


Figure 3.13: OpenFlow Performance Topology.

We can see that it does not show the device that plays the role of the controller as a part of the network because it only controls the flow of data paths in between the switches by making up a decision based on the rules on which port to forward the packets. For this experiment, it was enough to add two simple rules to the flow table of each switch. This is like placing static entries in a routing table or in the forwarding table for MPLS. For this purpose, we do not need any sophisticated network traffic controller, so we can restrict ourselves to use a tool called Data Path Controller (DPCtl) and default Open Virtual Switch (OVS) local controller in Mininet which will be bound to eth1 and bridged to S3, as seen in *Figure 3.14*.



The screenshot shows a desktop environment with two terminal windows. The left terminal window is titled 'Ubuntu Desktop' and shows the command output of 'sudo ovs-vsctl show'. The right terminal window is titled 'sdn@sdn: ~/Downloads' and shows the output of 'mininet> net' followed by 'mininet> pingall'. The 'pingall' command results are highlighted with a red box, showing successful pings between VM1 and VM2.

```

sdn@sdn:~$ sudo ovs-vsctl show
[sudo] password for sdn:
f2408541-6908-48b7-bcdd-2843daca80e6
  Bridge "S2"
    Controller "tcp:127.0.0.1:6653"
      is_connected: true
      fail_mode: secure
      Port "S2-eth2"
        Interface "S2-eth2"
      Port "S2-eth1"
        Interface "S2-eth1"
      Port "S2"
        Interface "S2"
        type: internal
  Bridge "S3"
    Controller "tcp:127.0.0.1:6653"
      is_connected: true
      fail_mode: secure
      Port "S3-eth2"
        Interface "S3-eth2"
      Port "eth1"
        Interface "eth1"
      Port "S3"
        Interface "S3"
        type: internal
      Port "S3-eth1"
        Interface "S3-eth1"
  Bridge "S1"
    Controller "tcp:127.0.0.1:6653"
      is_connected: true
      fail_mode: secure
      Port "S1"
        Interface "S1"
        type: internal
      Port "S1-eth1"
        Interface "S1-eth1"
      Port "S1-eth2"
        Interface "S1-eth2"
  ovs_version: "2.5.2"
sdn@sdn:~$ 

mininet>
mininet>
mininet>
mininet>
mininet>
VM1 VM1-eth0:s1-eth1
VM2 VM2-eth0:S2-eth2
S1 lo: S1-eth1:VM1-eth0 S1-eth2:S3-eth1
S2 lo: S2-eth1:S3-eth2 S2-eth2:VM2-eth0
S3 lo: S3-eth1:S1-eth2 S3-eth2:S2-eth1 eth1:
C0
mininet> dump
<Host VM1: VM1-eth0:192.16.20.2 pid=19543>
<Host VM2: VM2-eth0:192.16.20.4 pid=19545>
<OVSSwitch S1: lo:127.0.0.1,S1-eth1:None,S1-eth2:None pid=19550>
<OVSSwitch S2: lo:127.0.0.1,S2-eth1:None,S2-eth2:None pid=19553>
<OVSSwitch S3: lo:127.0.0.1,S3-eth1:None,S3-eth2:None,eth1:None pid=19556>
<Controller C0: 127.0.0.1:6653 pid=19536>
mininet> pingall
*** Ping: testing ping reachability
VM1 -> VM2
VM2 -> VM1
*** Results: 0% dropped (2/2 received)
mininet> 

```

Figure 3.14: OVS Bridge and Default Local Controller in Mininet.

This is a program which allows us to communicate with a switch using the OpenFlow protocol to add and delete entries from flow tables, view traffic statistics and perform basic switch configurations (KickstartSDN, 2015).

We will allow all TCP traffic in on a specific port and out on another one without any delay to gain the best performance possible as per the commands executed on the switches displayed

in *Figure 3.15*.

```
= xterm S1 - OpenFlow =
dpctl add-flow tcp:127.0.0.1:6653 in_port=1,idle_timeout=0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6653 in_port=2,idle_timeout=0,actions=output:1
= xterm S3 - OpenFlow =
dpctl add-flow tcp:127.0.0.1:6653 in_port=1,idle_timeout=0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6653 in_port=2,idle_timeout=0,actions=output:1
= xterm S2 - OpenFlow =
dpctl add-flow tcp:127.0.0.1:6653 in_port=1,idle_timeout=0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6653 in_port=2,idle_timeout=0,actions=output:1
```

Figure 3.15: DPCTL Switch Commands.

Scripts displaying the exact configurations used in this comparison are available in *Appendix 4*.

Network performance studies based on different technologies were conducted using the previously described traffic generation tools discussed in *Chapter 2.2*. These programs gave us the ability to observe selected transmission parameters and the results are discussed further in the next section.

3.2.2. Results

3.2.2.1. Throughput

Network throughput was checked using iPerf3 whereas for reference we have tested the throughput of the connection that we observed when VM1 was connected directly to VM2 via Internal vSwitch within a single subnet, as seen in *Figure 3.16*.

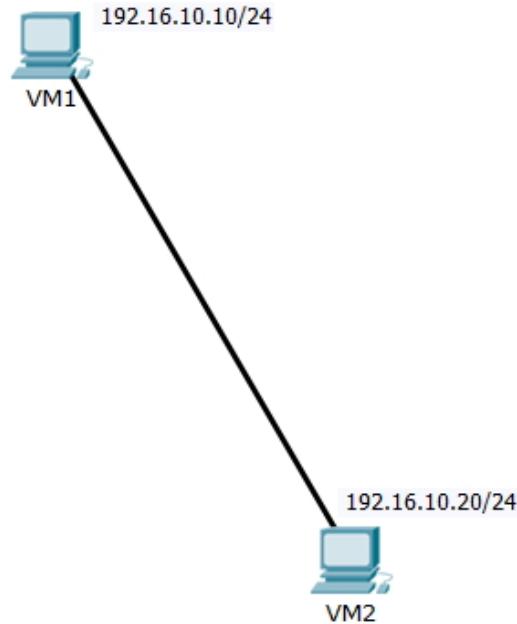


Figure 3.16: P2P Connection on Internal vSwitch.

Since our Cisco routers, NICs maximum throughput is 100 Megabits per second (Mbps) we must assure that the test conditions will meet the validation criteria and therefore we have used a bandwidth limit of 100000000 Bits per second (Bps).

To do this we have decided to run iPerf3 Server on VM1 via `iperf3 192.16.10.10 -s -i 1` and then execute a bidirectional TCP test on VM2 (Client) which will continue for 60 seconds where the result will be saved to `C1VM2toVM1x` files (x - letter of alphabet) via `iperf3 -c 192.16.10.10 -t 60 -d -b 100000000 / tee C1VM2toVM1.txt`.

The average result we got at a nominal connection speed of 100 Mbps was 99.8 Mbps, as seen in *Figure 3.17*.

OpenFlow1 on MWS2012R2 - Virtual Machine Connection

```

File Action Media Clipboard View Help
openflow[1]:~$ ./openflow -DDownloads
[ 5] 20.08-21.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 21.08-22.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 22.08-23.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 23.08-24.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 24.08-25.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 25.08-26.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 26.08-27.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 27.08-28.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 28.08-29.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 29.08-30.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 30.08-31.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 31.08-32.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 32.08-33.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 33.08-34.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 34.08-35.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 35.08-36.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 36.08-37.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 37.08-38.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 38.08-39.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 39.08-40.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 40.08-41.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 41.08-42.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 42.08-43.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 43.08-44.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 44.08-45.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 45.08-46.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 46.08-47.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 47.08-48.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 48.08-49.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 49.08-50.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 50.08-51.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 51.08-52.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 52.08-53.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 53.08-54.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 54.08-55.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 55.08-56.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 56.08-57.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 57.08-58.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 58.08-59.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 5] 59.08-60.00 sec 12.0 MBytes 101 Mbits/sec
[ 5] 60.08-00.04 sec 8.00 Bytes 8.00 bits/sec

[ ID] Interval           Transfer     Bandwidth   Retr
[ 5] 0.00-00.04 sec    714 MBytes  99.8 Mbits/sec  0          sender
[ 5] 0.00-00.04 sec    714 MBytes  99.8 Mbits/sec  0          receiver

```

Server listening on 5001

Status: Running

OpenFlow2 on MWS2012R2 - Virtual Machine Connection

```

File Edit View Search Tools Documents Help
openflow[1]:~$ ./openflow -DDownloads
[ 4] 55.08-56.00 sec 12.0 MBytes 101 Mbits/sec
[ 4] 56.08-57.00 sec 12.0 MBytes 101 Mbits/sec
[ 4] 57.08-58.00 sec 11.9 MBytes 99.6 Mbits/sec
[ 4] 58.08-59.00 sec 12.0 MBytes 101 Mbits/sec
[ 4] 59.08-00.04 sec 12.0 MBytes 101 Mbits/sec
[ 4] 0.00-00.04 sec 714 MBytes 99.8 Mbits/sec  0          sender
[ 4] 0.00-00.04 sec 714 MBytes 99.8 Mbits/sec  0          receiver

```

iperf Done.

Status: Running

Plain Text Tab Width: 8 Ln 1 Col 1

Figure 3.17: VM2 to VM1 P2P Throughput Test.

By treating the above results as a kind of benchmark we will examine the throughput of networks based on MPLS technologies in Linux and Cisco routers as well as OpenFlow implementation.

To test forwarding of the packets with TCP/IP we have only implemented static and default routes for networks in the topology shown in *Figure 3.1*. This time we are going to save them into Comma Separated Values (CSV) *C1VM2toVM1x* files (x - letter of alphabet) via *iperf3 -c 192.16.10.10 -t 60 -d -b 100000000 / tee C1VM2toVM1.csv* for further analytics.

During MPLS in the Linux test, we have used the configuration as in *Chapter 3.1.1* where the Kildare and Laois routers acted as LERs and Dublin were acting as LSR (*Appendix 3*).

While we were testing MPLS on Cisco hardware routers we have decided to configure all the devices with static bindings to FECs according to the topology shown in *Figure 3.4* where the Dublin router was an LSR as it was previously discussed in *Chapter 3.1.1*.

For OpenFlow tests, we have decided to create the *NATSwitch* discussed by Finn (2017) with Internal vSwitch and GW of 192.16.20.1/24 to route the traffic for 192.16.20.0/24 network as per topology displayed in *Figure 3.13*. In this scenario, VM2 (OpenFlow2) acts as iPerf3 Client and VM1 is the Server in Mininet topology. It's important to mention that to allow packets out of the network for Network Address Translation (NAT) to the GW from Mininet environment we must enable Media Access Control (MAC) address spoofing or so-called “*Promiscuous Mode*” which will allow changing the MAC address of outgoing packets sent from VMs.

IP Technology	Throughput		StDev	
	Mbps	Kbps	Mbps	Kbps
P2P	99.947	12493.375	1.358	169.750
IP Forwarding	94.332	11791.500	1.563	195.375
MPLS in Linux	0.011	1.413	10.879	1359.875
Cisco MPLS	92.814	11601.750	1.102	137.750
OpenFlow	99.948	12493.500	1.376	172.000

Figure 3.18: Network Throughput Depending on the IP Technology.

The results obtained are presented in *Figure 3.18* which allows us to state that the use of OpenFlow provides slightly higher throughput than the P2P link between two VMs. This is possibly because controller makes the forwarding decision based on network port number,

while both VMs would use their routing tables what involves additional processing and delay in result. We also have proven that MPLS support in Linux kernel 4.12 with IPRoute2 isn't effective in terms of speed between LERs due to lowest results in terms of throughput and highest StDev. We can see from the arithmetic mean based on the four measurements made for each method that, other than this fact, the most of results are very close to each other. We can also assume that the results obtained with iPerf3 are reliable from a sample of four measurements since the Standard Deviation (StDev) of the random variable is small for the remaining test cases. It also shows that IP traffic control technology operates unpredictably if we consider its throughput in OpenFlow. We can state this as usage of Layer 2 technology in P2P and OpenFlow gives different results than those expected and that MPLS in Linux should provide a more effective solution since HW routers are usually limited by their physical resources (Dawson, 2000).

3.2.2.2. Delay

The iPerf bandwidth measurement tool uses a TCP-based technique, but it also allows us to check the network parameters using UDP. UDP in contrast to TCP is a P2P protocol. It does not provide mechanisms for acknowledging the delivery of packets to the destination and retransmitting the lost data. In addition, messages can reach the destination out of order. However, it has one very important advantage over TCP which is its speed. This protocol is commonly used for the live streaming of video and Voice over Internet Protocol (VoIP) services where minimizing latency and jitter are the parameters that determine the transmission of data delivered to the destination.

We are going to test the same network topologies we used with previous use cases, but this time using UDP datagrams. We will check how the jitter value changes depending on the traffic control technology used. In addition, we will examine the impact of the above parameter on the transmission of data from several different ports at the same time. This test can simulate the actual situation when the transmission channel encounters packets from different sources. As before, we can treat the result obtained when two VMs were connected directly to one subnet while one port was used on the client and one on the server. The mean value calculated from the eight samples on the iPerf2 Client-side was 0.069 Milliseconds (ms) with a StDev of 0.087 ms and on the iPerf2 Server side was 0.029 ms with a StDev of 0.037 ms. The remaining results are presented in *Figure 3.21*.

For each of the analysed configurations, we have performed the same measurements eight times and then calculated the arithmetic mean of the obtained results for both iPerf2 Client and iPerf Server. This was done to eliminate the randomness of the obtained samples

and to provide repetitive and reliable data for further analysis. In addition, the StDev of the observed random variable is presented which determines how the results were differentiated. The symbols in parentheses “*p1*” and “*p10*” indicate the mode in which the datagram transmission was performed. Correspondingly this indicates that the transmission occurred between one client and one server (*p1*) or ten clients and one server (*p10*). The jitter values in *Figure 3.21* refer to a single bidirectional NIC to NIC port mapping, but to multiple port requests when stated as “*p10*”. To execute the test, we can use CLI as per highlighted syntax in *Figure 3.20* or jPerf2 GUI itself, where -*P* parameter stands for multiple simultaneous requests and result will be saved to *C2VM2toVM1xY.csv* (client-side delay) or *C2VM1toVM2x.csv* (server-side delay), *x* - letter of the alphabet, *Y* - concurrent test number.

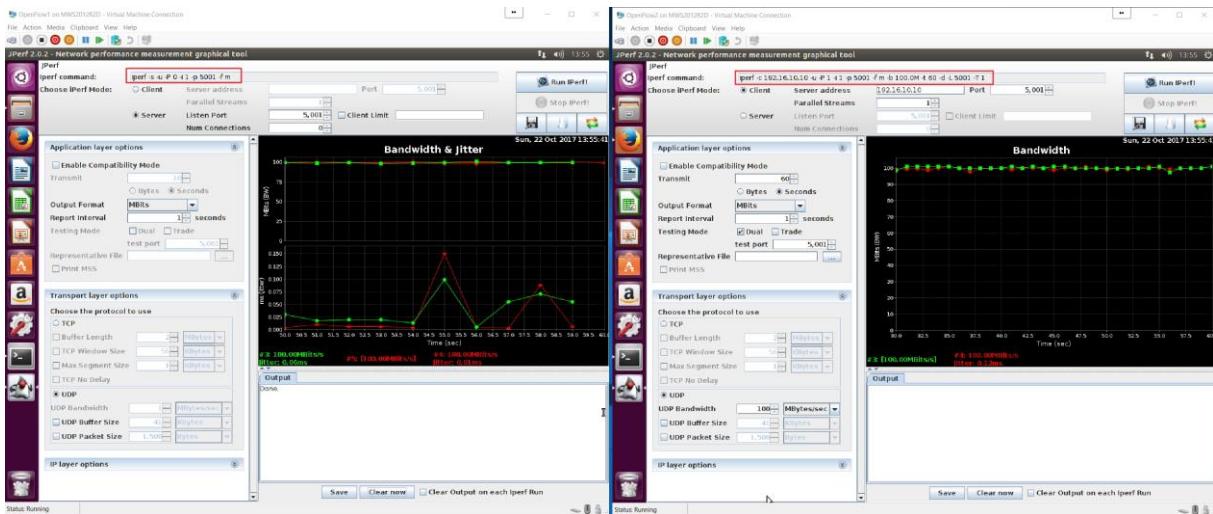


Figure 3.19: GUI of jPerf2 for Initial P2P Test.

During the *MPLS in Linux* scenarios, we have observed that iPerf2 Client was refused a connection to the Server for UDP tests. This is a well-known bug in this version reported by Yuri (2012), so we decided to use iPerf3 with below commands (Darbha, 2017).

= iPerf3 Server = <pre>sudo iperf3 -s</pre>
= iPerf3 Client = <pre>iperf3 -c 192.16.x.x -u -i 1 -t 60 -P 1 -d -b 100M tee C2VM2toVM1xY iperf3 -c 192.16.x.x -u -i 1 -t 60 -P 1 -d -b 100M tee C2VM1toVM2xY iperf3 -c 192.16.x.x -u -i 1 -t 60 -P 10 -d -b 100M tee C2VM2toVM1xY iperf3 -c 192.16.x.x -u -i 1 -t 60 -P 10 -d -b 100M tee C2VM1toVM2xY</pre>

Figure 3.20: Syntax of UDP MPLS in Linux Tests with iPerf3.

This will allow to test previously mentioned IP technologies to check what are the delays on

client and server side for one connection (p1) and during multiple simultaneous requests (p10).

IP Technology	Jitter (ms)		StDev (ms)	
	Client	Server	Client	Server
P2P	0.069	0.029	0.087	0.037
IP Forwarding (p1)	0.190	0.192	0.046	0.057
IP Forwarding (p10)	1.841	1.115	4.467	1.088
MPLS in Linux (p1)	0.000	0.000	0.000	0.000
MPLS in Linux (p10)	0.000	0.000	0.000	0.000
Cisco MPLS (p1)	0.389	0.260	0.331	0.098
Cisco MPLS (p10)	4.137	4.135	1.013	1.065
OpenFlow (p1)	0.279	0.363	0.733	0.915
OpenFlow (p10)	0.070	42.986	0.067	305.028

Figure 3.21: Network Delay Depending on the IP Technology.

The results showed a clear advantage to OpenFlow over tested technologies except for P2P connections established between two endpoints. Since all of our tests with MPLS in Linux didn't provide a summary of sent datagrams we need to invalidate the 0 ms results within our sample. We can tell that they all work quite similarly while one connection is established because the jitter values aren't high and StDev is below 1 ms. An interesting observation is that increasing the number of parallel transmissions causes a significant increase in the jitter value. For IP Forwarding and Cisco MPLS, it is approximately ten times the number of senders. However, this is probably because the server accepts datagrams as a group-send to a given port which in turn results in an irregularity of how packets reach the destination. In terms of performance, we can say that the optimal results were given by OpenFlow while iPerf Server's response was ten times higher than for Cisco MPLS. For the results obtained as a benchmark, we can assume that the acceptable jitter for video and voice transmissions over IP network must be below 30 ms. (Lewis and Pickavance, 2006). The results presented in *Figure 3.21* are at least seven times slower than that threshold value except for iPerf Server's response to multiple requests considering the small size of the network on which the tests were conducted.

3.2.2.3. Packet Loss

To investigate the amount of lost data during the transfer depends on the IP

technology we have used, the MGEN app. Although iPerf also provides statistics on lost packet transmission, MGEN will give us more options for specifying the nature of the traffic generated. It is more flexible in terms of packet size selection and transmission characteristics.

In this series of tests, network traffic was generated by sending UDP datagrams. The transmission quality was checked at four different levels of network load. We have changed the frequency with which the packets were sent as well as their size. VM1 and VM2 worked in a client-server architecture where the client was broadcasting the information for a period of 10 seconds using three UDP ports simultaneously, while the server listened at the same time on three different UDP ports. The “*Periodic*” option was used to determine traffic characteristics which means that packets of a fixed size and equal time intervals were generated (MGEN, 2017). The remaining parameters for the individual test cases are described below and the results are presented in *Figure 3.22* and *Figure 3.23*. For each test case, three tests were performed, and the results presented are their arithmetic mean.

- 50B-Medium: Packets of 50 Bytes (B) are sent with a frequency of 25000 times per second (P2P link utilized in about 96 %).
- 100B-Medium: 100 B packets are sent with a frequency of 12500 times per second (P2P link utilized in about 99 %).
- 1000B-High: 1000 B packets send with frequency 3000 times per second (P2P link utilized in about 100 %).
- 100B-Low: 100 B packets are sent with a frequency of 6000 times per second (P2P link utilized in about 98 %).

The results of the above scenarios were saved to *x-output-TEST-Y.csv* where: *x* - test number, *TEST* - test name, *Y* - concurrent test number. All of the scripts used to create output CSV samples on VM2 for both MGEN5 Client and MGEN Server are included in *Appendix 5*.

We also have ensured that the maximum link throughput was set to 100 Mbps which equals to 13,107.200 B per second (CheckYouMath, 2017) on the MGEN Server for receive buffer (*RXBUFFER*) and 1/3 of transmit buffer (*TXBUFFER*) per port on MGEN Client.

IP Technology	50B-Medium	100B-Medium	1000B-High	100B-Low
P2P	722.899	370.591	90.091	177.171
	750.000	375.000	90.000	180.000
IP Forwarding	359.021	372.638	90.065	180.730
	750.000	375.000	90.000	180.000
MPLS in Linux	330.304	368.782	90.093	178.272
	750.000	375.000	90.000	180.000
Cisco MPLS	740.579	375.005	90.101	180.730
	750.000	375.000	90.000	180.000
OpenFlow	625.459	367.162	87.066	4.675
	750.000	375.000	90.000	180.000

Figure 3.22: Packets Received in Comparison to Total Packets Transmitted.

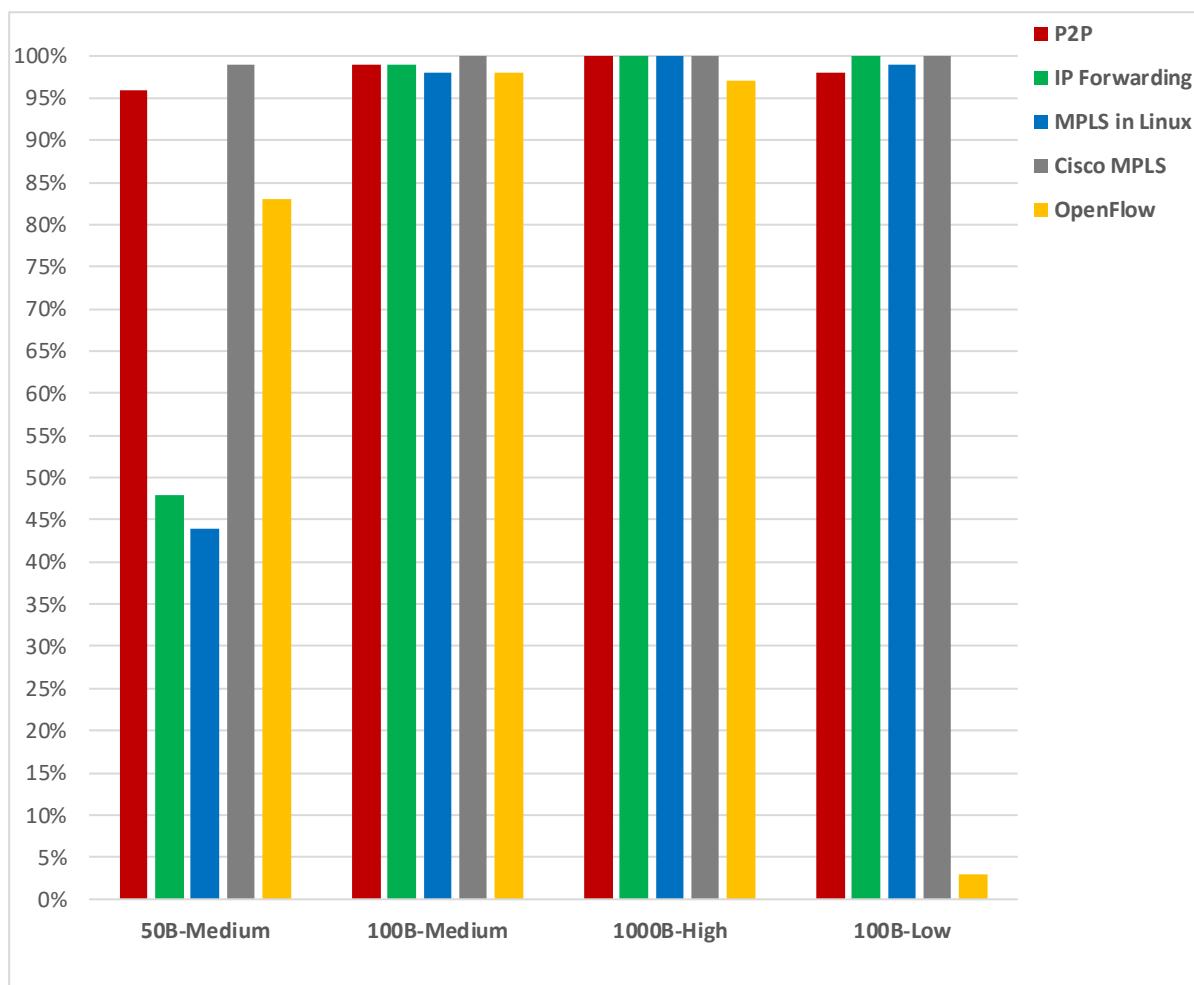


Figure 3.23: Correctly Delivered Data Depending on the IP Technology Used for Individual Test Cases.

Most selective test case consisted of the transmission of small packets with high frequency in such way that the link load oscillated around 100 %. The issue is that the limited frequency when datagrams can be sent before reaching saturation on the endpoint. The maximum value of this parameter which was used in this experiment was sending 50 B packets 25000 times per second in the first test.

This is shown in *Figure 3.23*, where the number of generated packages differs depending on the technology used in the 50B-Medium column. We also need to note that these are average values from three consecutive measurements and the deviation received in subsequent samples in the three remaining test cases was very small which means that the remaining tests were less significant. However, imperfections of the MGEN5 have been verified in terms of significant P2P link utilization of 96 %. In the remaining three cases large volumes of data sent out at lower frequencies with all the IP technologies reporting to be doing well except for OpenFlow when datagrams were 100 B and the rate was set to 6000 times per second. This would be caused by high rate of Packet-In messages to the controller as discussed by Zhihao and Wolter (2016). We can also see that MPLS in Linux reported the lowest value of 44 % during the high-frequency test case which means that it has been identified as the slowest performer taking into consideration that first test case which is most significant due to the high variation between the results.

3.2.2.4. RTT

The delay in relation to computer networks, often referred to as RTT, is the time it takes to send a signal from the sender to the receiver and then back to the sender. Normally this parameter is not used for QoS, but it can be used to compare the performance of the solutions we use with each IP technology.

Like in previous experiments, the results obtained for each test case are the mean values calculated from several, in this case, three measurements. During each test, we were sending 300 ICMP packets in size of 78 or 51200 B (50 KB) and saving them to CSV files using this command: *ping 192.16.x.x -c 300 -s SIZE / tee C3VM1toVM2xYSIZE.csv* (x - letter of the alphabet for case number, Y - concurrent test number).

Figure 3.24 shows the exact results and *Figure 3.25* displays the comparison between P2P and other IP technologies in percentage values.

IP Technology	78 B (ms)	50 KB (ms)	1 KB (ms)
P2P	0.548	1.031	0.375
IP Forwarding	1.581	11.787	NA
MPLS in Linux	1.425	NA	1.555
Cisco MPLS	1.155	11.465	NA
OpenFlow	0.566	1.192	NA

Figure 3.24: RTT Results in Milliseconds.

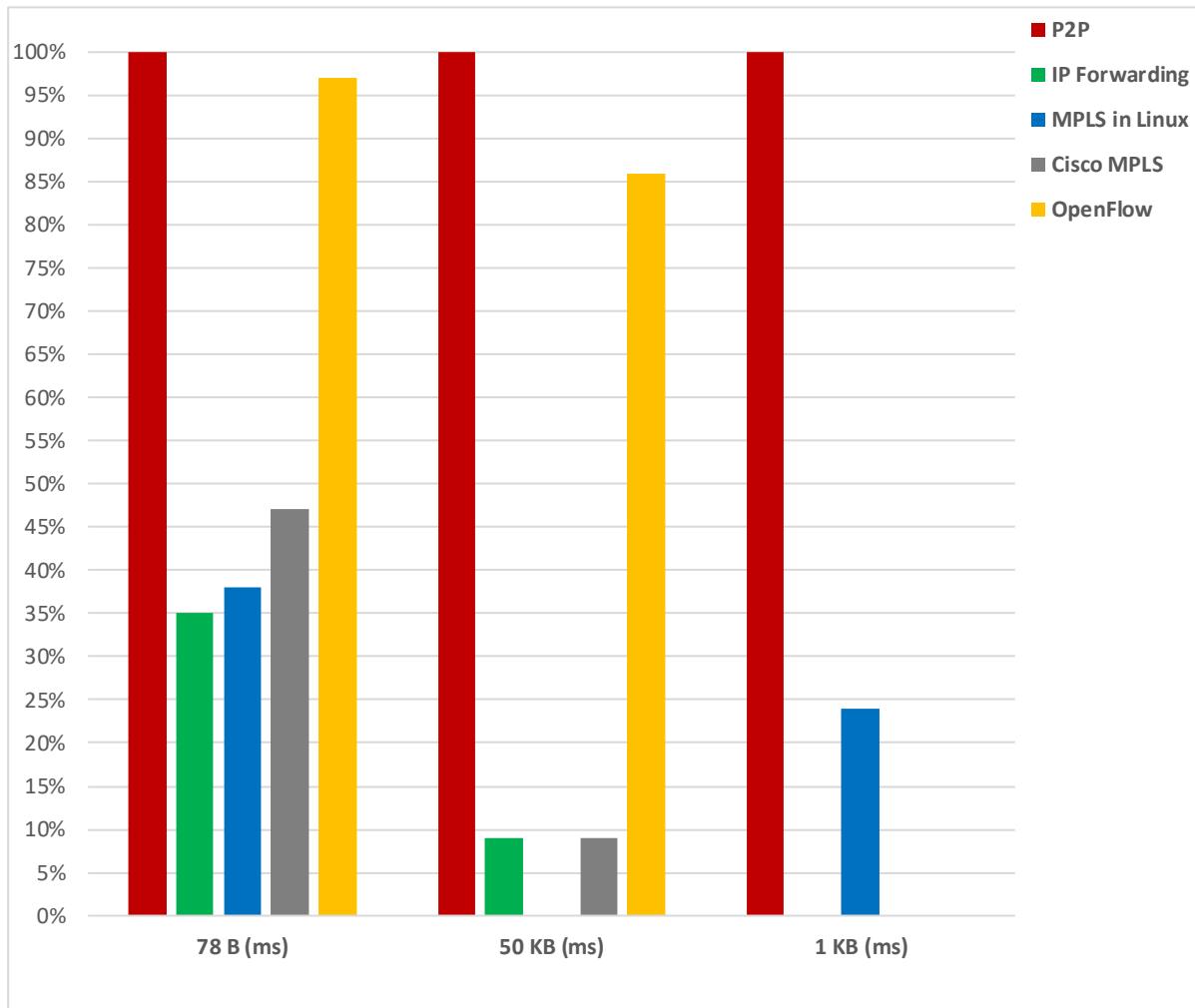


Figure 3.25: RTT Comparison in Percentage Values.

The results obtained by sending packets of 78 B allowed us to distinguish that IP Forwarding and MPLS in Linux are the worst performers, taking into consideration that during these tests the infrastructure load was quite small. Therefore, we have decided to use packets of 51200 B which allowed us to determinate the slowest IP technology while sending packets of 50 KB which appeared to be MPLS in Linux. This was investigated further with a smaller 1 KB (1024 B) packet which was the largest possible load in this situation and

introduced a higher delay than IP Forwarding and Cisco MPLS. The smallest delay, however, was achieved using the OpenFlow protocol for both packet sizes possibly because proactive approach to flow entries discussed in *Chapter 1.4.12* was used. This proves that MPLS in Linux delays are highest for larger packets while OpenFlow performs nearly as well as P2P with 85 % ratio in comparison to other IT technologies.

3.3. Scalability

At the beginning of this work, the hypothesis was that the switching of MPLS labels should be faster than routing based on IP headers, and OpenFlow protocol should outperform all tested technologies except when connecting two devices directly. The presented results contradict this in the case of MPLS in Linux in terms of performance, especially when transferring a high amount of data on a highly utilized link with the requirement for low delay.

However, when we reflect on what might be causing this we need to highlight that the network used for testing consisted of only five nodes, while the MPLS domain included three nodes: two LERs and one LSR.

On LER nodes labels are removed and this is a very CPU high intensive operation, while the benefits of using MPLS should be revealed when the packets flow through the LSR where the MPLS labels are switched. To see if the thesis is reflected we have conducted another experiment where we have decided to add an extra LSR node vertically on each side of the Dublin router and then connect the LERs to these devices.

The purpose of these scalability scenarios is to build similar networks with mixture of IP technologies to proof that they can be easily expanded by adding in extra nodes to the previously discussed topologies thought *Chapter 3.1* and *Chapter 3.2*. It will also allow to compare wheatear the delay would impact favourable OF protocol over MPLS solutions.

3.3.1. Configuration

For our environments, we will also use the same vSwitches and NICs as in *Chapter 3.1.1* for the edge networks while we will investigate the RTT like tests in *Chapter 3.2.2.4*. In the first topology, all three nodes will act as LSRs and two as LERs with a combination of Cisco 2801 and Cisco CSRs, as seen in *Figure 3.26*. The only differences in the second topology are that two directly connected LERs will be running MPLS in Linux rather than on Cisco routers, as seen in *Figure 3.27*. In the last topology, we are going to scale-up the Mininet environment by adding two extra switches, as seen in *Figure 3.28*.

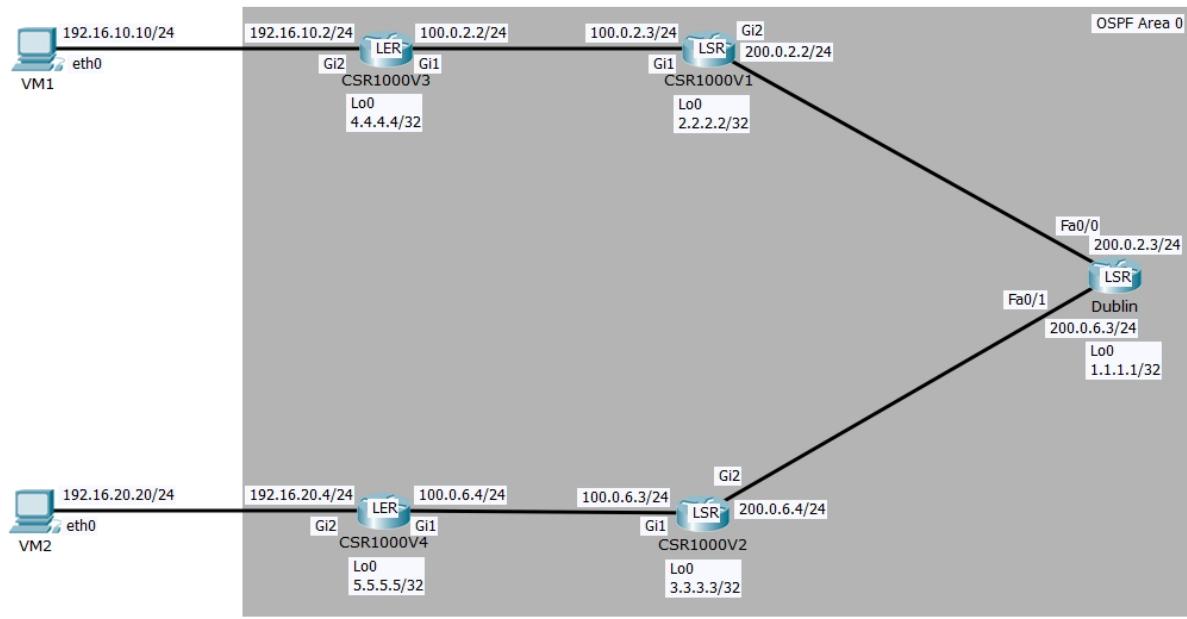


Figure 3.26: Three Cisco MPLS LSR Nodes and Two LER Nodes.

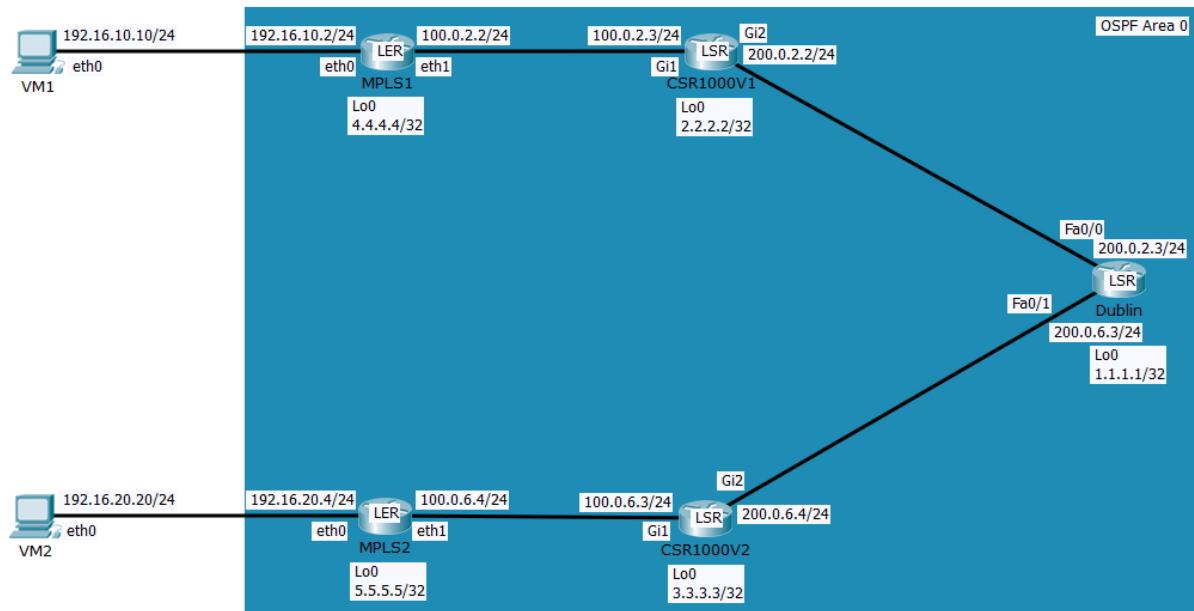


Figure 3.27: Three Cisco MPLS LSR Nodes and Two MPLS in Linux LER Nodes.

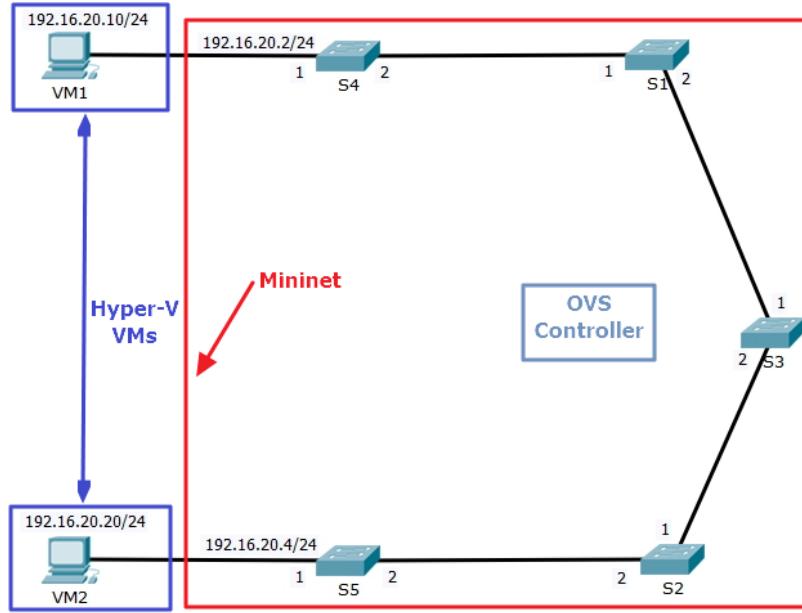


Figure 3.28: Mininet OpenFlow Scaled-Up Topology.

This time with MPLS we have used OSPF dynamic protocol as Interior Gateway Protocol (IGP) which was discussed by Perkin (2016) to exchange LDP binding information (Mert, 2017), as seen in *Figure 3.29*.

The screenshot displays four Cisco routers (CSR1000V1, CSR1000V2, CSR1000V3, CSR1000V4) running on MWS2012R2D virtual machines. Each router window shows two tables: the MPLS Forwarding Table and LDP Binding Information.

MPLS Forwarding Table:

Local Label	Outgoing Label or Tunnel Id	Prefix	Bytes Switched	Label	Outgoing interface	Next Hop
16	Pop Label	2.2.2.2/32	0	Fa0/0	200.0.2.2	
17	Pop Label	100.0.2.0/24	3706	Fa0/0	200.0.2.2	
18	Pop Label	3.3.3.3/32	0	Fa0/1	200.0.6.4	
19	Pop Label	100.0.6.0/24	25854	Fa0/1	200.0.6.4	
20	20	4.4.4.4/32	0	Fa0/0	200.0.2.2	
21	21	5.5.5.5/32	0	Fa0/1	200.0.6.4	
22	22	192.16.10.0/24	49865334	Fa0/0	200.0.2.2	
23	23	192.16.20.0/24	49887978	Fa0/1	200.0.6.4	

LD Bindings:

Label	Local Label or Tunnel Id	Prefix	Bytes Switched	Label	Outgoing interface	Next Hop
16	2.2.2.2/32	CSR1000V2#	21051	G12	200.0.6.3	
17	2.2.2.2/32	CSR1000V2#	3264	G12	200.0.6.3	
18	100.0.2.0/24	CSR1000V2#	1078	G12	200.0.6.3	
19	100.0.2.0/24	CSR1000V2#	22632	G11	100.0.6.4	
20	4.4.4.4/32	CSR1000V2#	49240004	G11	200.0.6.3	
21	5.5.5.5/32	CSR1000V2#	49436934	G12	200.0.6.3	
22	192.16.10.0/24	CSR1000V2#	49235398	G11	100.0.6.4	
23	192.16.20.0/24	CSR1000V2#				

Figure 3.29: MPLS Forwarding Tables and LDP Binding Information on Cisco Nodes.

For Linux nodes, we have first implemented Quagga Routing Suite 0.99.24.1 (Jakma, 2011) and Label Distribution Protocol daemon (LDPd) 1.0.20160315 to redistribute MPLS labels via IPRoute2 (Westphal, 2016). However, after discussion with the Quagga-LDPd project owner, we have decided to move to a newer and updated repository called FRRouting (FRR) 3.1 which is the precursor of Quagga. This IP Routing Protocol Suite has the LDP daemon embedded (FRRouting, 2017) and well-tested compliance with IPv4 (OpenSourceRouting, 2017), so we have used McLendon's (2017) instructions for our process of installation.

On CSRs we need to make sure that MPLS will be enabled which can be done by starting the 60-day trial period via *license boot level premium* command (Fryguy, 2013).

In the Linux environment, label 21 was used to forward packets to 192.16.10.0/24 network and label 23 for 192.16.20.0/24 network, as seen in *Figure 3.30*.

Figure 3.30: MPLS Forwarding Tables and LDP Binding Information in Linux Environment.

Relevant configurations which will allow reproducing the set up can be found in *Appendix 6*.

3.3.2. Results

Since we need to have some initial benchmark to compare against, we have decided to use IP Forwarding with static routes on all nodes and RTT packet sizes of 78 B, 50 KB, and 1 KB. Output data of three concurrent tests were saved into CSV files $C4VM1toVM2xYSIZE.csv$ (x - letter of the alphabet for case number, Y - concurrent test number). The results of the above test cases are shown in *Figure 3.31* where *Figure 3.32* displays comparison between IP Forwarding and MPLS in Cisco and Linux topologies.

IP Technology	78 B (ms)	50 KB (ms)	1 KB (ms)
IP Forwarding	2.040	12.610	2.452
Cisco MPLS	2.716	16.213	2.922
MPLS in Linux	2.199	14.816	2.697
OpenFlow	0.473	4.103	2.538

Figure 3.31: RTT Results for Scaled-Up Environment.

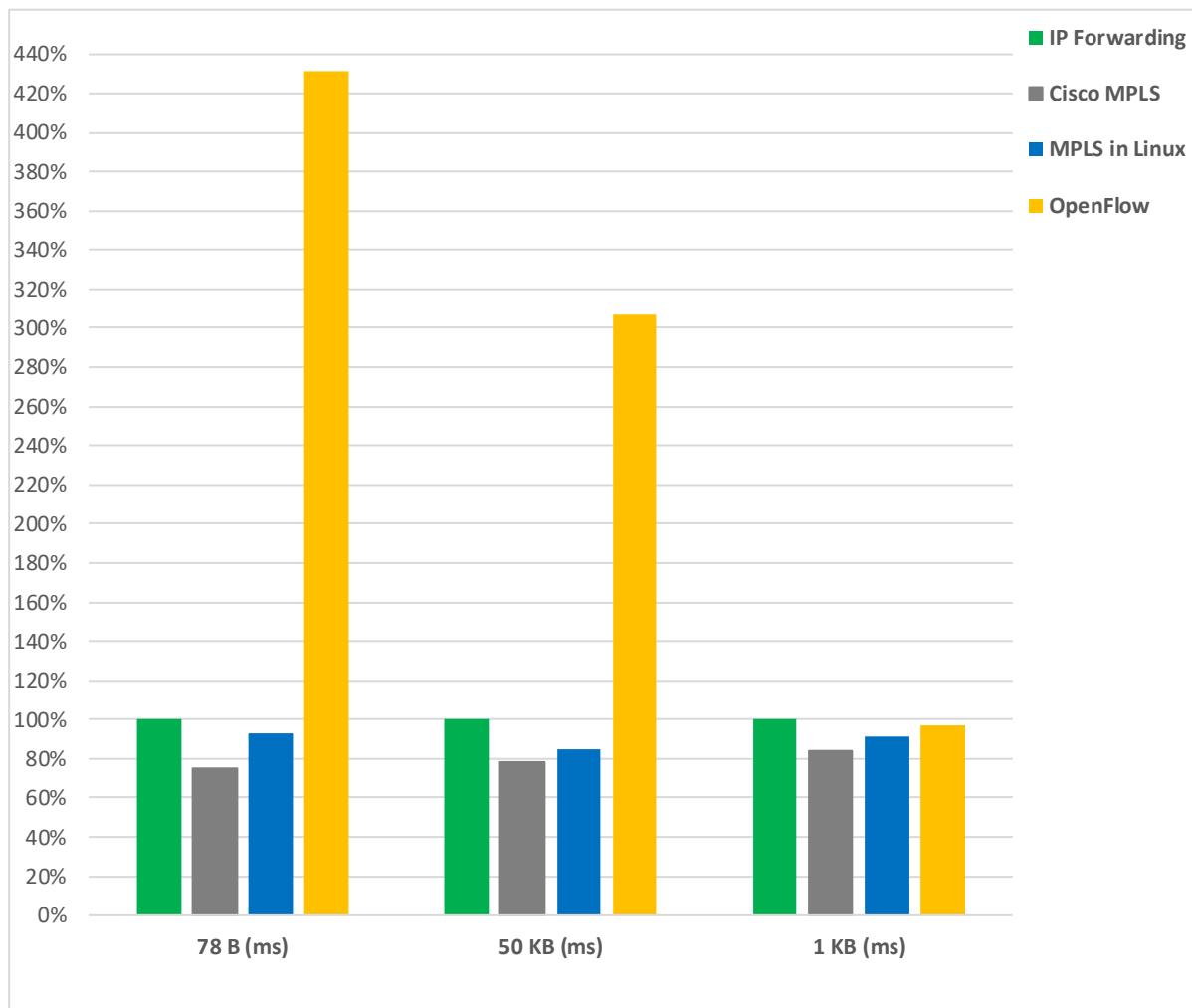


Figure 3.32: RTT Comparison in Percentage Values for Scaled-Up Environment.

From the tests, we can make a succinct conclusion that OpenFlow outperformed all other IP technologies, while LDP implemented together with OSPF and FFR on Linux provided better results than MPLS on Cisco routers.

This also has proven that all tested technologies can be easily scaled-up within the “*test-bed*” no matter what routing method has been used. However, in terms of manageability, it’s always easier to manage a dynamic protocol over static routing as the topology would adapt to changes automatically independent of the size of the network (Cisco, 2014).

3.4. QoS

According to Cisco (2014), the ability to provide QoS across the network consists of:

- Traffic Classification - identifying the received traffic so later it's possible to form its characteristics to guarantee its priority or fixed bandwidth.
- Congestion Management - limiting the impact of unmanageable protocols and apps on another network traffic.
- Congestion Avoidance - proactively preventing saturation or explicitly imposing policies that prevent a given type of traffic from saturating the entire available bandwidth.

Type of Service (ToS) is a standard definition of the traffic class of an IP packet as described in RFC 1349. It defines six types of traffic and assigns each of them a value written to bits 3-6 in IPv4 packet header field (Almquist, 1992). The Precedence field is a value in bits 0-2 of the same 1 B field that specifies the priority of a given traffic. The Differentiated Services Codepoint (DSCP) defined in the RFC 2474 (Nicholas et al., 1998) describes how to use the same 1 B field in the IPv4 header to divide traffic into classes using 0-5 bits leaving the other two unused.

By identifying and marking packets that match specific criteria devices, we can differentiate the treatment of network traffic. It is generally assumed that, intelligent network devices that are closest to end-user denote different types of traffic with different ToS, Precedence or DSCP labels to use these labels for traffic prioritization, bandwidth guarantee or its limitation while forwarding the packets.

For our experiments with MPLS, we are going to use Class-Based Weighted Fair Queuing (CBWFQ) where we will first classify traffic that will be treated differently, in other words, divided into classes. Cisco IOS offers several classification methods for Enhanced Interior Gateway Routing Protocol (EIGRP) to match the traffic with Access Control List (ACL) to a specific host on port or by using Network-Based Application Recognition (NBAR) to shape up specific QoS policies with pre-defined MPLS EXP classes (Sogay, 2013). These, in turn, can be associated with the specific interface on the CE nodes and Differentiated Services (DiffServ) for MPLS label forwarding (Le Faucheur et. al, 2002) via VPN tunnel to PE side while using BGP to create so-called Multiprotocol Border Gateway Protocol (MP-BGP) as discussed by Molenaar (2017). NBAR will be used on specific device interfaces to allow for examination of traffic in terms of packet transmitted and what type of application this traffic is associated with (Hebert, 2018).

In SDN however, according to Slattery (2014) it is possible to use dynamic QoS policies per app on the edge nodes which would be forwarded through the core network to provide SDN enhancement such as for Skype (Bauer, 2015) and implement Group Based Policies (GBP) as discussed in PR Newswire (2015).

3.4.1. Configuration

3.4.1.1. MPLS

Cisco topology consists of two CE routers: CSR1000V3 and CSR1000V4, two PE routers: CSR1000V1 and CSR1000V2 as well as one Provider (P) Cisco 2801 router (Dublin). OSPF is enabled on all ISP devices in *network 0.0.0.0 255.255.255.255 area 0* and routing between PE and CE nodes is achieved with EIGRP, as seen in *Figure 3.33*.

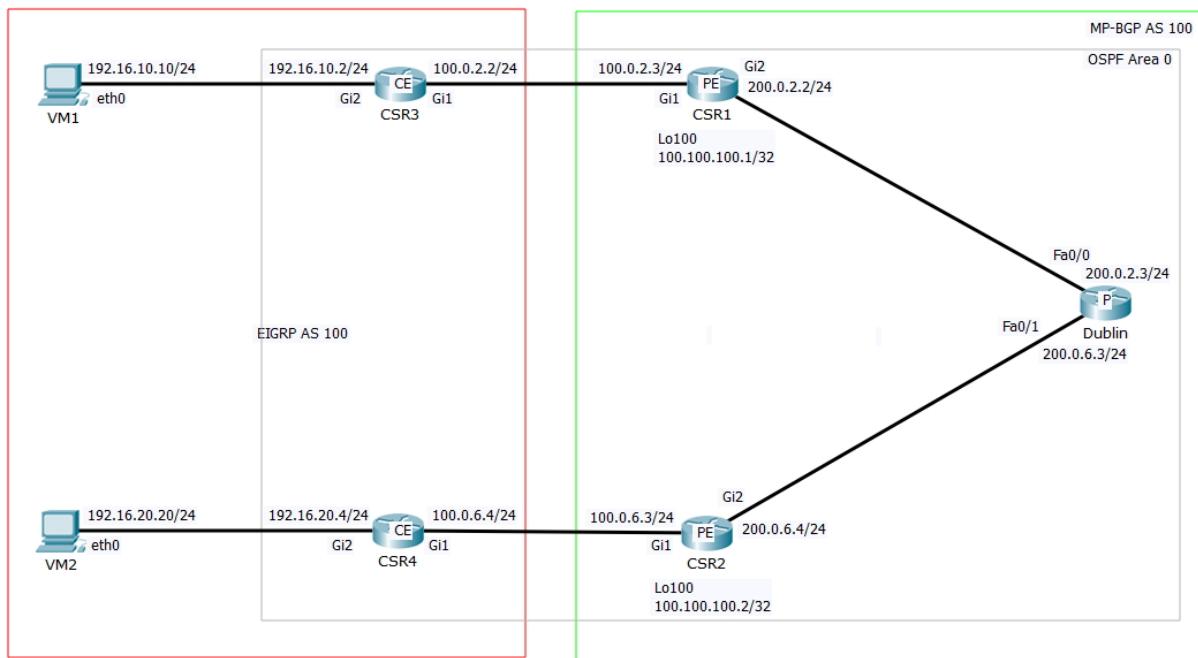


Figure 3.33: Two Cisco PE MP-BGP Nodes and Two CE EIGRP Nodes.

All relevant configurations are attached in *Appendix 7*. However, we will explain key concepts of configuration before proceeding further with MPLS QoS setup via VPN.

We can see from *Figure 3.33* that there is no Loopback (Lo) interface configured for LDP neighbourship on the Dublin router which might cause a loss of connectivity if the highest IP address interface fails which in these circumstances is set to be the LSR interface IP address 200.0.6.3/24. Routers are generating specific label ranges as follows: CSR1000V1 1000-1999, Dublin 2000-2999 and CSR1000V2 3000-3999. Devices in the local subnets are set not to generate labels locally with *implicit-null* as these are used to facilitate the transport of packets within an MPLS domain.

```

Dublin>en
Dublin#show mpls ldp bindings local
lib entry: 100.100.100.1/32, rev 8
    local binding: label: 2001
lib entry: 100.100.100.2/32, rev 6
    local binding: label: 2000
lib entry: 192.168.2.0/24, rev 10(no route)
    local binding: label: 2002
lib entry: 200.0.2.0/24, rev 14
    local binding: label: imp-null
lib entry: 200.0.6.0/24, rev 4
    local binding: label: imp-null
Dublin#show mpls forwarding-table
Local      Outgoing   Prefix       Bytes Label   Outgoing   Next Hop
Label     Label or Tunnel Id Switched   interface
2000     Pop Label  100.100.100.2/32 36425    Fa0/1     200.0.6.4
2001     Pop Label  100.100.100.1/32 35201    Fa0/0     200.0.2.2
2002     No Label   192.168.2.0/24   0        drop
Dublin#

```

```

CSR1000V1 on MWS2012R2D - Virtual Machine Connection
File Action Media Clipboard View Help
CSR1000V1>
CSR1000V1>
CSR1000V1>
CSR1000V1>
CSR1000V1>
CSR1000V1>show mpls ldp bindings local
lib entry: 100.100.100.1/32, rev 2
    local binding: label: imp-null
lib entry: 100.100.100.2/32, rev 4
    local binding: label: 1000
lib entry: 192.168.2.0/24, rev 14(no route)
    local binding: label: imp-null
lib entry: 200.0.2.0/24, rev 10
    local binding: label: imp-null
lib entry: 200.0.6.0/24, rev 10
    local binding: label: 1001
CSR1000V1>show mpls forwarding-table
Local      Outgoing   Prefix       Bytes Label   Outgoing   Next Hop
Label     Label or Tunnel Id Switched   interface
1000     100.100.100.2/32 0        G11      200.0.2.3
1001     Pop Label  200.0.6.0/24 0        G12      200.0.2.3
1002     No Label   100.0.2.0/24[V1] 0        aggregate/cust
1004     No Label   192.16.10.0/24[V1] 25072    G11      100.0.2.2
CSR1000V1#

```

```

CSR1000V2 on MWS2012R2D - Virtual Machine Connection
File Action Media Clipboard View Help
CSR1000V2>
CSR1000V2>
CSR1000V2>
CSR1000V2>
CSR1000V2>
CSR1000V2>show mpls ldp bindings local
lib entry: 100.100.100.1/32, rev 2
    local binding: label: 3000
lib entry: 100.100.100.2/32, rev 4
    local binding: label: imp-null
lib entry: 192.168.2.0/24, rev 14(no route)
    local binding: label: imp-null
lib entry: 200.0.2.0/24, rev 8
    local binding: label: 3001
lib entry: 200.0.6.0/24, rev 10
    local binding: label: imp-null
CSR1000V2>show mpls forwarding-table
Local      Outgoing   Prefix       Bytes Label   Outgoing   Next Hop
Label     Label or Tunnel Id Switched   interface
3000     2001      100.100.100.1/32 0        G12      200.0.6.3
3001     Pop Label  200.0.6.0/24 0        G12      200.0.6.3
3003     No Label   100.0.6.0/24[V1] 0        aggregate/cust
3005     No Label   192.16.20.0/24[V1] 30380    G11      100.0.6.4
CSR1000V2#

```

Figure 3.34: MPLS Domain Label Bindings.

We have used MP-BGP to exchange CE labels between CSR1000V1 and CSR1000V2 with VRF “*cust*” as well as with Route Distinguisher (RD) and Route Target (RT) of 100:1, as seen in *Figure 3.35*.

```

CSR1000V1 on MWS2012R2D - Virtual Machine Connection
File Action Media Clipboard View Help
CSR1000V1>
CSR1000V1>
CSR1000V1>
CSR1000V1>
CSR1000V1>
CSR1000V1>Route Distinguisher: 100:1 (cust)
100.0.2.0/24 0.0.0.0 1002:nolabel(cust)
100.0.6.0/24 100.100.100.2 1003:nolabel
192.16.10.0 100.0.2.2 1004:nolabel
192.16.20.0 100.100.100.2 1005:nolabel
CSR1000V1>show ip vrf detail cust
URF cust (URF Id = 1); default RD 100:1; default UPMID <not set>
Old CLI format, supports IPv4 only
Flags: 0xC
Interfaces:
G11
Address family ipv4 unicast (Table ID = 0x1):
Flags: 0x0
Export UPM route-target communities
    RT:100:1
Import UPM route-target communities
    RT:100:1
No Import route-map
No global export route-map
No export route-map
URF label distribution protocol: not configured
URF label allocation mode: per-prefix
CSR1000V1#

```

```

CSR1000V2 on MWS2012R2D - Virtual Machine Connection
File Action Media Clipboard View Help
CSR1000V2>
CSR1000V2>
CSR1000V2>
CSR1000V2>
CSR1000V2>
CSR1000V2>Route Distinguisher: 100:1 (cust)
100.0.2.0/24 100.100.100.1 1002:nolabel/1002
100.0.6.0/24 0.0.0.0 3003:nolabel(cust)
192.16.10.0 100.100.100.1 1004:nolabel
192.16.20.0 100.0.6.4 3005:nolabel
CSR1000V2>show ip vrf detail cust
URF cust (URF Id = 1); default RD 100:1; default UPMID <not set>
Old CLI format, supports IPv4 only
Flags: 0xC
Interfaces:
G11
Address family ipv4 unicast (Table ID = 0x1):
Flags: 0x0
Export UPM route-target communities
    RT:100:1
Import UPM route-target communities
    RT:100:1
No Import route-map
No global export route-map
No export route-map
URF label distribution protocol: not configured
URF label allocation mode: per-prefix
CSR1000V2#

```

Figure 3.35: BGP VPN Label Bindings.

This means that a VPN label will be used to reach the MPLS domain on the PE side where it will be stripped out and replaced with a transport label which will be used to forward it to

specific MPLS interface associated.

When we execute an ICMP request from VM1 to VM2 the packet reaches CSR1000V1 on VRF interface (Gi1) via VPN, the routing table for VPN “*cust*” is checked to match the destination address. This determinates that network 192.16.20.0 would be encapsulated with outgoing label 3005 and the next hop IP address is 100.100.100.2 (*Figure 3.35*). This packet next traverses across the MPLS domain to the Dublin router with transport label 2000 where its swapped and sent via an outgoing interface to CSR1000V2 with a VPN label set to 3005 (*Figure 3.34*). This label value is used to send a packet to 100.0.6.4/24 (Gi1) on CSR1000V4 where it’s removed before delivery to the destination according to the switching table as VM2 is directly connected to this node.

For our policies, we have decided to use File Transfer Protocol (FTP) and Hypertext Transfer Protocol (HTTP) which will be used to connect servers built on VM2 with vsFTPD 3.0.3 on port 21 (Anderson, 2016) and Apache 2.4.18 (Ellingwood, 2017) on port 80.

To start with the creation of the policy we first tested uncongested bandwidth between VM1 and VM2 to calculate the average of the maximum connection throughput. This was measured based on three concurrent download requests with *ftp get* and *wget* commands for four previously created sample files in size of 1 MB, 10 MB, 100 MB and 1000 MB. Mean values of captured results of FTP and the HTTP download speeds in Megabytes per second (MB/s) are presented in *Figure 3.36*.

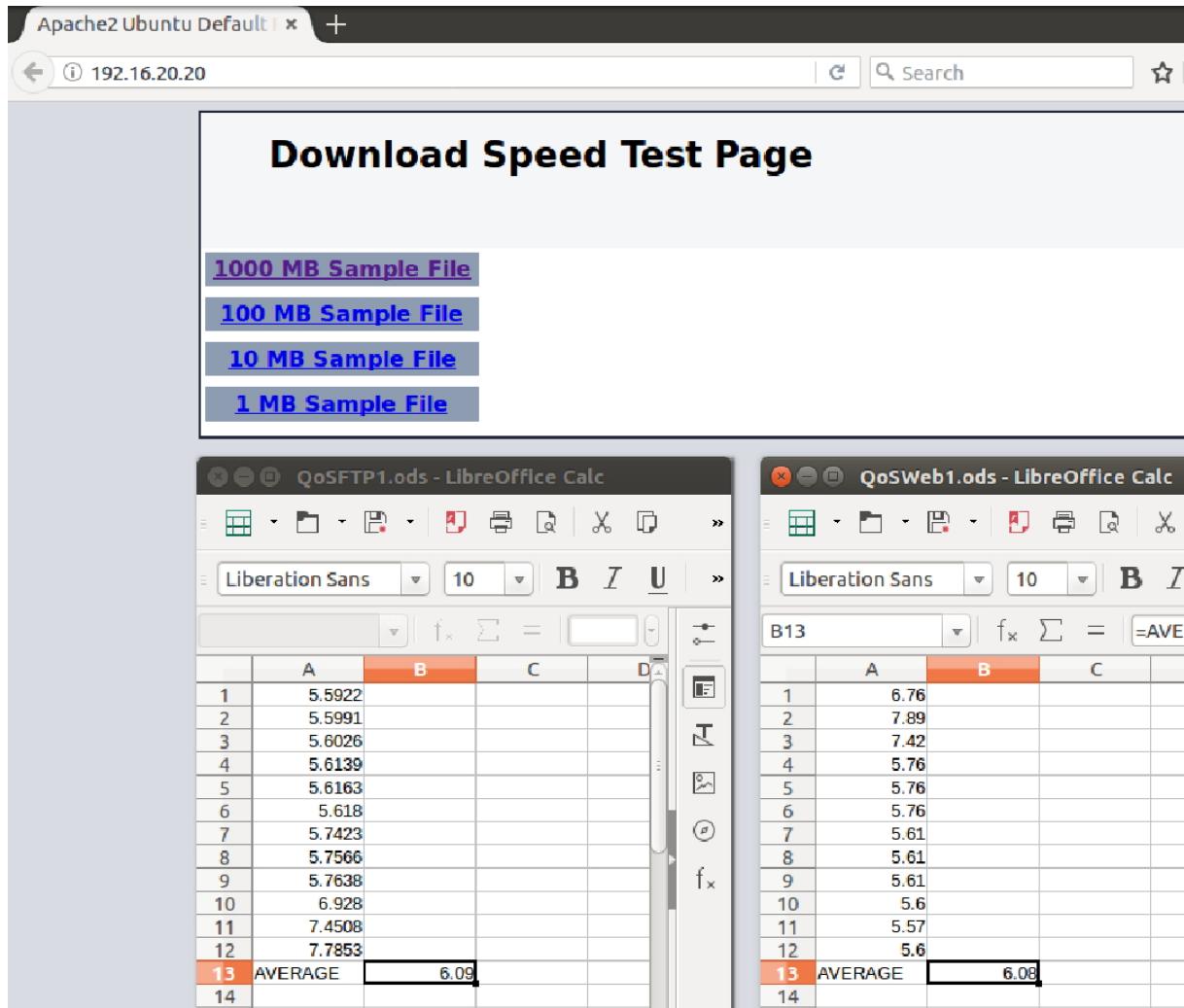


Figure 3.36: Mean Values of Throughput Test of FTP and HTTP Servers.

This value of 6 Mbps was used to limit the bandwidth to 6144 Kbps on the interfaces for outbound traffic where we have applied QoS policies. We are going to use the marking scheme discussed by Sogay (2013) which displays mappings between DSCP and MPLS EXP bits, as seen in *Figure 3.37*.

Application	CoS=IPP	AF	DSCP	EXP
Best Effort	0	0	0	0
Scavenger	1	CS1	8	1
Bulk Data	1	AF11	10	1
	1	AF12	12	1
	1	AF13	14	1
Network Mgmt.	2	CS2	16	2
Transaction Data	2	AF21	18	2
	2	AF22	20	2
	2	AF23	22	2
Call Signaling	3	CS3	24	3
Mission-Critical	3	AF31	26	3
Streaming Video	3	AF32	28	3
	3	AF33	30	3
	4	CS4	32	4
Interactive Video	4	AF41	34	4
	4	AF42	36	4
	4	AF43	38	4
	5	CS5	40	5
Voice	5	EF	46	5
Routing	6	CS6	48	6
	7	CS7	56	7

Figure 3.37: Mappings between DSCP and MPLS EXP Bits (Sogay, 2013)

To provide the QoS we have implemented a maximum FTP data transfer of 1024 Kbps (1.024 Mbps) with the same rate of guaranteed transfer for HTTP data. This means that if there would be a simultaneous transfer of FTP and HTTP data, then FTP will use approx. 1 Mbps while remaining transfer will be given to HTTP which will not be less than 1 Mbps, but not gather more than approx. 5 Mbps.

On CSR1000V3 FTP-data packets are marked as DSCP EF and HTTP is marked as DSCP AF41. Next, when packets reach the CSR1000V1 via VPN, the markings are associated with EXP bits together with their policies before they enter the MPLS domain. After that, when data leaves the PE and moves across the VPN to the other side on CE they are again associated with their DSCP mappings for the relevant policies before they reach the destination.

According to Taleqani (2017), a VPN tunnel built with FRR will not send updates and therefore we have decided to use TE tunnels rather than GRE tunnels with Resource Reservation Protocol (RSVP) to exchange the LSPs in the PE and MPLS LDP in the CE as discussed by Bertrand (2014). In this MPLS on Linux test case we have used two CE nodes with FRR, two PE nodes with CSR IOS and one Cisco 2801 HW router to provide QoS with a tunnel maximum and interface with reserved bandwidth, as seen in *Figure 3.38*.

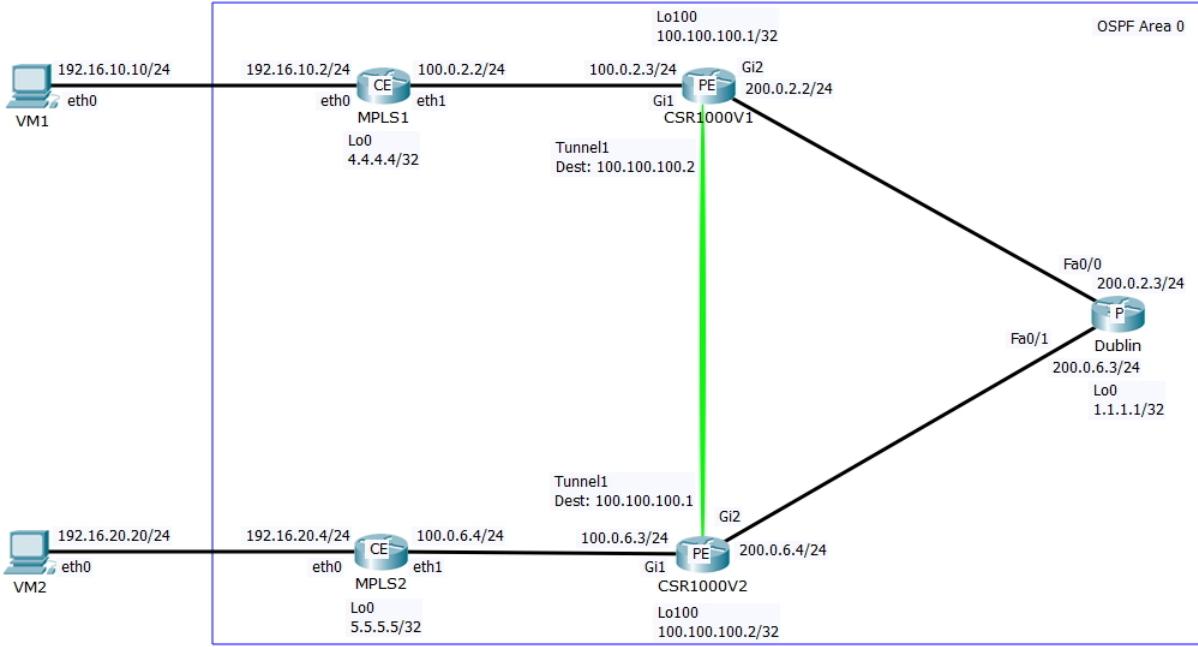


Figure 3.38: MPLS-TE Tunnels with RSVP Topology.

We have set up unidirectional tunnels from CSR1000V1 to Dublin and to CSR1000V2 on next hop interface basis as well as the way back from CSR1000V2 to CSR1000V1 via Dublin.

All routers also operate OSPF protocol in area 0 to exchange their MPLS labels while FRR devices use *implicit-null* labels for performance reasons which are popped out on CSRs and replaced with *explicit-null* labels within the MPLS domain or in this situation sent via TE Tunnel with RSVP, as seen in *Figure 3.39*.

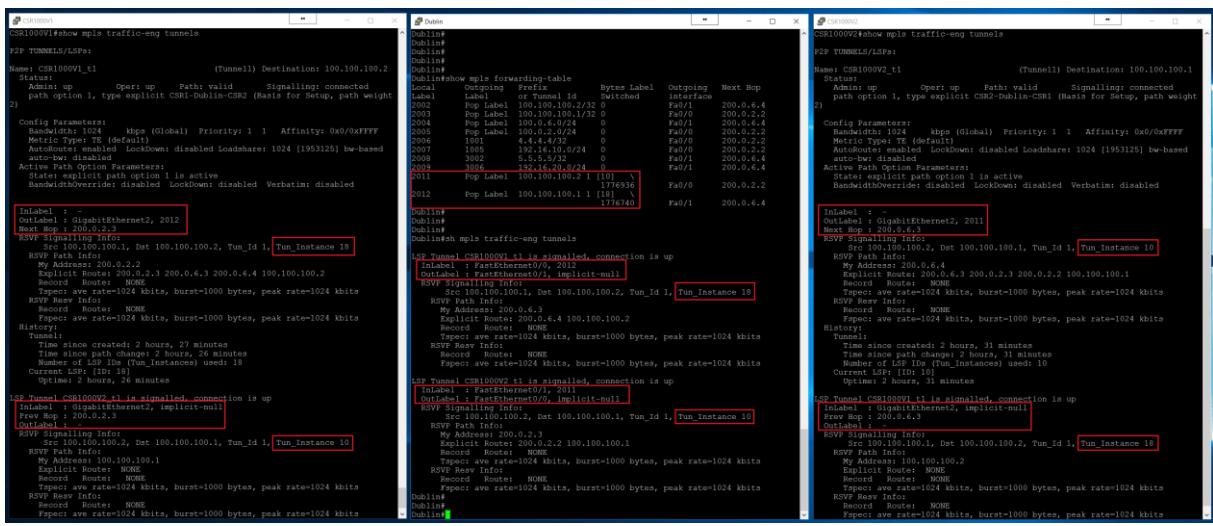


Figure 3.39: MPLS-TE Tunnels Signalling.

This way LDP binding relationships between neighbours are achieved with MPLS labels, but tunnel endpoints are set up with RSVP signalling between *Lo100* on CSR1000V1 and CSR1000V2 nodes. This means that the traffic from VM1 travels across to MPLS1, then the

local label is used to communicate with CSR1000V1 which is popped out and the packet is encapsulated into the tunnel to CSR1000V2. This tunnel uses signalling archived with an LDP relationship to *Lo100* endpoint, so it's using the local label 2012 to move via the MPLS domain with explicit route to the Dublin (Tunnel Instance 18) and then to CSR1000V2, while the other reply packets are traversing back with the label 2011 to reach the P router (Tunnel Instance 10) and then CSR1000V1.

Popelnjak (2007) has discussed the ten myths of MPLS-TE and in the fifth point, he mentioned that MPLS bandwidth reservations are purely used as an accounting mechanism to not allow oversubscription on the link. This explains why we cannot shape the traffic policies, but we can provide QoS for specific paths based on the maximum bandwidth which will not be greater than the RSVP value on the interface and it will not be less than the reserved parameter for the TE tunnel.

Taking these facts into consideration we have limited interface bandwidth to 6144 Kbps on all interfaces which are using MPLS-TE and we have reserved 1024 Kbps to the TE tunnel between CSRs. Next, we will execute a series of test cases to see how traffic behaves with specific samples to see the behaviour of throughput parameter and its fluctuation.

3.4.1.2. OpenFlow

In OF we are going to explore REST to configure QoS based on the type of data and bandwidth limit per flow, with the use of DiffServ as well as with Meter Table and a CPqD SW switch (CPqD GitHub, 2018)

The first test case will look at the reservation of the bandwidth to shape the traffic on the server side (s1-eth2) to 1 Mbps for UDP and FTP, 10 Mbps for HTTP and limits to 50 % in case of congestion with three Linux Hierarchical Token Buckets (HTBs) as discussed by Benita (2005), as seen in *Figure 3.40*.

Queue	Type	Port	Max Rate (bps)	Min Rate (bps)
1	UDP	5000	1000000	500000
0	TCP	21	1000000	N/A
1	TCP	80	10000000	5000000

Figure 3.40: Linux HTB Queues for FTP and Web Server Scenario.

All of the relevant configurations are included in *Appendix 9*.

We have started the Topo with two hosts and one switch as well as set the OF13 for the controller to access OVSDDB via port 6632, so after we run custom apps on the controller we will see that our node joins the QoS switch, as seen in *Figure 3.41*.

The figure consists of three vertically stacked terminal windows. The top window shows the Ryu controller being configured with a command like `sudo mn --mac --switch=ovs --ipphase=192.16.20.0/24 --controller remote` and then setting the bridge protocol to OpenFlow13 using `ovs-vsctl set bridge s1 protocol=OpenFlow13`. The middle window shows the Ryu controller starting up with logs indicating the loading of various apps and the start of a WSGI server on port 8080. The bottom window shows the ovs-vsctl command being run to show the connection between the Ryu controller and the OVSDB database.

```

ryu@Ryu:~$ sudo mn --mac --switch=ovs --ipphase=192.16.20.0/24 --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6653
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> xterm s1
mininet> 

```

```

root@Ryu:~# sudo su
root@Ryu:/home/ryu# ovs-vsctl set bridge s1 protocol=OpenFlow13
root@Ryu:/home/ryu# ovs-vsctl set-manager ptcp:6632
root@Ryu:/home/ryu# 

Node: s1 (root)
root@Ryu:~# ovs-vsctl set bridge s1 protocol=OpenFlow13
root@Ryu:/home/ryu# ovs-vsctl set-manager ptcp:6632
root@Ryu:/home/ryu# 

root@Ryu:/home/ryu# Run 'do-release-upgrade' to upgrade to it.

Last login: Sat Jan 13 13:27:16 2018 from 192.16.20.1
ryu@Ryu:~$ sudo su
[sudo] password for ryu:
root@Ryu:/home/ryu# curl -g -X PUT -d '{"tcp:127.0.0.1:6632"}' http://127.0.0.1:80
80/v1.0/conf/switches/0000000000000001/ovsdb_addr
root@Ryu:/home/ryu# ovs-vsctl show
691b2ba-a779-41cc-a4ef-8885c545fec1
    Manager "tcp:6632"
    Bridge "s1"
        Controller "tcp:127.0.0.1:6653"
            is_connected: true
            Controller "ptcp:6654"
            fail mode: secure
            Port "s1-eth1"
                Interface "s1-eth1"
            Port "s1-eth2"
                Interface "s1-eth2"
            Port "s1"
                Interface "s1"
                    type: internal
            ovs_version: "2.2.2"
root@Ryu:/home/ryu# 

```

Figure 3.41: Connecting Controller to OVSDB.

There are no entries in the QoS or Queue tables and only default entries done by the Ryu apps during *pingall* are in the flow tables, as seen in *Figure 3.42*.

The figure shows a terminal window displaying the results of several commands: `ovs-vsctl list qos`, `ovs-vsctl list queue`, `tc class show dev s1-eth2`, and `ovs-ofctl -O OpenFlow13 dump-flows s1`. The output shows that there are no entries in the QoS or Queue tables, and the flow table for interface s1-eth2 contains three default entries (xid=0x2) with priority values of 65535 and 0.

```

root@Ryu:/home/ryu# ovs-vsctl list qos
root@Ryu:/home/ryu# ovs-vsctl list queue
root@Ryu:/home/ryu# tc class show dev s1-eth2
root@Ryu:/home/ryu# ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
  cookie=0x0, duration=713.635s, table=0, n_packets=0, n_bytes=0, priority=65535,
  dl_dst=01:80:c2:00:00:0e, dl_type=0x88cc actions=CONTROLLER:65535
  cookie=0x0, duration=713.635s, table=0, n_packets=0, n_bytes=0, priority=0 actions=goto_table:1
  cookie=0x0, duration=713.636s, table=1, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER:65535
root@Ryu:/home/ryu# 

```

Figure 3.42: List of QoS and Queue Tables and S1 Flow Entries.

Next, we have sent the setting of the queue to the controller for interface s1-eth2, as seen in *Figure 3.43*.

Figure 3.43: HTB Queue Settings.

Now, when we send settings of the QoS flows as per their entries in the Flow Table 0 we can see that they are *goto* instructions on them pointing the traffic to the controller, while all priorities for port destinations 21, 5000 and 80 are set to 1 with the destination IP address of 192.16.20.2, as seen in *Figure 3.44*.

Figure 3.44: QoS Settings and S1 Flow Entries.

This way Ryu apps running on the NBI decide with the use of OVSDB QoS and Queue tables based on the flow entries in the switch.

Since our previous test case isn't scalable due to the increased amount of flow entries on the switch we have decided to divide flows into QoS classes on the entrance switch which will act as Router1 to DiffServ domain where flows will be controlled as per their associated classes. Next, these classes will be mapped to DSCP values for ToS to shape the traffic on the Router2 interface (s2-eth1), as seen in *Figure 3.45*.

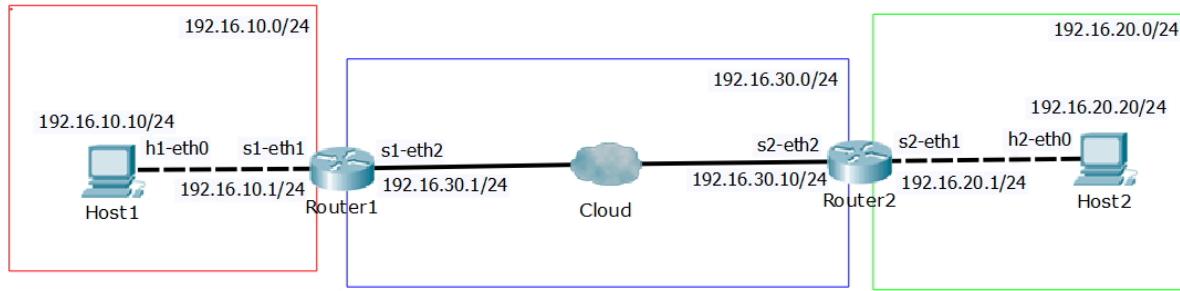


Figure 3.45: Topo for per Class with DSCP QoS.

We are going to use UDP and QoS classes for Router1 with bandwidth limits which will be mapped to DSCP on Router2, as seen in *Figure 3.46*.

Queue	Type	Port	DSCP	Max Rate (bps)	Min Rate (bps)
0	UDP	5000	48	1000000	N/A
1	UDP	21	18	1000000	300000
2	UDP	80	36	1000000	500000

Figure 3.46: Linux HTB Queues and DSCP Mapping for Cloud Scenario.

First, we have started the Topo with OF13 used to connect between the Ryu controller and the switches and then we have set up the OVSDB connection from Switch2 via port 6632. We also had to configure both hosts and add default routes to the GWs on of the network, as seen in *Figure 3.47*.

```

Node: s1" (root)
root@Ryu:"# sudo su
root@Ryu:/home/ryu# ovs-vsctl set bridge s1 protocols=OpenFlow13
root@Ryu:/home/ryu# 

Node: s2" (root)
root@Ryu:"# sudo su
root@Ryu:/home/ryu# ovs-vsctl set bridge s2 protocols=OpenFlow13
root@Ryu:/home/ryu# ovs-vsctl set-manager ptcp:6632
root@Ryu:/home/ryu# 

Node: h1"
root@Ryu:"# ip addr del 10.0.0.1/8 dev h1-eth0
root@Ryu:"# ip addr add 192.16.10.10/24 dev h1-eth0
root@Ryu:"# ip route add default via 192.16.10.1
root@Ryu:"# route -n
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref Use Iface
0.0.0.0         192.16.10.1   0.0.0.0       UG    0      0    0 h1-eth0
192.16.10.0     0.0.0.0       255.255.255.0 U     0      0    0 h1-eth0
root@Ryu: # 

Node: h2"
root@Ryu:"# ip addr del 10.0.0.2/8 dev h2-eth0
root@Ryu:"# ip addr add 192.16.20.20/24 dev h2-eth0
root@Ryu:"# ip route add default via 192.16.20.1
root@Ryu:"# route -n
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref Use Iface
0.0.0.0         192.16.20.1   0.0.0.0       UG    0      0    0 h2-eth0
192.16.20.0     0.0.0.0       255.255.255.0 U     0      0    0 h2-eth0
root@Ryu: # 

ryu@Ryu: ~
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s2) (s2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth2
s2 lo: s2-eth1:h2-eth0 s2-eth2:s1-eth2
c0
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s2-eth1 (OK OK)
s2-eth2<->s1-eth2 (OK OK)

```

Figure 3.47: Configuration of Bridges, Routes and IP Addresses of Hosts.

When we run Ryu with apps it will report that both switches have joined the QoS, so now we can use REST to configure the connection to OVSDB. Before we set up the QoS queue on Switch2 we were able to see four entities on each switch, as seen in *Figure 3.48*.

Figure 3.48: Switches Joining QoS and HTB Queue Creation.

From Figure 3.49 we can see that there is an HTB with a maximum rate limit set to 1 Mbps with three queues of 1 Mbps for maximum limit as well as 500 Kbps and 300 Kbps for minimum transfer rate.

```
root@Ryu:/home/ryu# ovs-vsctl list qos
_uuid : e53790f8-0be6-4213-89b9-a91a622b9aee
_external_ids : {}
_other_config : {max-rate="1000000"}
queues : {0=e2665566-150b-4e7c-9fa1-db6c2a9a2dde, 1=83032aad-897d-46af-955c-656147b6e694, 2=2e
2be9a8-16cb-4c48-899e-919b110706e6}
type : linux-htb
root@Ryu:/home/ryu# ovs-vsctl list queue
_uuid : 2e2be9a8-16cb-4c48-899e-919b110706e6
_dscp : []
_external_ids : {}
_other_config : {min-rate="500000"}
_uuid : 83032aad-897d-46af-955c-656147b6e694
_dscp : []
_external_ids : {}
_other_config : {min-rate="300000"}
_uuid : e2665566-150b-4e7c-9fa1-db6c2a9a2dde
_dscp : []
_external_ids : {}
_other_config : {max-rate="1000000"}
```

Figure 3.49: HTB with Settings of QoS Transfer Limits.

Now we need to send the IP address settings and default routes via NBI to the app running on Ryu with HTTP *post* method. This creates new entries' in the Flow Table 1 which will be used to transfer the forwarding logic to the controller, as seen in *Figure 3.50*.

```

root@Ryu:/home/ryu# curl -g -X GET http://127.0.0.1:8080/router/0000000000000000
[{"internal_network": [{"route": [{"route_id": 1, "destination": "0.0.0.0/0", "gateway": "192.16.30.10"}], "address": [{"address_id": 1, "address": "192.16.10.1/24"}, {"address_id": 2, "address": "192.16.30.1/24"}]}, {"switch_id": "0000000000000001"}]root@Ryu:/home/ryu# curl -g -X GET http://127.0.0.1:8080/router/0000000000000002
[{"internal_network": [{"route": [{"route_id": 1, "destination": "0.0.0.0/0", "gateway": "192.16.30.1"}], "address": [{"address_id": 1, "address": "192.16.20.1/24"}, {"address_id": 2, "address": "192.16.30.10/24"}]}, {"switch_id": "0000000000000002"}]root@Ryu:/home/ryu# ovs-ofctl -O OpenFlow13 dump-flows s1
OFPT_FLOW reply (OF1.3) (xid=0x2):
  cookie=0x0, duration=1608.369s, table=0, n_packets=3, n_bytes=180, priority=0 actions=goto table:1
  cookie=0x1, duration=76.761s, table=1, n_packets=0, n_bytes=0, priority=1037, ip,nw_dst=192.16.10.1 actions
=CONTROLLER:65535
  cookie=0x2, duration=70.257s, table=1, n_packets=0, n_bytes=0, priority=1037, ip,nw_dst=192.16.30.1 actions
=CONTROLLER:65535
  cookie=0x1, duration=76.761s, table=1, n_packets=0, n_bytes=0, priority=2, ip,nw_dst=192.16.10.0/24 actions
=CONTROLLER:65535
  cookie=0x2, duration=70.258s, table=1, n_packets=0, n_bytes=0, priority=2, ip,nw_dst=192.16.30.0/24 actions
=CONTROLLER:65535
  cookie=0x1, duration=76.761s, table=1, n_packets=0, n_bytes=0, priority=36, ip,nw_src=192.16.10.0/24, nw_dst
=192.16.10.0/24 actions=NORMAL
  cookie=0x2, duration=70.257s, table=1, n_packets=0, n_bytes=0, priority=36, ip,nw_src=192.16.30.0/24, nw_dst
=192.16.30.0/24 actions=NORMAL
  cookie=0x10000, duration=52.392s, table=1, n_packets=0, n_bytes=0, priority=1, ip actions=dec_ttl, set_field
:da:60:7e:1c:6b:8b->eth_src, set_field:8e:bd:8a:d4:78:48->eth_dst, output:2
  cookie=0x0, duration=1608.369s, table=1, n_packets=3, n_bytes=180, priority=1, arp actions=CONTROLLER:65535
  cookie=0x0, duration=1608.369s, table=1, n_packets=0, n_bytes=0, priority=0 actions=NORMAL
root@Ryu:/home/ryu# ovs-ofctl -O OpenFlow13 dump-flows s2
OFPT_FLOW reply (OF1.3) (xid=0x2):
  cookie=0x0, duration=1613.024s, table=0, n_packets=4, n_bytes=240, priority=0 actions=goto table:1
  cookie=0x2, duration=57.05s, table=1, n_packets=0, n_bytes=0, priority=1037, ip,nw_dst=192.16.30.10 actions
=CONTROLLER:65535
  cookie=0x1, duration=63.059s, table=1, n_packets=0, n_bytes=0, priority=1037, ip,nw_dst=192.16.20.1 actions
=CONTROLLER:65535
  cookie=0x1, duration=63.059s, table=1, n_packets=0, n_bytes=0, priority=2, ip,nw_dst=192.16.20.0/24 actions
=CONTROLLER:65535
  cookie=0x2, duration=57.051s, table=1, n_packets=0, n_bytes=0, priority=2, ip,nw_dst=192.16.30.0/24 actions
=CONTROLLER:65535
  cookie=0x1, duration=63.059s, table=1, n_packets=0, n_bytes=0, priority=36, ip,nw_src=192.16.20.0/24, nw_dst
=192.16.20.0/24 actions=NORMAL
  cookie=0x2, duration=57.05s, table=1, n_packets=0, n_bytes=0, priority=36, ip,nw_src=192.16.30.0/24, nw_dst
=192.16.30.0/24 actions=NORMAL
  cookie=0x10000, duration=49.735s, table=1, n_packets=0, n_bytes=0, priority=1, ip actions=dec_ttl, set_field
:8e:bd:8a:d4:78:48->eth_src, set_field:da:60:7e:1c:6b:8b->eth_dst, output:2
  cookie=0x0, duration=1613.025s, table=1, n_packets=4, n_bytes=240, priority=1, arp actions=CONTROLLER:65535
  cookie=0x0, duration=1613.025s, table=1, n_packets=0, n_bytes=0, priority=0 actions=NORMAL
root@Ryu:/home/ryu#

```

Figure 3.50: Flow Table Entries with IP Addresses and Default Routes.

Created classes on Switch1 are mapped to three *qos_ids* with UDP and the destination IP address of 192.16.10.10 (Host1) for ports: 21, 80 and 5000, as seen in *Figure 3.51*.

```

root@Ryu:/home/ryu# curl -g -X POST -d '{"match": {"nw_dst": "192.16.10.10", "nw_proto": "UDP", "tp_dst": "21", "actions": [{"mark": "18"}]}' http://127.0.0.1:8080/qos/rules/0000000000000001 [{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "QoS added. : qos_id=1"}]}]
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "QoS added. : qos_id=1"}]}]curl: (6) Could not resolve host: [switch_id
curl: (6) Could not resolve host: 0000000000000001
curl: (6) Could not resolve host: command result
curl: (6) Could not resolve host: [result
curl: (6) Could not resolve host: success
curl: (6) Could not resolve host: details
curl: (6) Could not resolve host: qos added.
root@Ryu:/home/ryu# curl -g -X POST -d '{"match": {"nw_dst": "192.16.10.10", "nw_proto": "UDP", "tp_dst": "80", "actions": [{"mark": "36"}]}' http://127.0.0.1:8080/qos/rules/0000000000000001 [{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "QoS added. : qos_id=2"}]}]
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "QoS added. : qos_id=2"}]}]curl: (6) Could not resolve host: [switch_id
curl: (6) Could not resolve host: 0000000000000001
curl: (6) Could not resolve host: command result
curl: (6) Could not resolve host: [result
curl: (6) Could not resolve host: success
curl: (6) Could not resolve host: details
curl: (6) Could not resolve host: qos added.
root@Ryu:/home/ryu# curl -g -X POST -d '{"match": {"nw_dst": "192.16.10.10", "nw_proto": "UDP", "tp_dst": "5000", "actions": [{"mark": "48"}]}' http://127.0.0.1:8080/qos/rules/0000000000000001 [{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "QoS added. : qos_id=3"}]}]
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "QoS added. : qos_id=3"}]}]curl: (6) Could not resolve host: [switch_id
curl: (6) Could not resolve host: 0000000000000001
curl: (6) Could not resolve host: command result
curl: (6) Could not resolve host: [result
curl: (6) Could not resolve host: success
curl: (6) Could not resolve host: details
curl: (6) Could not resolve host: qos added.
root@Ryu:/home/ryu# curl -g -X POST -d '{"qos_id": "0000000000000001", "command_result": [{"qos": "IPv4", "priority": 1, "dl_type": "IPV4", "nw_proto": "UDP", "tp_dst": 21, "qos_id": 1, "nw_dst": "192.16.10.10", "actions": [{"mark": "18"}]}, {"qos": "IPv4", "priority": 1, "dl_type": "IPV4", "nw_proto": "UDP", "tp_dst": 80, "qos_id": 2, "nw_dst": "192.16.10.10", "actions": [{"mark": "36"}]}, {"qos": "IPv4", "priority": 1, "dl_type": "IPV4", "nw_proto": "UDP", "tp_dst": 5000, "qos_id": 3, "nw_dst": "192.16.10.10", "actions": [{"mark": "48"}]}]}]root@Ryu:/home/ryu#

```

Figure 3.51: QoS Classes with DSCP Settings on Switch1.

These classes are matched on s2-eth1 to their DSCP values to support queuing for ToS on Router2, as seen in *Figure 3.52*.

Figure 3.52: QoS Classes and their DSCP Matches on Switch2.

In the next test case, we are going to explore meters and DSCP mappings which control the incoming rate of traffic before forwarding the packets to the output port. Meters are managed by OF protocol by means of the Meter Table and they are attached to the switch to the flows as a replacement for switch ports. Band component specifies the transfer rate and meter can have one or more bands, but only a single band can be used on a flow while the packet rate is measured. Flow directs packets to meter where the rate is measured, and the appropriate band threshold is applied to limit the bandwidth. Meter is an OF feature, so it's supported by the protocol, but not by OVS. Therefore, we will use a CPqD SW switch called OFSoftSwitch13 and compare the deployment against the most recent Open vSwitch 2.8.1 (Open vSwitch, 2018).

First, we created a script to upgrade OVS as per instructions by Gulam (2015) and then we built a custom Topo, as seen in *Figure 3.53*.

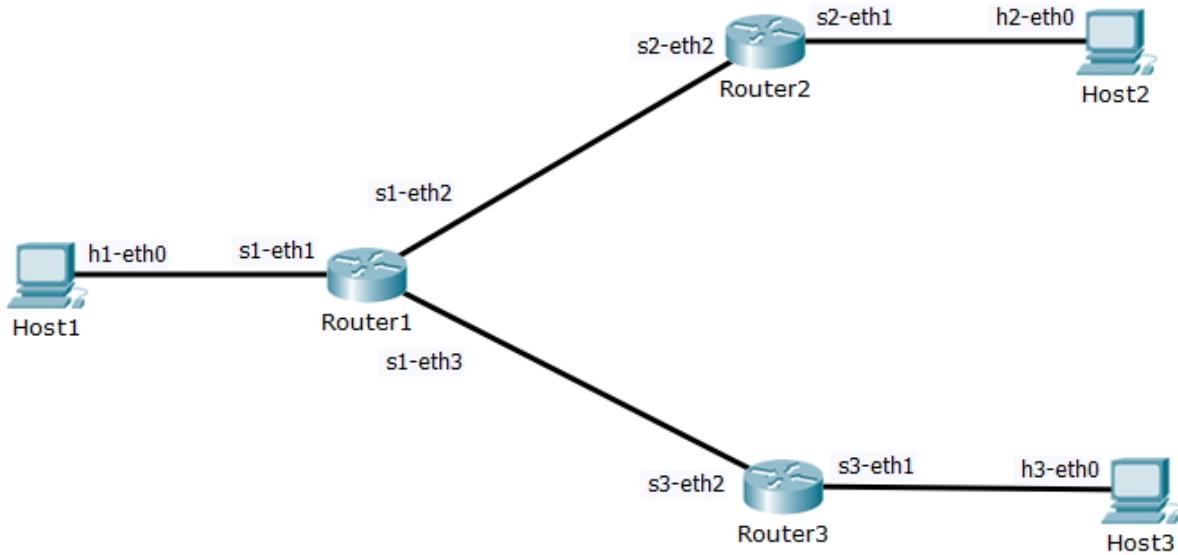


Figure 3.53: Custom QoS Topo without Separation.

This way we will be able to attach all three routers to the HTB queue and QoS rules with modified Ryu apps and centralized OVSDB to use REST for rules programming on the Router1 s1-eth1 interface, as seen in *Figure 3.54*.

```
root@Ryu:/home/ryu
ryu@Ryu:~ ->
ryu@Ryu:~ -> curl -g -X PUT -d '{"tcp:12.0.0.1:6632": "http://localhost:8080/v1.0/ovsdb_ad"}' http://localhost:8080/v1.0/ovsdb_ad
ryu@Ryu:~ -> curl -g -X PUT -d '{"tcp:12.0.0.1:6632": "http://localhost:8080/v1.0/ovsdb_addr"}' http://localhost:8080/v1.0/ovsdb_addr
ryu@Ryu:~ -> curl -g -X PUT -d '{"tcp:12.0.0.1:6632": "http://localhost:8080/v1.0/ovsdb_addr"}' http://localhost:8080/v1.0/ovsdb_addr
ryu@Ryu:~ -> curl -g -X PUT -d '{"tcp:12.0.0.1:6632": "http://localhost:8080/v1.0/ovsdb_addr"}' http://localhost:8080/v1.0/ovsdb_addr
ryu@Ryu:~ -> curl -g -X PUT -d '{"tcp:12.0.0.1:6632": "http://localhost:8080/v1.0/ovsdb_addr"}' http://localhost:8080/v1.0/ovsdb_addr
ryu@Ryu:~ -> curl -g -X POST -d '{"port_name": "si-eth1", "type": "linux-htb", "max_rate": "1000000", "queues": [{"min_rate": "1000000", "min_rate": "3000000"}, {"min_rate": "6000000"}]}' http://localhost:8080/qos/queue/0000000000000001
[{"switch_id": "0000000000000001", "command_result": {"result": "success", "details": [{"port": "config": {"min-rate": "1000000"}, "id": 1, "config": {"min-rate": "3000000"}, "id": 2, "config": {"min-rate": "6000000"}]}]}
ryu@Ryu:~ -> curl -g -X POST -d '{"port_name": "si-eth1", "type": "linux-htb", "max_rate": "1000000", "queues": [{"min_rate": "1000000", "min_rate": "3000000"}, {"min_rate": "6000000"}]}' http://localhost:8080/qos/queue/0000000000000001
[{"switch_id": "0000000000000001", "command_result": {"result": "success", "details": [{"port": "config": {"min-rate": "1000000"}, "id": 1, "config": {"min-rate": "3000000"}, "id": 2, "config": {"min-rate": "6000000"}]}]}
ryu@Ryu:~ -> curl -g -X POST -d '{"match": {"ip": {"queue": "1")'}} http://localhost:8080/qos/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": {"result": "success", "details": [{"qoS added : qos id=1"]]}]
ryu@Ryu:~ -> curl -g -X POST -d '{"match": {"ip": {"queue": "2")'}} http://localhost:8080/qos/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": {"result": "success", "details": [{"qoS added : qos id=2"]]}]
ryu@Ryu:~ -> curl -g -X POST -d '{"match": {"ip": {"queue": "1")'}} http://localhost:8080/qos/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": {"result": "success", "details": [{"qoS added : qos id=1"]]}]
ryu@Ryu:~ -> curl -g -X POST -d '{"match": {"ip": {"queue": "2")'}} http://localhost:8080/qos/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": {"result": "success", "details": [{"qoS added : qos id=2"]]}]
ryu@Ryu:~ -> curl -g -X POST -d '{"match": {"ip": {"queue": "3")'}} http://localhost:8080/qos/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": {"result": "success", "details": [{"qoS added : qos id=3"]]}]
ryu@Ryu:~ -> curl -g -X POST -d '{"match": {"ip": {"queue": "4")'}} http://localhost:8080/qos/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": {"result": "success", "details": [{"qoS added : qos id=4"]]}]
ryu@Ryu:~ -> curl -g -X POST -d '{"match": {"ip": {"queue": "5")'}} http://localhost:8080/qos/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": {"result": "success", "details": [{"qoS added : qos id=5"]]}]
ryu@Ryu:~ -> curl -g -X POST -d '{"match": {"ip": {"queue": "6")'}} http://localhost:8080/qos/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": {"result": "success", "details": [{"qoS added : qos id=6"]]}]
```

Figure 3.54: HTB Queue and QoS Rules on Switch1.

Since the switches joined the QoS API on the Ryu controller we can inspect our QoS rules and DSCP markings for the s1-eth1 port on Switch1, as seen *Figure 3.55*.

```

root@Ryu:/home/ryu# ryu-manager ryu.app.rest_qos ryu.app.rest_conf_switch ryu.app.qos_simple_switch_13
loading app ryu.app.rest_qos
loading app ryu.app.rest_conf_switch
loading app ryu.app.qos_simple_switch_13
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
instantiating app None of ConfSwitchSet
creating context conf_switch
creating context wsgi
instantiating app ryu.app.rest_conf_switch of ConfSwitchAPI
instantiating app ryu.app.qos_simple_switch_13 of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.app.rest_qos of RestQoSAPI
(7233) wsgi starting up on http://0.0.0.0:8080
[QoS] [INFO] dpid=00000000000000000001: Join qos switch.
[QoS] [INFO] dpid=00000000000000000002: Join qos switch.
[QoS] [INFO] dpid=00000000000000000003: Join qos switch.

```

```

ryu@Ryu:~$ curl -g -X GET http://localhost:8080/qos/rules/00000000000000000001
[{"switch_id": "0000000000000001", "command_result": [{"qos": [{"priority": 1, "dl_type": "IPv4", "actions": [{"queue": "1"}], "in_port": 2, "qos_id": 1}, {"priority": 1, "dl_type": "IPv4", "actions": [{"queue": "3"}], "qos_id": 2, "in_port": 2, "ip_dscp": 18}, {"priority": 1, "dl_type": "IPv4", "actions": [{"queue": "1"}], "qos_id": 3, "in_port": 2, "ip_dscp": 36}, {"priority": 1, "dl_type": "IPv4", "actions": [{"queue": "3"}], "qos_id": 5, "in_port": 3, "ip_dscp": 18}, {"priority": 1, "dl_type": "IPv4", "actions": [{"queue": "2"}], "qos_id": 6, "in_port": 3, "ip_dscp": 36}]}]}ryu@Ryu:~$ 

```

Figure 3.55: QoS Rules on Switch1 after Switches connected to the REST QoS API with OVS.

Queues are mapped to DSCP values on Switch1 (*Figure 3.56*) which in turn will be remarked to remaining switches via Meter 1 and DSCP 18 to provide a guaranteed rate of 400 Kbps for incoming traffic on ports: s1-eth2 and s1-eth3, as seen in *Figure 3.57*.

Queue	DSCP	Max Rate (bps)	Min Rate (bps)
0	0	1000000	100000
1	18	1000000	300000
2	36	1000000	600000

Figure 3.56: Linux HTB Queues and DSCP Mapping for Unsliced QoS Topo.

```

root@Ryu:/home/ryu# ryu-manager ryu.app.rest_qos ryu.app.rest_conf_switch ryu.app.qos_simple_switch_13
(7233) accepted ('127.0.0.1', 46686) 127.0.0.1 - [20/Jan/2018 10:33:16] "POST /qos/rules/0000000000000002 HTTP/1.1" 200 238 0.000614
(7233) accepted ('127.0.0.1', 46690) 127.0.0.1 - [20/Jan/2018 10:33:22] "POST /qos/meter/0000000000000002 HTTP/1.1" 200 242 0.000654
(7233) accepted ('127.0.0.1', 46694) 127.0.0.1 - [20/Jan/2018 10:33:28] "POST /qos/rules/0000000000000003 HTTP/1.1" 200 238 0.000782
(7233) accepted ('127.0.0.1', 46698) 127.0.0.1 - [20/Jan/2018 10:33:56] "POST /qos/meter/0000000000000003 HTTP/1.1" 200 242 0.000529
(7233) accepted ('127.0.0.1', 46702)

[6, "in_port": 3, "ip_dscp": 36}]]])ryu@Ryu:~$ clear
ryu@Ryu:~$ curl -g -X POST -d '{"match": {"ip_dscp": "18"}, "actions": {"meter": "1"} }' http://localhost:8080/qos/rules/0000000000000002
[{"switch_id": "0000000000000002", "command_result": [{"result": "success", "details": {"QoS added. : qos_id=1"} }]}]ryu@Ryu:~$ curl -g -X POST -d '{"meter_id": "1", "rate": "400", "prec_level": "1"}' http://localhost:8080/qos/meter/0000000000000002
[{"switch_id": "0000000000000002", "command_result": [{"result": "success", "details": {"Meter added. : Meter ID=1"} }]}]ryu@Ryu:~$ curl -g -X POST -d '{"match": {"ip_dscp": "18"}, "actions": {"meter": "1"} }' http://localhost:8080/qos/rules/0000000000000003
[{"switch_id": "0000000000000003", "command_result": [{"result": "success", "details": {"QoS added. : qos_id=1"} }]}]ryu@Ryu:~$ curl -g -X POST -d '{"meter_id": "1", "rate": "400", "prec_level": "1"}' http://localhost:8080/qos/meter/0000000000000003
[{"switch_id": "0000000000000003", "command_result": [{"result": "success", "details": {"Meter added. : Meter ID=1"} }]}]ryu@Ryu:~$ 

```

Figure 3.57: QoS Meters and Rules on Switch2 and Switch3 with OVS.

With *ofsoftswitch13* we had to use DPCtl and *queue-mod* to implement three queues: 100 Kbps, 300 Kbps, and 600 Kbps within a sliceable Switch1 to divide networks from each other, as seen in *Figure 3.58*.

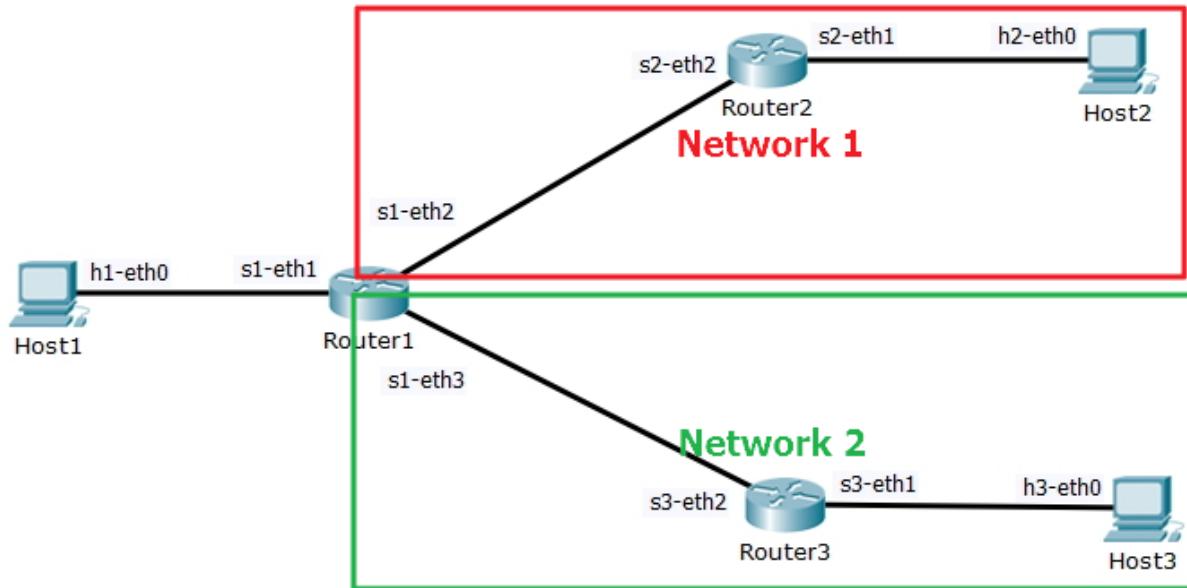


Figure 3.58: Custom QoS Topo with Separation.

After the switches have joined the QoS app via REST API on Ryu we can see that six rules were put on Switch1 ports: s1-eth2 and s1-eth3 with DSCP classes such as BE, AF21 (ToS Hex 0x48) and AF42 (ToS Hex 0x90), as seen in *Figure 3.59*.

Two terminal windows are shown. The left window shows the command `curl -g -X GET http://127.0.0.1:8080/qos/rules/000000000000000000000001` being run, which returns a JSON object containing six QoS rules for ports s1-eth2 and s1-eth3. The right window shows the logs from the Ryu REST API receiving these POST requests and accepting them.

```

root@Ryu:/home/ryu# curl -g -X GET http://127.0.0.1:8080/qos/rules/000000000000000000000001
[{"switch_id": "0000000000000001", "command_result": [{"qos": [{"priority": 1, "dl_type": "IPv4", "actions": [{"queue": "1"}, {"in_port": 2, "qos_id": 1}, {"priority": 1, "dl_type": "IPv4", "actions": [{"queue": "3"}], "qos_id": 2, "in_port": 2, "ip_dscp": 18}, {"priority": 1, "dl_type": "IPv4", "actions": [{"queue": "2"}], "qos_id": 3, "in_port": 2, "ip_dscp": 36}, {"priority": 1, "dl_type": "IPv4", "actions": [{"queue": "1"}], "in_port": 3, "qos_id": 4}, {"priority": 1, "dl_type": "IPv4", "actions": [{"queue": "3"}], "qos_id": 5, "in_port": 3, "ip_dscp": 18}, {"priority": 1, "dl_type": "IPv4", "actions": [{"queue": "2"}], "qos_id": 6, "in_port": 3, "ip_dscp": 36}]}], "dpid=0000000000000001: Join qos switch.", "dpid=0000000000000003: Join qos switch.", "dpid=0000000000000001: Join qos switch.", "(10006) accepted ('127.0.0.1', 42890)", "127.0.0.1 -- [20/Jan/2018 17:22:19] "POST /qos/rules/0000000000000001 HTTP/1.1" 200 238 0.007817, "(10006) accepted ('127.0.0.1', 42892)", "127.0.0.1 -- [20/Jan/2018 17:22:29] "POST /qos/rules/0000000000000001 HTTP/1.1" 200 238 0.000682, "(10006) accepted ('127.0.0.1', 42894)", "127.0.0.1 -- [20/Jan/2018 17:22:35] "POST /qos/rules/0000000000000001 HTTP/1.1" 200 238 0.000719, "(10006) accepted ('127.0.0.1', 42896)", "127.0.0.1 -- [20/Jan/2018 17:22:42] "POST /qos/rules/0000000000000001 HTTP/1.1" 200 238 0.001050, "(10006) accepted ('127.0.0.1', 42898)", "127.0.0.1 -- [20/Jan/2018 17:22:55] "POST /qos/rules/0000000000000001 HTTP/1.1" 200 238 0.000653, "(10006) accepted ('127.0.0.1', 42900)", "127.0.0.1 -- [20/Jan/2018 17:23:04] "POST /qos/rules/0000000000000001 HTTP/1.1" 200 238 0.001011, "(10006) accepted ('127.0.0.1', 42902)", "127.0.0.1 -- [20/Jan/2018 17:24:18] "GET /qos/rules/0000000000000001 HTTP/1.1" 200 802 0.009248

```

Figure 3.59: QoS Rules on Switch1 after Switches connected to the REST QoS API with OFSoftSwitch13.

Next, we have set up meters and match actions to DSCP 18 on Switch2 and Switch3 with band limit of 400 Kbps, as seen in *Figure 3.60*.

```

root@Ryu:/home/ryu# ovs-vsctl show
2629b56c-3746-4e85-94d9-4c5e830b8f42
  Manager "ptcp:6632"
  ovs_version "2.6.1"

root@Ryu:/home/ryu# curl -g -X POST -d '{"match": {"ip dscp": "16"}, "actions": [{"meter": "1"}]}' http://127.0.0.1:8080/qos/rules/0000000000000002
  ("switch_id": "0000000000000002", "command_result": [{"result": "success", "details": ["QoS added. : qos_id=1"]}], "meter_id": "1", "rate": "400", "prec_level": "1"})
root@Ryu:/home/ryu# curl -g -X POST -d '{"meter_id": "1", "flags": "KBPS", "bands": [{"type": "DSCP_MARK", "rate": "400", "prec_level": "1"}]}' http://127.0.0.1:8080/qos/meter/0000000000000002
  ("switch_id": "0000000000000002", "command_result": [{"result": "success", "details": ["Meter added. : Meter ID=1"]}], "meter_id": "1", "rate": "400", "prec_level": "1"))
root@Ryu:/home/ryu# curl -g -X POST -d '{"match": {"ip dscp": "16"}, "actions": [{"meter": "1"}]}' http://127.0.0.1:8080/qos/rules/0000000000000003
  ("switch_id": "0000000000000003", "command_result": [{"result": "success", "details": ["QoS added. : qos_id=1"]}], "meter_id": "1", "rate": "400", "prec_level": "1"))
root@Ryu:/home/ryu# curl -g -X POST -d '{"meter_id": "1", "flags": "KBPS", "bands": [{"type": "DSCP_MARK", "rate": "400", "prec_level": "1"}]}' http://127.0.0.1:8080/qos/meter/0000000000000003
  ("switch_id": "0000000000000003", "command_result": [{"result": "success", "details": ["Meter added. : Meter ID=1"]}], "meter_id": "1", "rate": "400", "prec_level": "1"))

(10006) accepted ('127.0.0.1', 42904)
127.0.0.1 - - [20/Jan/2018 17:36:36] "POST /qos/rules/0000000000000002 HTTP/1.1"
200 238 0.000641
(10006) accepted ('127.0.0.1', 42906)
127.0.0.1 - - [20/Jan/2018 17:36:44] "POST /qos/meter/0000000000000002 HTTP/1.1"
200 242 0.000608
(10006) accepted ('127.0.0.1', 42908)
127.0.0.1 - - [20/Jan/2018 17:36:52] "POST /qos/rules/0000000000000003 HTTP/1.1"
200 238 0.000602
(10006) accepted ('127.0.0.1', 42910)
127.0.0.1 - - [20/Jan/2018 17:36:58] "POST /qos/meter/0000000000000003 HTTP/1.1"
200 242 0.000448

```

Figure 3.60: QoS Meters and Rules on Switch2 and Switch3 with OFSoftSwitch13.

We can also see that there are not OVS bridges present as at this time as we do not use OVS, but OFSoftSwitch13. However, when we check the meters and rules on Switch2 and Switch3 we noticed that they are correctly mapped to DSCP 18 with *meter_id=1*, as seen in *Figure 3.61*.

```

root@Ryu:/home/ryu# curl -g -X GET http://127.0.0.1:8080/qos/rules/0000000000000002
  ("switch_id": "0000000000000002", "command_result": [{"qos": [{"priority": 1, "dl_type": "IPv4", "ip_dscp": 18, "actions": [{"meter": "1", "qos_id": "1"}]}]}], "rule_id": "0038")
root@Ryu:/home/ryu# curl -g -X GET http://127.0.0.1:8080/qos/rules/0000000000000003
  ("switch_id": "0000000000000003", "command_result": [{"qos": [{"priority": 1, "dl_type": "IPv4", "ip_dscp": 18, "actions": [{"meter": "1", "qos_id": "1"}]}]}], "rule_id": "0039")
root@Ryu:/home/ryu# curl -g -X GET http://127.0.0.1:8080/qos/meter/0000000000000002
  ("switch_id": "0000000000000002", "command_result": [{"band_stats": [{"byte_in_count": 0, "flow_count": 0, "packet_in_count": 0, "duration_nsec": 592000, "duration_sec": 629, "meter_id": "1"}]}], "meter_id": "1", "rate": "400", "prec_level": "1"))
root@Ryu:/home/ryu# curl -g -X GET http://127.0.0.1:8080/qos/meter/0000000000000003
  ("switch_id": "0000000000000003", "command_result": [{"band_stats": [{"byte_in_count": 0, "flow_count": 0, "packet_in_count": 0, "duration_nsec": 369000, "duration_sec": 622, "meter_id": "1"}]}], "meter_id": "1", "rate": "400", "prec_level": "1"))

(10006) accepted ('127.0.0.1', 42912)
127.0.0.1 - - [20/Jan/2018 17:46:50] "GET /qos/rules/0000000000000002 HTTP/1.1"
200 283 0.002905
(10006) accepted ('127.0.0.1', 42914)
127.0.0.1 - - [20/Jan/2018 17:47:00] "GET /qos/rules/0000000000000003 HTTP/1.1"
200 283 0.002714
(10006) accepted ('127.0.0.1', 42916)
127.0.0.1 - - [20/Jan/2018 17:47:14] "GET /qos/meter/0000000000000002 HTTP/1.1"
200 383 0.001212
(10006) accepted ('127.0.0.1', 42918)
127.0.0.1 - - [20/Jan/2018 17:47:20] "GET /qos/meter/0000000000000003 HTTP/1.1"
200 383 0.001216

```

Figure 3.61: Retrieval of QoS Meters and Rules on Switch2 and Switch3 with OFSoftSwitch13.

3.4.2. Results

3.4.2.1. MPLS

We have executed a series of test cases to prove that implemented policies with Cisco MPLS are working correctly.

During ICMP requests from VM1 to VM2, we can observe that on output interfaces, packets are marked as *class-default*, as seen in *Figure 3.62*. This means that these aren't associated with EXP bits, but only provide best effort delivery with a DSCP label of *CS0*, as seen in *Figure 3.63*.

Figure 3.62: Default Policy Class and Captured Packets.

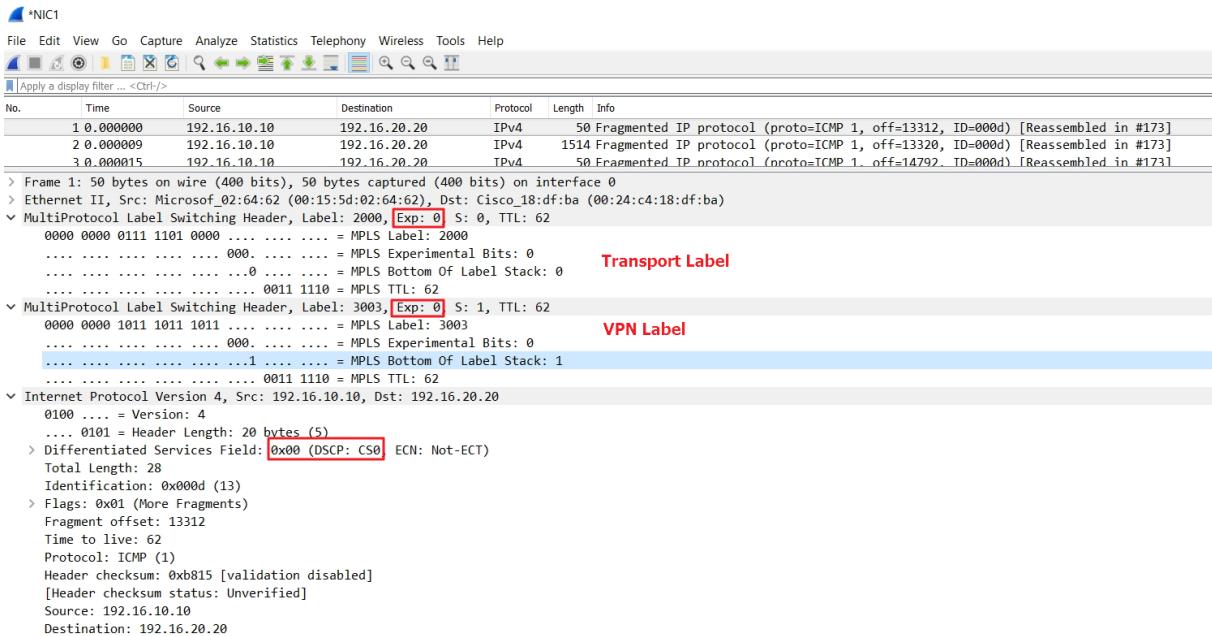


Figure 3.63: ICMP Wireshark Capture.

When we were downloading 1 MB and 10 MB sample files from the FTP server (VM2) we were able to see that the appropriate packets are captured across their policies except for the Dublin and the CSR1000V2 nodes where mappings use *class-default*, as seen in *Figure 3.64*. This is because that data remains in the MPLS domain where EXP bits are used for QoS, as seen in *Figure 3.65*. We can also see that the maximum transfer oscillates around 120 – 129 kilobytes per second (kB/s) which is equal to approx. 1 Mbps as specified for the FTP class and that FTP data uses EXP 5 in the MPLS domain which is mapped to DSCP EF.

Figure 3.64: FTP Policy Class and Captured Packets.

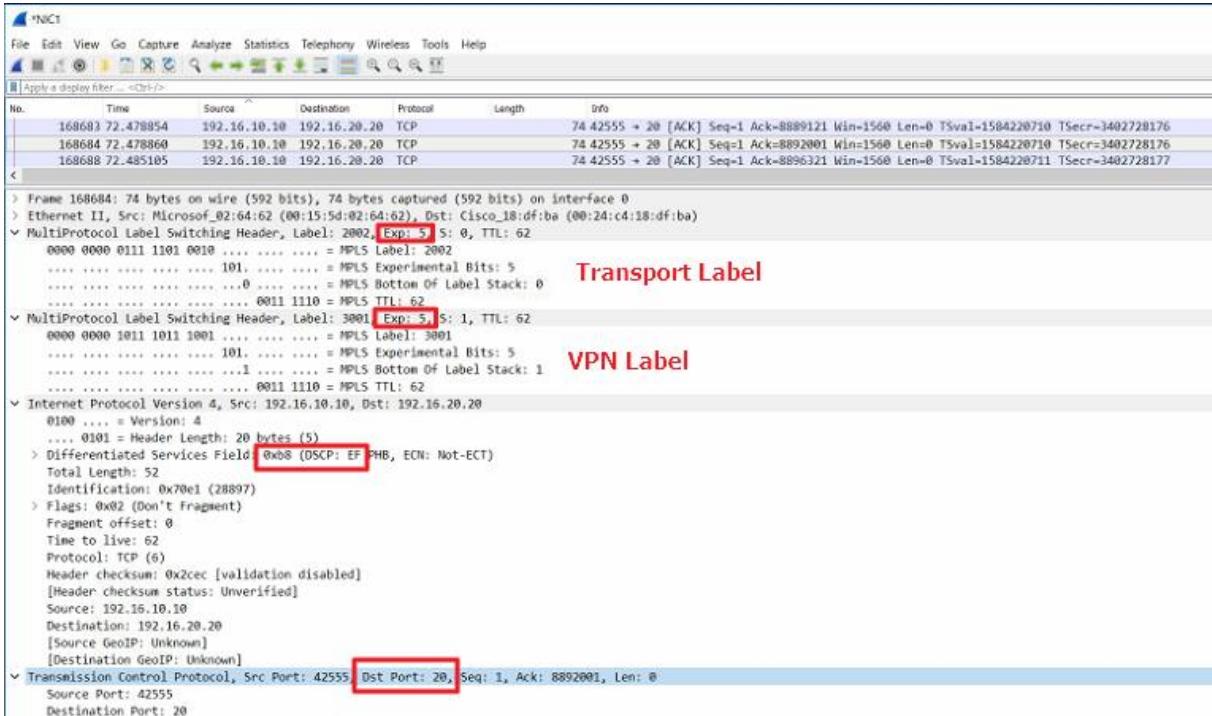


Figure 3.65: FTP Wireshark Capture.

However, when using `wget` to download samples of 1 MB, 10 MB, 100 MB, 1000 MB while simultaneously saturating the link with ICMP requests, the average transfer rate was around 5.54 MB/s which utilized the link at above 90 %. Like FTP packets, HTTP requests were captured in accordance with their class, as seen in *Figure 3.66*.

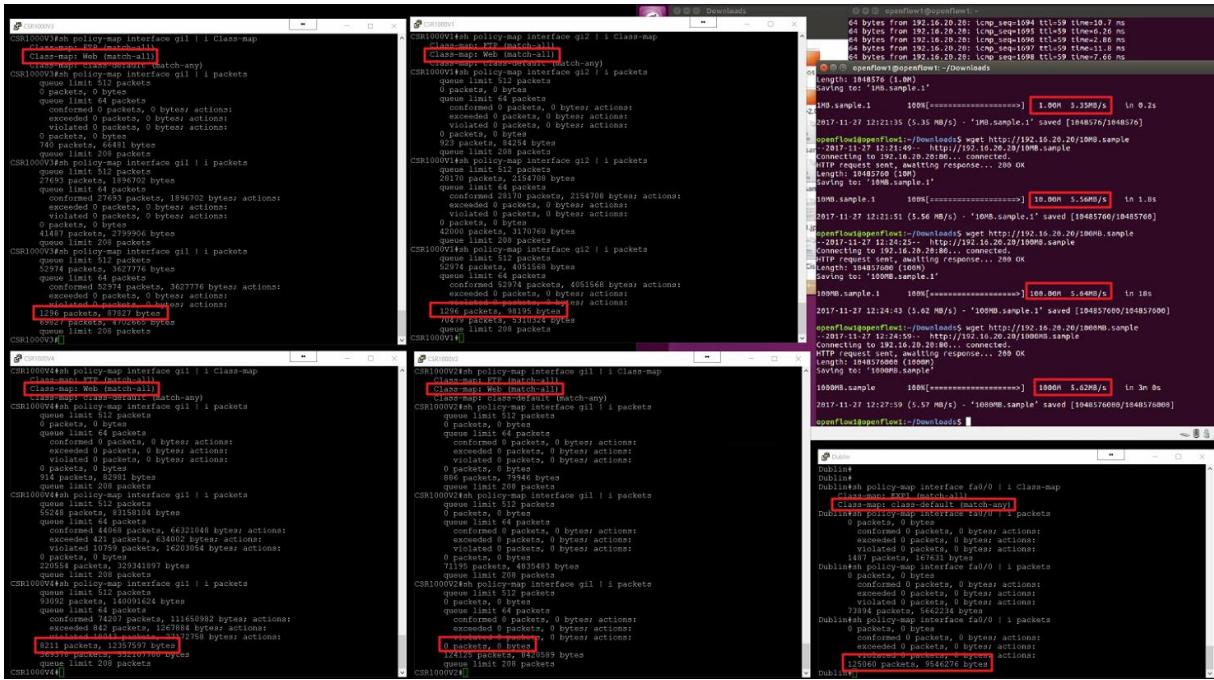


Figure 3.66: Web Policy Class and Captured Packets.

From Wireshark capture (*Figure 3.67*) we can also see that MPLS EXP 4 label mapping to DSCP AF41 is correct while making a request to the Web server from VM1.

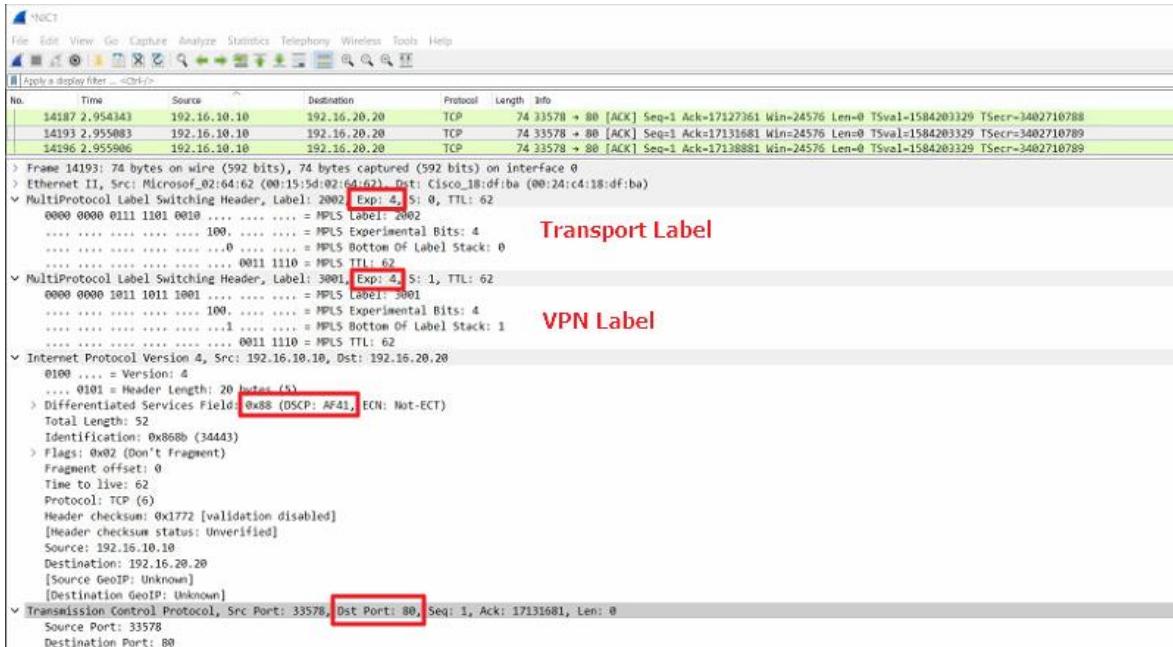


Figure 3.67: HTTP Wireshark Capture.

After enabling NBAR (Cisco, 2017) on output interfaces we were able to see the same results for the nodes within the Cisco MPLS topology as discussed previously. Packets for HTTP and FTP traffic were transferred from CE side via VPN to PE side before they reached the MPLS domain, where these are not marked by protocol discovery until they left it on the other side via (Gi1) on CSR1000V2 through VPN to CSR100V4 which is directly connected to

VM2. When we executed simultaneous downloads via FTP and HTTP transfer rates were matching our QoS policies taking into consideration that congestion was small enough.

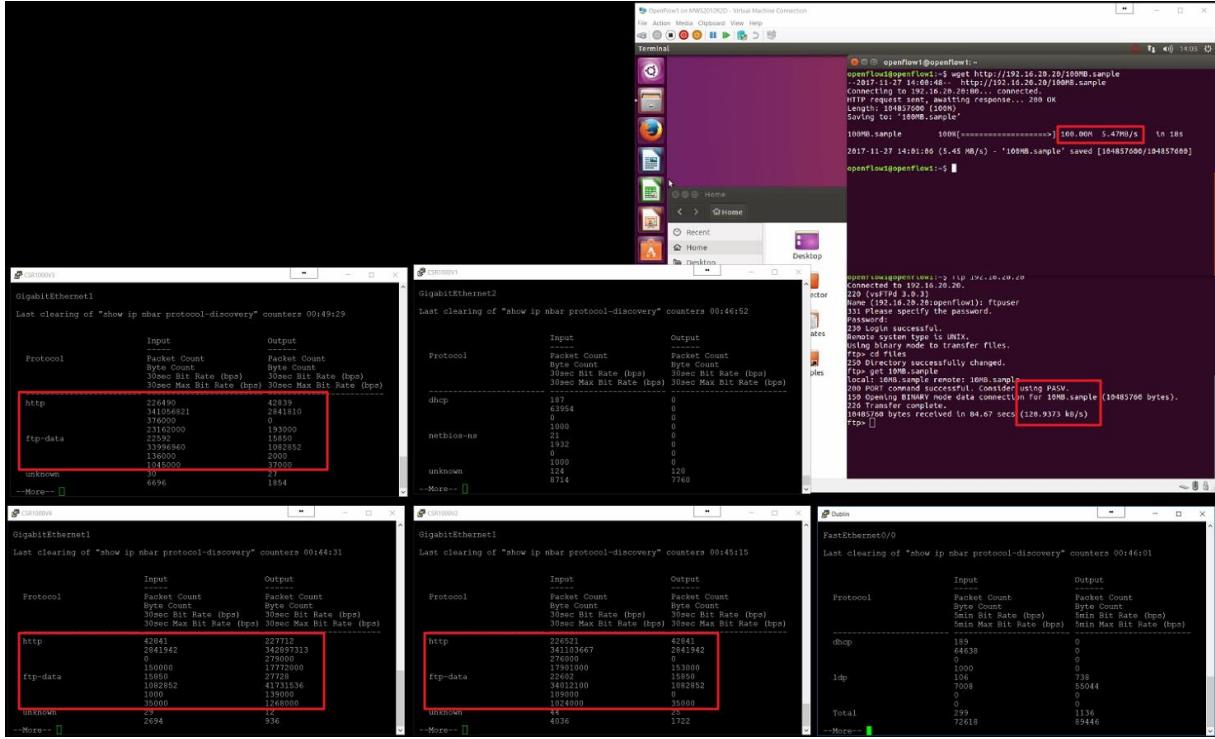


Figure 3.68: NBAR Protocol Discovery with 10 MB Sample for FTP and 100 MB Sample for HTTP.

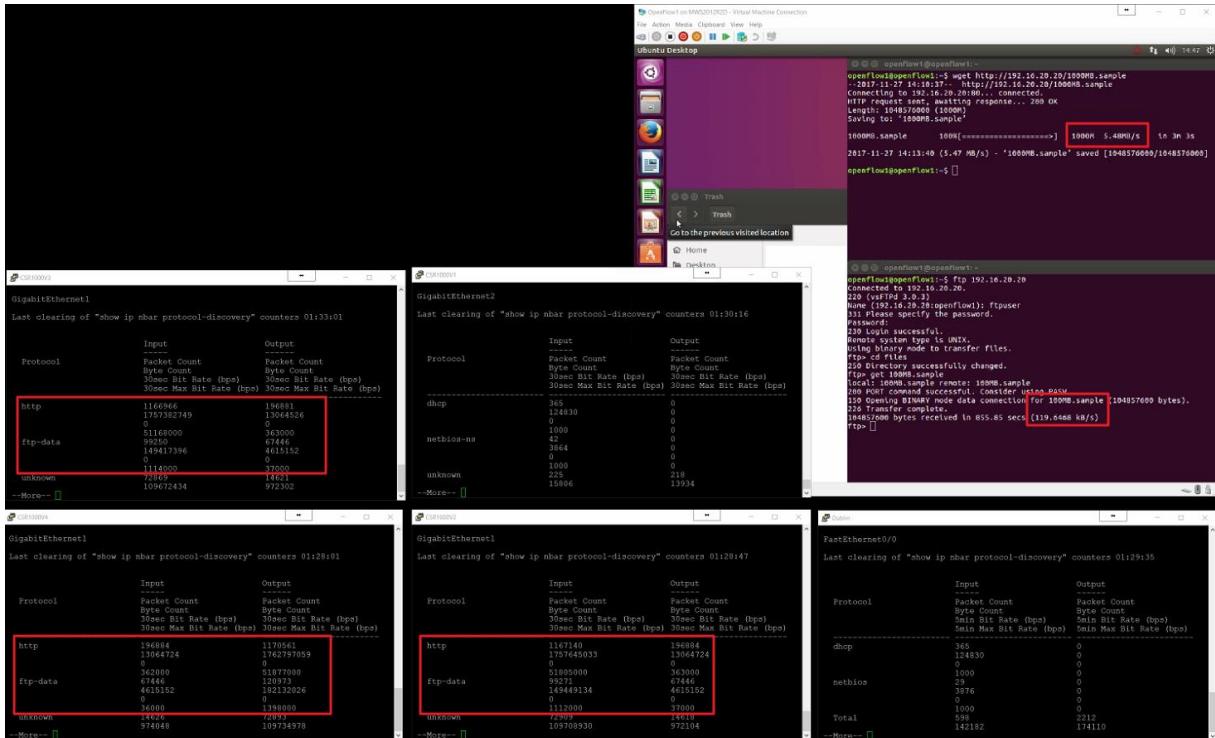


Figure 3.69: NBAR Protocol Discovery with 100 MB Sample for FTP and 1000 MB Sample for HTTP.

To investigate the policies further we have decided to saturate the link with hping3 by sending ICMP requests of size 52000 B with 10 packets per second to VM2 via *sudo hping3 192.16.20.20 -1 -d 52000 -i u10000* (Sanfilippo, 2014). The purpose of this test wasn't the overutilization of the interface, but to see the reaction of implemented data classifications in relation to bandwidth. This has shown that FTP remained in and around 1 Mbps where HTTP never dropped below 1 Mbps while downloading large samples, as seen in *Figure 3.70*. It proves that created policies reacted accordingly to data types to provide QoS.

```

openflow1@openflow1:~$ ftp 192.16.20.20
Connected to 192.16.20.20.
220 (vsFTPd 3.0.3)
Name (192.16.20.20:openflow1): ftpuser
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd files
250 Directory successfully changed.
ftp> get 100MB.sample
local: 100MB.sample remote: 100MB.sample
200 PORT command successful. Consider using PASV.
V.
150 Opening BINARY mode data connection for 100
MB.sample (104857600 bytes).
226 Transfer complete.
104857600 bytes received in 866.43 secs (118.18
56 kB/s)
ftp> 

```

```

openflow1@openflow1:~/Downloads$ wget http://192.16.20.20/1000MB.sample
Connecting to 192.16.20.20:80... ^
openflow1@openflow1:~/Downloads$ wget http://192.16.20.20/1000MB.sample
--2017-11-27 17:55:11-- http://192.16.20.20/1000MB.sample
Connecting to 192.16.20.20:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1048576000 (1000M)
Saving to: '1000MB.sample.2'

1000MB.sample.2      5%[>          ] 56.50M  5.38MB/s   eta
5m 30s ^C
openflow1@openflow1:~/Downloads$ wget http://192.16.20.20/1000MB.sample
--2017-11-27 17:56:10-- http://192.16.20.20:80/1000MB.sample
Connecting to 192.16.20.20:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1048576000 (1000M)
Saving to: '1000MB.sample.3'

1000MB.sample.3    13%[=>          ] 133.96M  1.77MB/s   eta
1000MB.sample.3  100%[=====] 1000M  1.78MB/s  in 9m 9s
2017-11-27 18:05:19 (1.82 MB/s) - '1000MB.sample.3' saved [1048576000/1
048576000]

openflow1@openflow1:~/Downloads$ 

```

```

openflow1@openflow1:~$ 

```

Figure 3.70: Link Congestion with 100 MB Sample for FTP and 1000 MB Sample for HTTP.

In the first MPLS on Linux test case we have executed hping3 with a rate of 10 packets per second like previous examples simultaneously with the FTP transfer of all four sample files one by one. Congestion on the tunnel endpoint resulted in an average transfer of 2.12 MB/s, as seen in *Figure 3.71*.

```

OpenFlow1 on MWS2012R2D - Virtual Machine Connection
File Action Media Clipboard View Help
Terminal 10:40
openflow1@openflow1: ~/Downloads
len=1492 ip=192.16.20.20 ttl=59 DF id=49150 icmp_seq=15394 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49154 icmp_seq=15395 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49157 icmp_seq=15396 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49167 icmp_seq=15402 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49172 icmp_seq=15404 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49173 icmp_seq=15405 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49177 icmp_seq=15406 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49182 icmp_seq=15408 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49190 icmp_seq=15414 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49192 icmp_seq=15416 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49195 icmp_seq=15417 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49196 icmp_seq=15418 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49199 icmp_seq=15420 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49204 icmp_seq=15421 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49205 icmp_seq=15422 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49213 icmp_seq=15430 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49215 icmp_seq=15431 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49220 icmp_seq=15432 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49222 icmp_seq=15433 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49224 icmp_seq=15434 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49228 icmp_seq=15441 rtt=0.0 ms
len=1492 ip=192.16.20.20 ttl=59 DF id=49230 icmp_seq=15442 rtt=0.0 ms
openflow1@openflow1: ~/Downloads
local: 1MB.sample remote: 1MB.sample
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for 1MB.sample (1048576 bytes).
226 Transfer complete.
1048576 bytes received in 0.74 secs (1.3442 MB/s)
ftp> get 10MB.sample
local: 10MB.sample remote: 10MB.sample
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for 10MB.sample (10485760 bytes).
226 Transfer complete.
10485760 bytes received in 4.48 secs (2.2331 MB/s)
ftp> get 100MB.sample
local: 100MB.sample remote: 100MB.sample
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for 100MB.sample (104857600 bytes).
226 Transfer complete.
104857600 bytes received in 41.46 secs (2.4122 MB/s)
ftp> get 1000MB.sample
local: 1000MB.sample remote: 1000MB.sample
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for 1000MB.sample (1048576000 bytes).
226 Transfer complete.
1048576000 bytes received in 399.42 secs (2.5036 MB/s)
ftp>

```

Figure 3.71: FTP Transfer via TE Tunnel Bandwidth Test.

From NBAR enabled on all MPLS domain interfaces on the PE we can see that there are only a few incoming packets marked as FTP while properly accounted packets for FTP and ICMP on P node are marked on both output interfaces used in the tunnel, as seen in *Figure 3.72*.

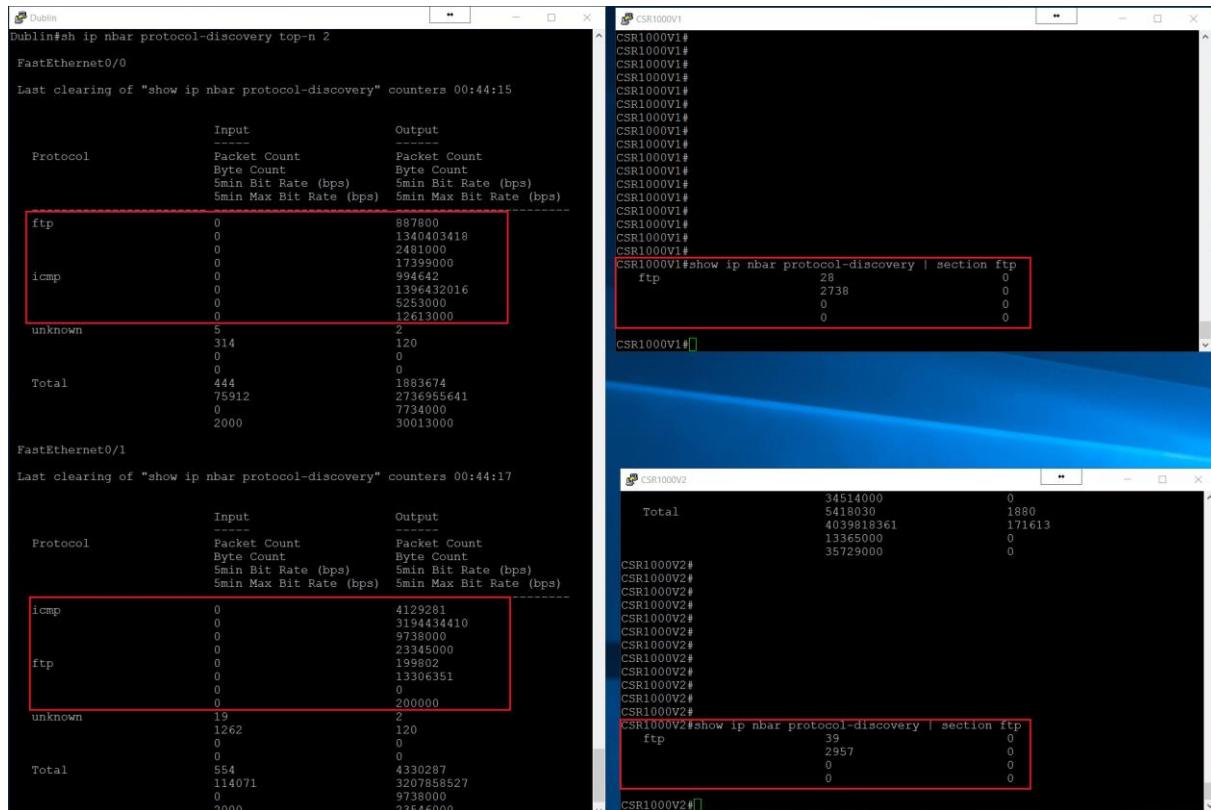


Figure 3.72: NBAR for FTP and ICMP of PE MPLS Domain.

During similar tests for HTTP, the average download speed was 2.2 MB/s while accounting for the protocol NBAR matched the amount of transferred data via the path to the tunnel endpoint, as seen in *Figure 3.73*.

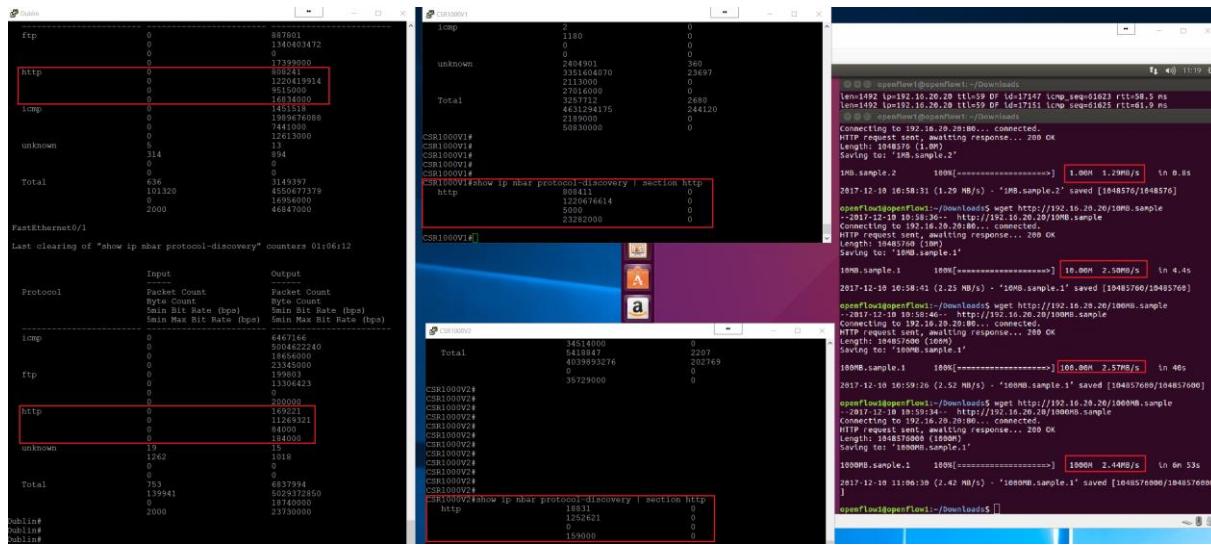


Figure 3.73: NBAR for HTTP of PE MPLS Domain and HTTP Transfer via TE Tunnel Bandwidth Test.

Now we have executed HTTP and FTP downloads at the same time as well as tried to saturate the link with hping3 to see whether the throughput falls below 1024 Kbps.

From *Figure 3.74* we can see that the mean values for both protocols were split nearly even with a 5% difference between each other with 1.68 MB/s for web traffic and 1.29 MB/s for data traffic.

```

openflow1@openflow1:~/Downloads$ Saving to: '1MB.sample'
1MB.sample 100% 1.00M 1.77MB/s in 0.6s
2017-12-10 11:24:13 (1.77 MB/s) - '1MB.sample' saved [1048576/1048576]

openflow1@openflow1:~/Downloads$ wget http://192.16.20.20/10MB.sample
20/10MB.sample
--2017-12-10 11:24:35-- http://192.16.20.20/10MB.sample
Connecting to 192.16.20.20:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10485760 (10M)
Saving to: '10MB.sample.1'

10MB.sample.1 100% 10.00M 1.41MB/s in 7.8s
2017-12-10 11:24:44 (1.28 MB/s) - '10MB.sample.1' saved [10485760/10485760]

openflow1@openflow1:~/Downloads$ wget http://192.16.20.20/100MB.sample
20/100MB.sample
--2017-12-10 11:24:59-- http://192.16.20.20/100MB.sample
Connecting to 192.16.20.20:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 104857600 (100M)
Saving to: '100MB.sample'

100MB.sample 100% 100.00M 2.03MB/s in 82s
2017-12-10 11:26:22 (1.22 MB/s) - '100MB.sample' saved [104857600/104857600]

openflow1@openflow1:~/Downloads$ wget http://192.16.20.20/1000MB.sample
20/1000MB.sample
--2017-12-10 11:27:07-- http://192.16.20.20/1000MB.sample
Connecting to 192.16.20.20:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1048576000 (1000M)
Saving to: '1000MB.sample'

1000MB.sample 100% 1000M 1.50MB/s in 13m 17s
2017-12-10 11:40:24 (1.26 MB/s) - '1000MB.sample' saved

openflow1@openflow1:~/Downloads$ ftp 192.16.20.20
Connected to 192.16.20.20.
220 (vsFTPd 3.0.3)
Name (192.16.20.20:openflow1): ftpuser
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd files
250 Directory successfully changed.
ftp> get 1MB.sample
local: 1MB.sample remote: 1MB.sample
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for 1MB.sample (1048576 bytes).
226 Transfer complete.
1048576 bytes received in 0.71 secs (1.4026 MB/s)
ftp> get 10MB.sample
local: 10MB.sample remote: 10MB.sample
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for 10MB.sample (1048560 bytes).
226 Transfer complete.
10485760 bytes received in 8.54 secs (1.1716 MB/s)
ftp> get 100MB.sample
local: 100MB.sample remote: 100MB.sample
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for 100MB.sample (104857600 bytes).
226 Transfer complete.
104857600 bytes received in 77.30 secs (1.2937 MB/s)
ftp> get 1000MB.sample
local: 1000MB.sample remote: 1000MB.sample
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for 1000MB.sample (1048576000 bytes).
226 Transfer complete.
1048576000 bytes received in 814.05 secs (1.2284 MB/s)
ftp>

```

Figure 3.74: Web and Data Traffic via TE Tunnel Bandwidth Test.

To test the maximum link the bandwidth was set to 6144 Kbps we have executed three continuous get requests with both protocols based on the largest sample file and then we have calculated the mean value which was 2.45 MB/s which is equal to 19600 Kbps, as seen in *Figure 3.75*.

```

openflow1@openflow1:~/Downloads$ wget http://192.16.20.20/1000MB.sample
--2017-12-10 11:59:54-- http://192.16.20.20/1000MB.sample
Connecting to 192.16.20.20:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1048576000 (1000M)
Saving to: '1000MB.sample'

1000MB.sample      64%[=====] 648.76M 2.4
8M1000MB.sample  71%[==> ] 714.74M 2.51MB/s eta 1m 5
1000MB.sample  71%[==> ] 715.22M 2.49MB/s eta 1m 55s
1000MB.sample 100% [1000M 2.32MB/s] in 6m 42s
2017-12-10 12:06:36 (2.49 MB/s) - '1000MB.sample' saved [1048576000/1048576000]

openflow1@openflow1:~/Downloads$ wget http://192.16.20.20/1000MB.sample
--2017-12-10 12:08:59-- http://192.16.20.20/1000MB.sample
Connecting to 192.16.20.20:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1048576000 (1000M)
Saving to: '1000MB.sample.1'

1000MB.sample. 100% [1000M 2.31MB/s] in 6m 46s
2017-12-10 12:15:46 (2.46 MB/s) - '1000MB.sample.1' saved [1048576000/1048576000]

openflow1@openflow1:~/Downloads$ wget http://192.16.20.20/1000MB.sample
--2017-12-10 12:15:55-- http://192.16.20.20/1000MB.sample
Connecting to 192.16.20.20:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1048576000 (1000M)
Saving to: '1000MB.sample.2'

1000MB.sample. 100% [1000M 2.61MB/s] in 6m 40s
2017-12-10 12:22:36 (2.50 MB/s) - '1000MB.sample.2' saved [1048576000/1048576000]

openflow1@openflow1:~/Downloads$ 
```

Figure 3.75: Maximum Link Bandwidth Test for Web and Data Traffic.

During FTP authentication to the server on the VM2 VPN label, 2003 was used to pass the username before starting the ftp-data transfer which is the local label used to transfer the packets between P node and CSR1000V1, as seen in *Figure 3.76*.

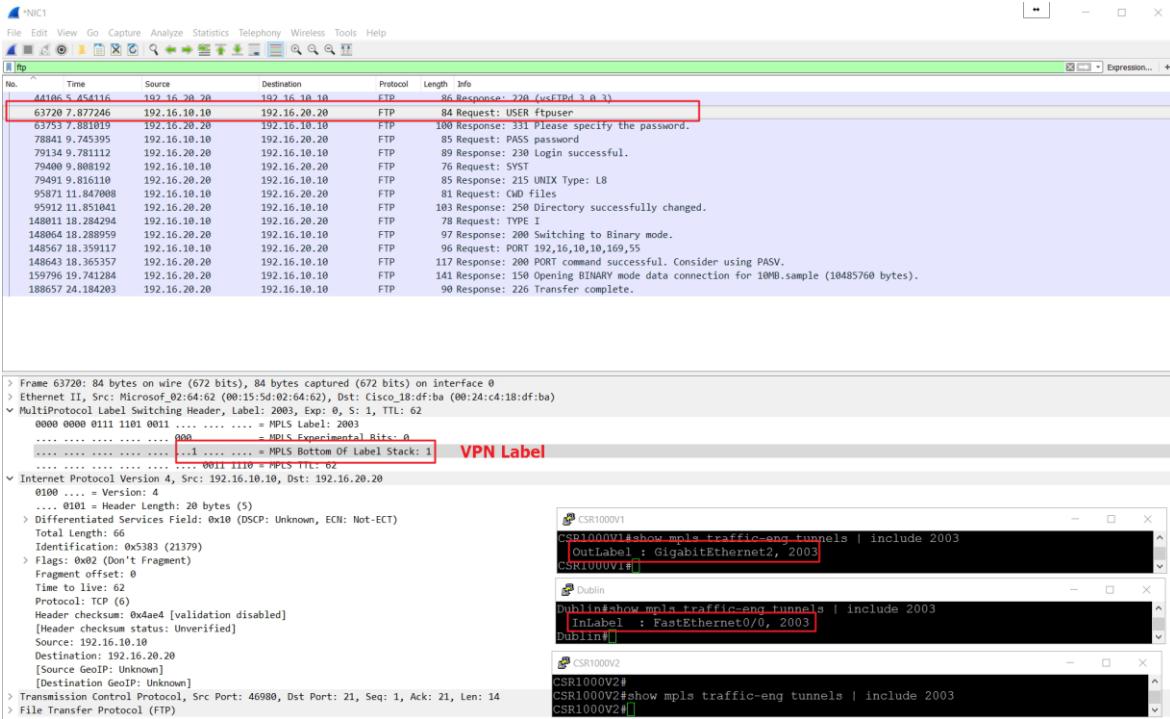


Figure 3.76: Wireshark Capture of VPN Label for FTP on NIC1.

We also have captured the reply packet to `wget` request from VM1 to VM2 which was labeled as 2002 which is used as a local label between CSR1000V2 and P node, as seen in *Figure 3.77*.

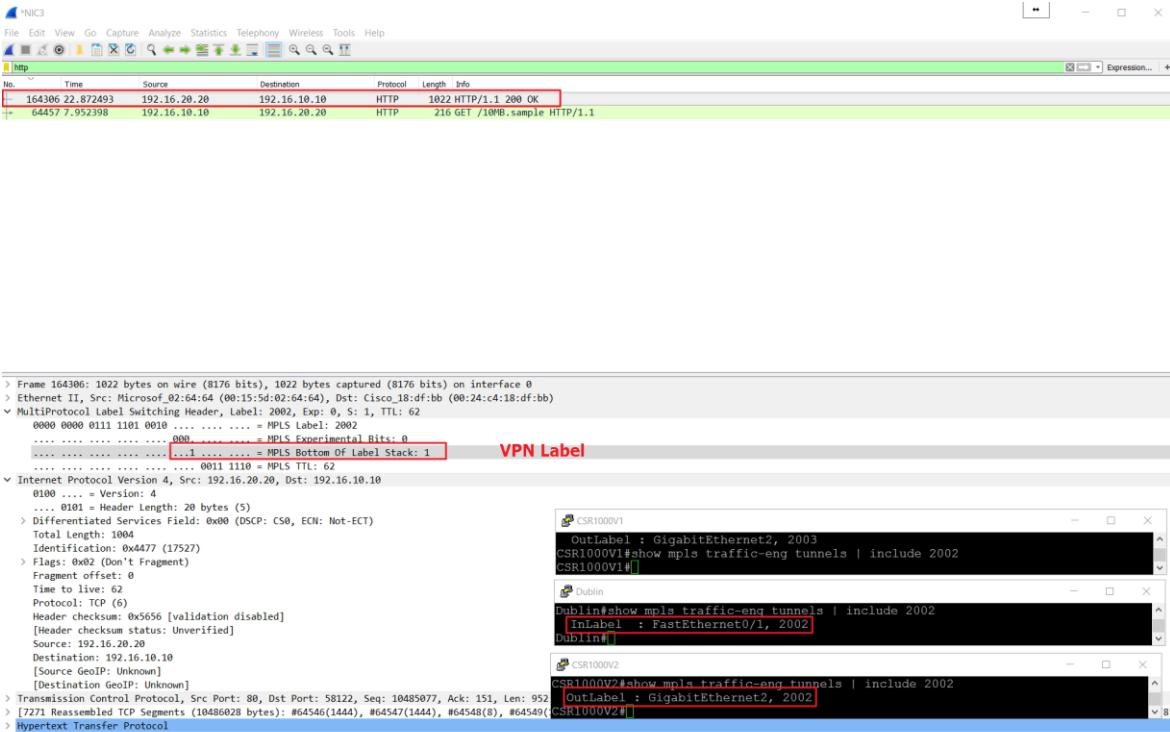


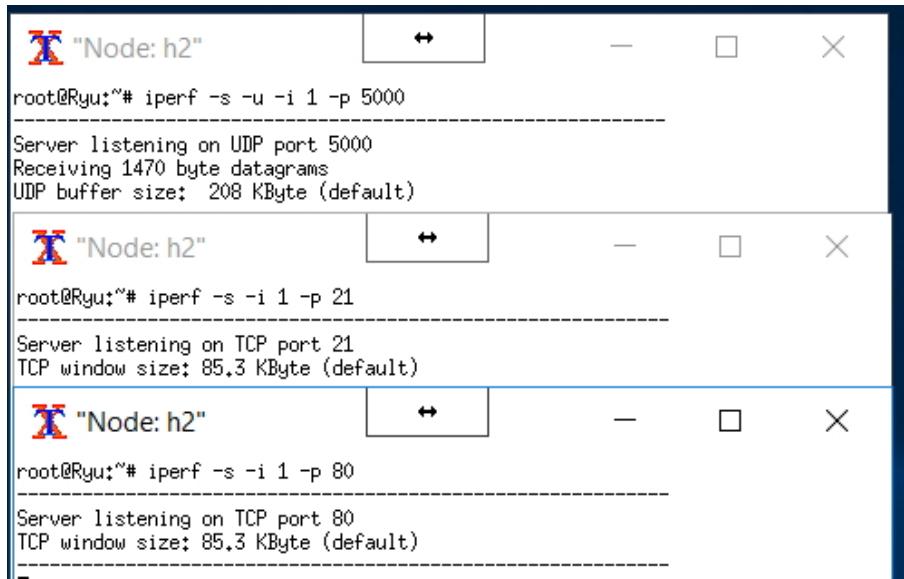
Figure 3.77: Wireshark Capture of VPN Label for HTTP on NIC3.

From the tests of TE tunnel, we have seen that RSVP bandwidth parameter for TE

doesn't work the same way as the bandwidth limit on the interface. However, bandwidth set on the tunnel resulted in expected values which were not less than 1024 Kbps. We also observed that the bottom of the label stack was used for local MPLS domain traffic for the explicit path to the tunnel endpoint rather than for transport labels. To summarize all facts, we can acknowledge that MPLS-TE has no good mechanism to limit bandwidth until there are multiple tunnels to a destination with QoS policies in conjunction, to divide the packets into classes. Our examples didn't use ToS as we can see on captured packets in *Figure 3.76* and *Figure 3.77* where they're marked as *0x00* or *0x10* which means that it's routine, not classified traffic for QoS (Digital Hybrid, 2012).

3.4.2.2. OpenFlow

To test the first scenario, we can use *get* to obtain the QoS settings out of the OVSDB or use iPerf which will be running as the Server on Host2 with a specific port number, as seen in *Figure 3.78*.



The figure displays three vertically stacked terminal windows, each titled "Node: h2".

- Top Window:** Shows the command `root@Ryu:~# iperf -s -u -i 1 -p 5000`. The output indicates the server is listening on UDP port 5000, receiving 1470 byte datagrams, and the UDP buffer size is 208 KByte (default).
- Middle Window:** Shows the command `root@Ryu:~# iperf -s -i 1 -p 21`. The output indicates the server is listening on TCP port 21, and the TCP window size is 85.3 KByte (default).
- Bottom Window:** Shows the command `root@Ryu:~# iperf -s -i 1 -p 80`. The output indicates the server is listening on TCP port 80, and the TCP window size is 85.3 KByte (default).

Figure 3.78: Server for iPerf on Ports 5000, 21 and 80.

During the execution of simultaneous requests, we were able to see that bandwidth did go above 10 Mbps on the link to Host2 (s1-eth2) while the maximum transfer rate to port 5000 and FTP server weren't higher than 1 Mbps, as seen in *Figure 3.79*.

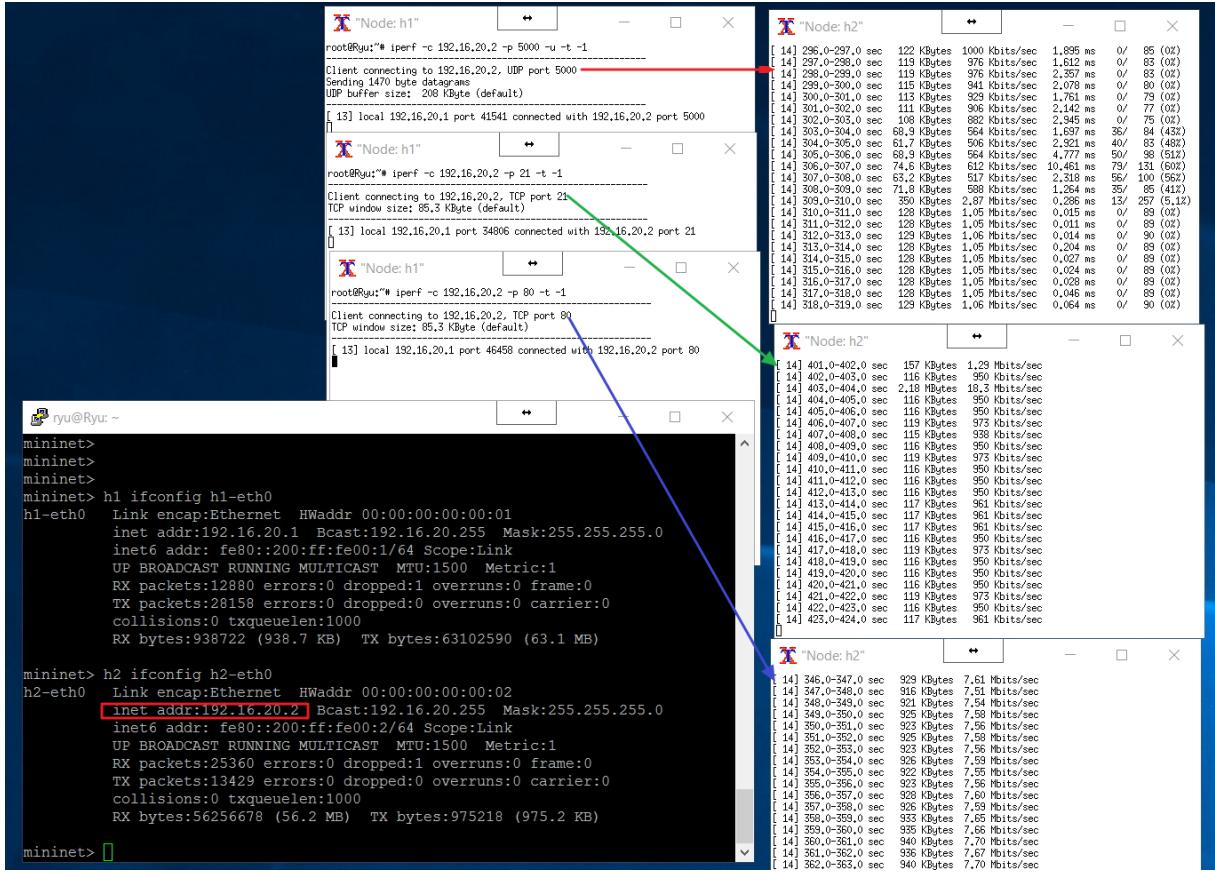


Figure 3.79: Simultaneous Requests to Ports 5000, 21 and 80.

However, when we stopped requests to port 5000 and the FTP server there were still some packets traversing for port 21 across until they were reassembled on the other side and then bandwidth for Web server increased to 10 Mbps, as seen in *Figure 3.80*.

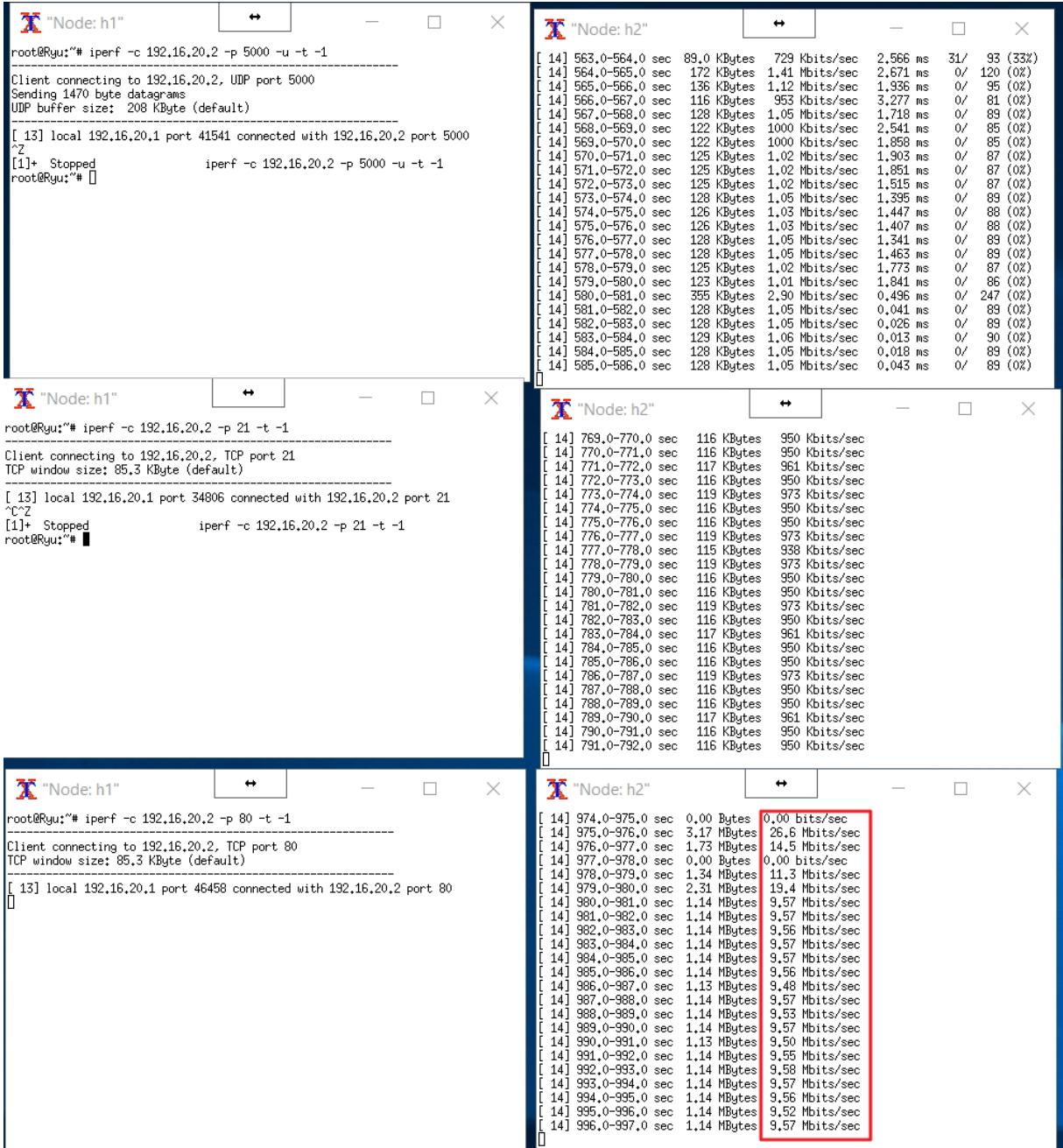


Figure 3.80: Requests to Port 80 for Web Server.

Next, we stopped packets going to port 80 to observe the behaviour and noticed that the transfer rate met our QoS criteria as per their flow entries as bandwidth isn't greater than 1 Mbps and no less than 500 Kbps, as seen in *Figure 3.81*.



Figure 3.81: Requests to Port 5000 for UDP Traffic and to Port 21 for FTP Server.

This experiment has proven that it's possible to use per-flow QoS policies with the use of OVSDB and external controllers such as Ryu to reserve bandwidth to specific apps running on different port numbers.

Irrespectively, like the previous case, to test the DSCP scenario we have started three iPerf servers on Host2 on the same ports, but with the use of `-u` to connect via UDP rather than TCP as we have noticed that the rules set up with this protocol were generating a lot of traffic due to acknowledgments of each packet received which was influencing the results. This only implicated the throughput parameter; the overall bandwidth wasn't impacted as the

sum of the transfer rate was around 1 Mbps. However, it was not allowing us to check the QoS policies due to the large fluctuations in the values.

When we started simultaneous requests, all traffic on port 5000 was taking most of the bandwidth as its set to Queue 0 and DSCP 48 which means it will try to gain all of the bandwidth during the congestion as there's no minimum rate configured for this queue, as seen in *Figure 3.82*.

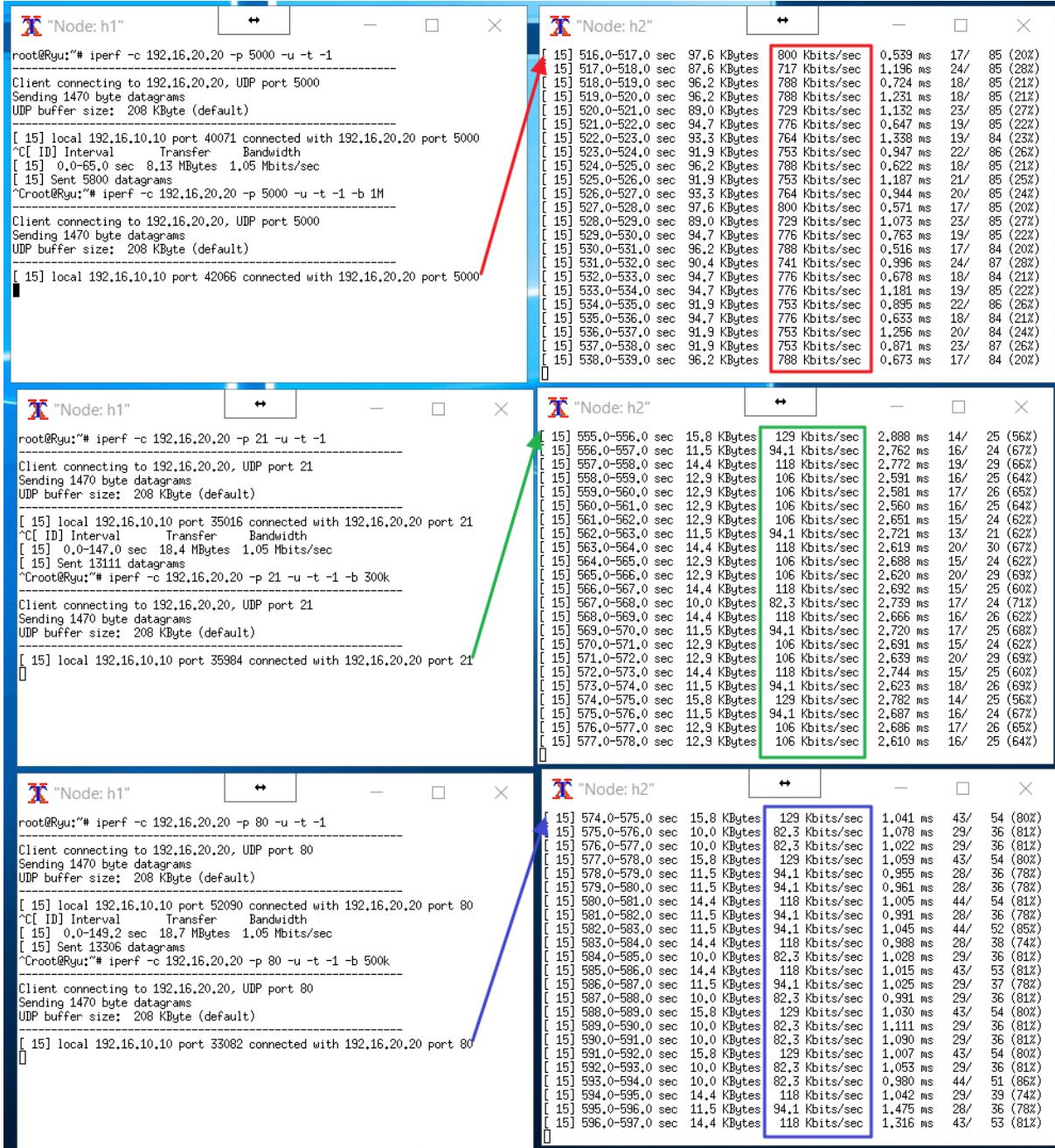


Figure 3.82: Simultaneous Requests to Ports 5000, 21 and 80 with DSCP.

Next, we decided to run only requests to port 21 and 80, but this time the minimum transfer rate wasn't met for Queue 2. Therefore, we have removed the matching action on s2-eth1 for

Queue 0 and created a new HTB with Queue 1 and Queue 2 only, as seen in *Figure 3.83*.

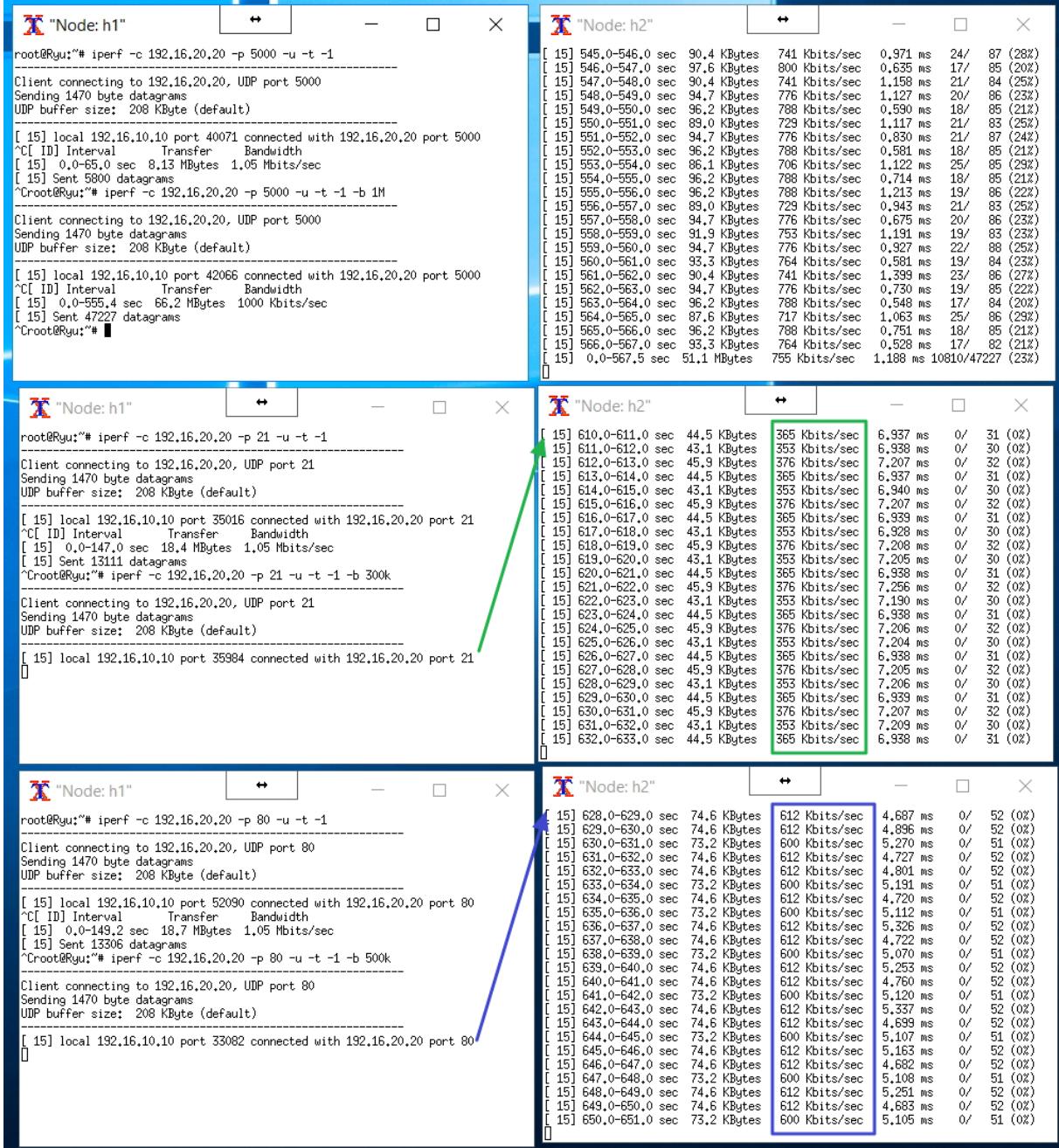


Figure 3.83: Requests to Port 21 and to Port 80 with DSCP.

Now when we execute requests to port 5000 and 80 we can see that criteria for DSCP 36 are met as the minimum rate isn't less than 600 Kbps and when we execute to port 21 instead with DSCP 18 it doesn't fall below 300 Kbps, as seen in *Figure 3.84*.

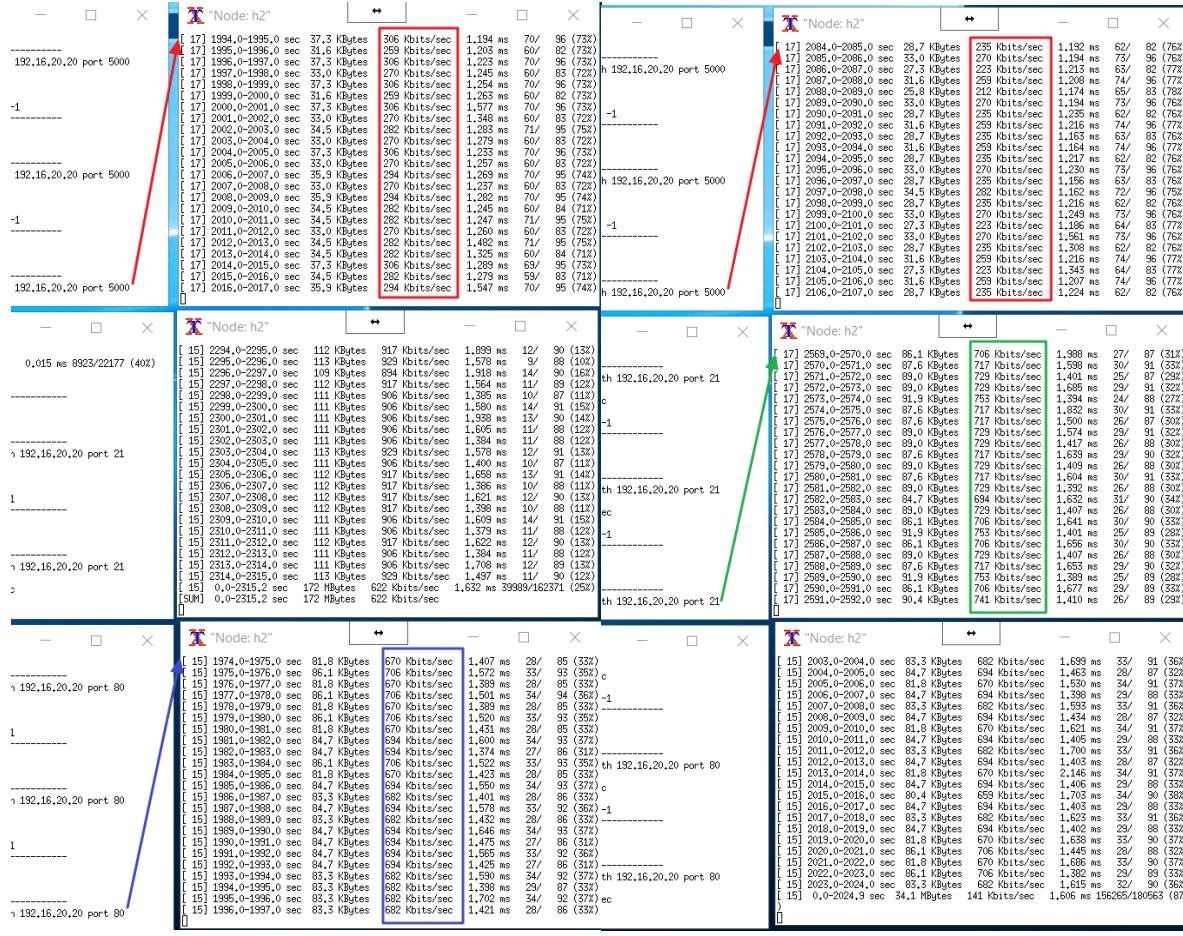


Figure 3.84: Requests to Port 5000 and Port 80 vs Port 5000 and Port 21 with DSCP.

This proves that mapping of QoS classes to DSCP values isn't much different than with MPLS on Cisco, but it requires more testing as first we created queues with buckets and then marked them as ToS on one side to map them on the other side to their DSCP values.

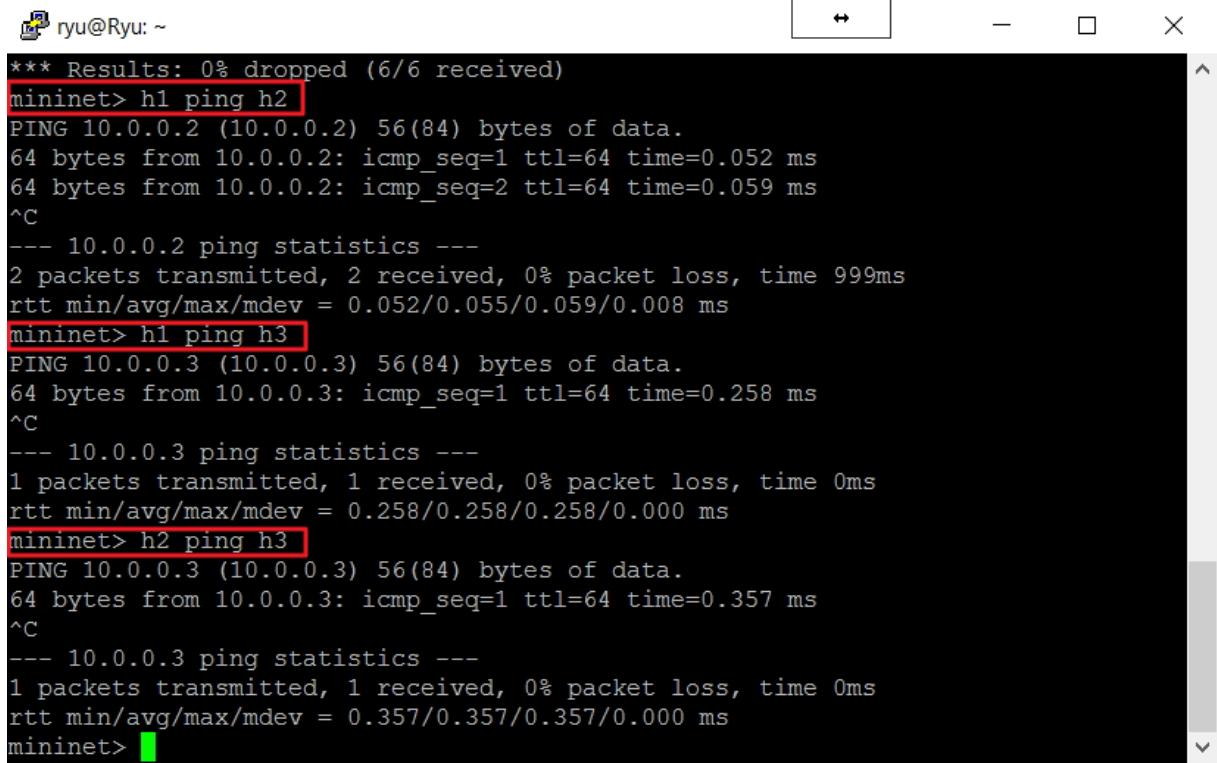
In the test case with the Meter Table, we can see that there are no QoS rules present on Switch1 and Switch2 what indicates that OVS doesn't support this feature, as seen in *Figure 3.85*.

```
127.0.0.1 - [20/Jan/2018 10:34:29] "GET /qos/rules/0000000000000002 HTTP/1.1"
200 180 0.001613
(7233) accepted ('127.0.0.1', 46706)
127.0.0.1 - [20/Jan/2018 10:34:45] "GET /qos/rules/0000000000000003 HTTP/1.1"
200 180 0.001781
ryu@Ryu:~$ curl -g -X GET http://localhost:8080/qos/rules/0000000000000002
[{"switch_id": "0000000000000002", "command_result": []}]ryu@Ryu:~$ curl -g -X GET http://localhost:8080/qos/rules/0000000000000003
[{"switch_id": "0000000000000003", "command_result": []}]ryu@Ryu:~$
```

To check whether meters are supported by this version of OVS we can also try to add a meter manually via `ovs-ofctl -O OpenFlow13 add-meter s2 meter=1, kbps, stats, bands=type=dscp_remark, rate=400, prec_level=1` or to add a flow with `ovs-ofctl -O OpenFlow13 add-flow s3 in_port=1, actions=meter:1, output:3`, but both actions returned an error message as these modification messages are not supported.

However, when we execute pings between the hosts we can see that they can all reach

each other, so we started three sessions of the iPerf server on Host1 with different ports to test the DSCP markings created on s1-eth1, as seen in *Figure 3.86*.



```
ryu@Ryu: ~
*** Results: 0% dropped (6/6 received)
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.052 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.059 ms
^C
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.052/0.055/0.059/0.008 ms
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.258 ms
^C
--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.258/0.258/0.258/0.000 ms
mininet> h2 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.357 ms
^C
--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.357/0.357/0.357/0.000 ms
mininet>
```

Figure 3.86: ICMP Requests between Hosts in Unsliced QoS Topo.

During requests to the Best Effort (BE) class and DSCP 36 from Host2 we have concluded that QoS uses DSCP mapping as traffic to port 5002 and is guaranteed at least 300 Kbps (ToS Hex 0x90) as both requests use bandwidth limitation of 600 Kbps, but only requests to 5002 are prioritized over BE traffic, as seen in *Figure 3.87*.

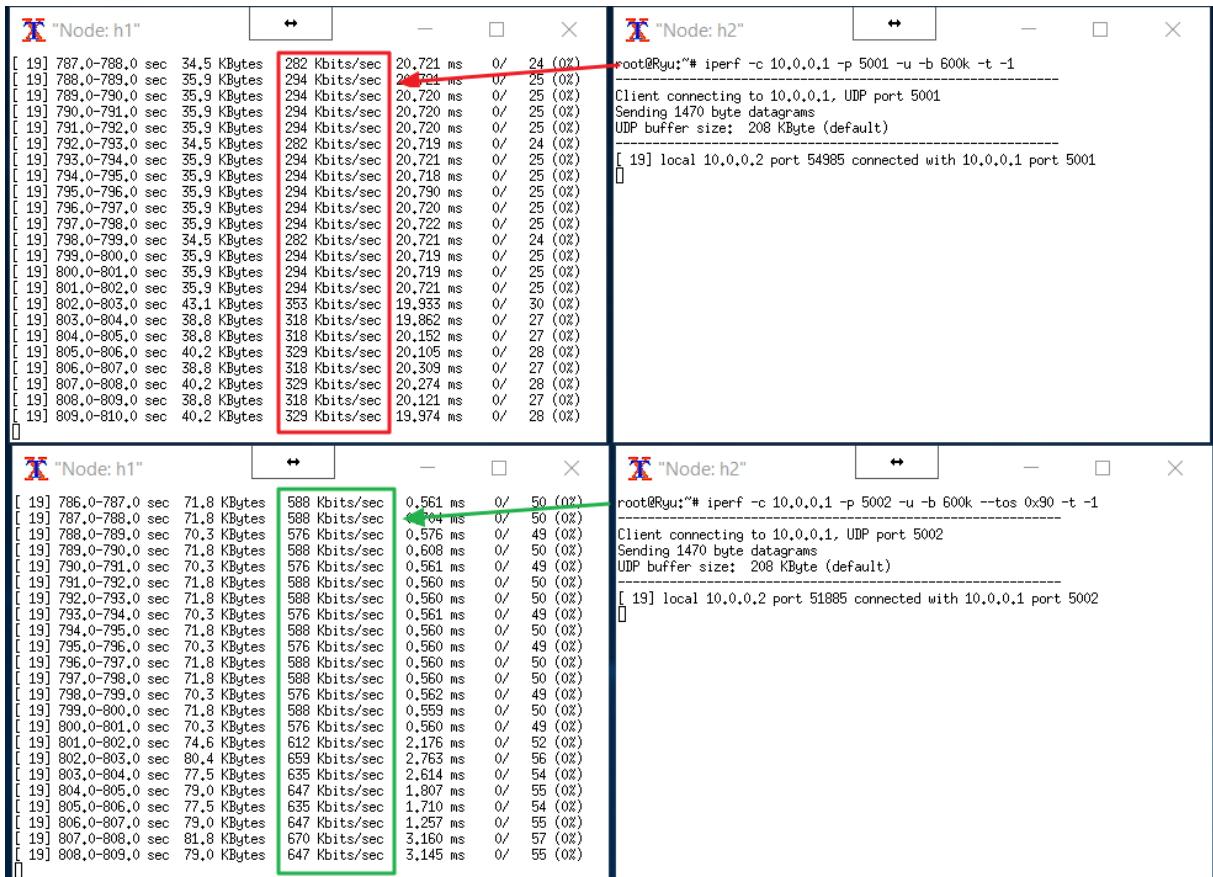


Figure 3.87: Requests to Port 5001 (BE) and Port 5002 (AF42) from Host2.

When making requests to Host1 from Host3 to port 5003 we have noticed that BE traffic doesn't go beyond the 300 Kbps mark, while requests from Host2 on port 5002 (AF42) has guaranteed at least 300 Kbps out of overall 1 Mbps limit, as seen in *Figure 3.88*.

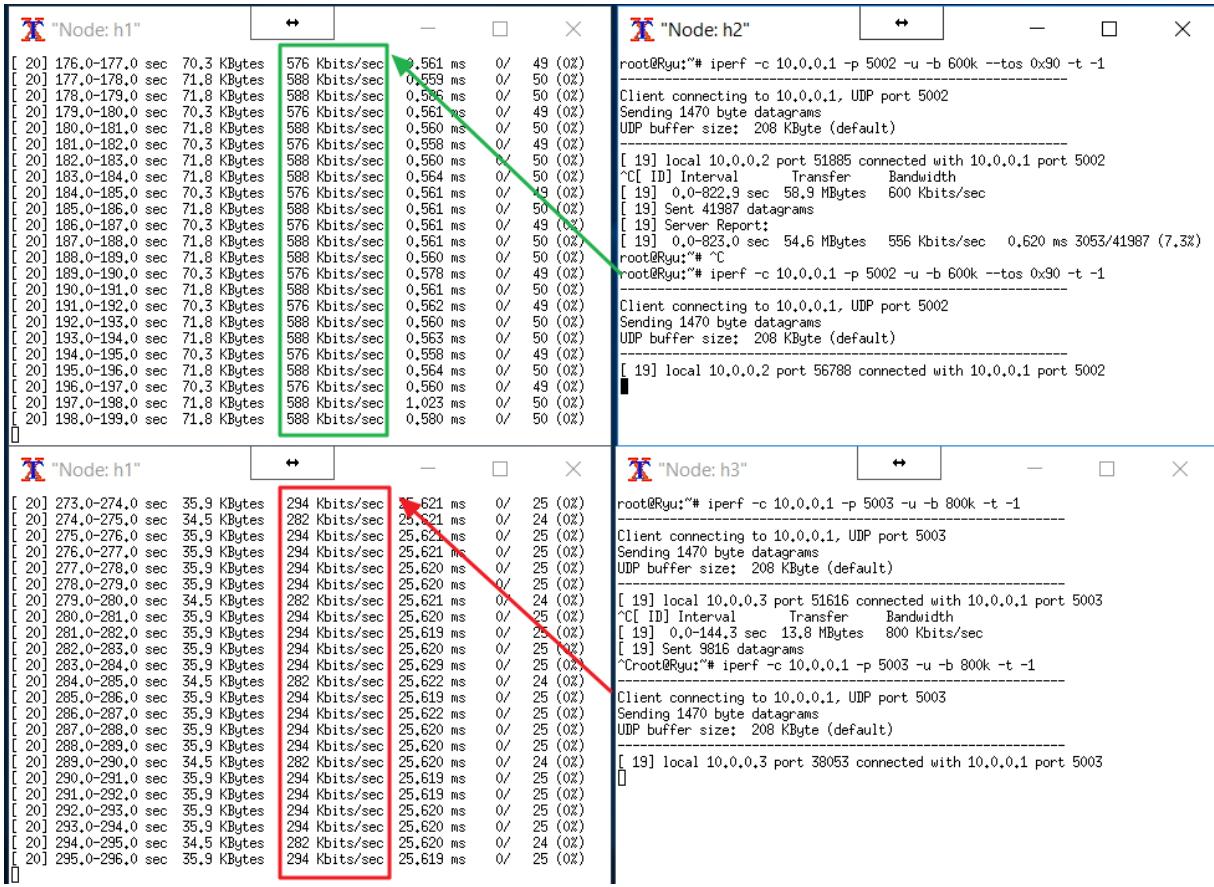


Figure 3.88: Requests to Port 5003 (BE) from Host3 and Port 5002 (AF42) from Host2.

From the above results, we know now that QoS rules are still working with HTB queues assigned to the s1-eth1 interface on Switch1 as well as their corresponding DSCP markings. Meters are not in place for Switch2 and Switch3, but it still uses a class-based approach on ingress interfaces.

Since this time our meters are created correctly on Switch2 and Switch3 with OFSoftSwitch13, we are not going to use iPerf to check the same QoS policies again, as seen in *Figure 3.89*.

```

root@Ryu:/home/ryu# curl -g -X GET http://127.0.0.1:8080/qos/meter/0000000000000000
002
[{"switch_id": "0000000000000002", "command_result": {"2": [{"band_stats": [{"byte_band_count": 0, "packet_band_count": 0}], "len": 56, "duration_nsec": 9000, "duration_sec": 2805, "byte_in_count": 11760, "flow_count": 0, "packet_in_count": 120, "meter_id": 1}]}]root@Ryu:/home/ryu#
root@Ryu:/home/ryu#
root@Ryu:/home/ryu#
root@Ryu:/home/ryu#
root@Ryu:/home/ryu#
root@Ryu:/home/ryu#
root@Ryu:/home/ryu#
root@Ryu:/home/ryu# curl -g -X GET http://127.0.0.1:8080/qos/meter/0000000000000000
003
[{"switch_id": "0000000000000003", "command_result": {"3": [{"band_stats": [{"byte_band_count": 222264, "packet_band_count": 147}], "len": 56, "duration_nsec": 88000, "duration_sec": 2812, "byte_in_count": 792736, "flow_count": 0, "packet_in_count": 543, "meter_id": 1}]}]root@Ryu:/home/ryu#

```

Figure 3.89: Retrieval of QoS Meters on Switch2 and Switch3 with OFSoftSwitch13.

From above *Figure 3.89*, we can see that packets are coming in on Port2 for Switch2 and Port3 for Switch3, while the Switch3 meter also keeps limiting the bandwidth as the packet count is changing. We also noticed that ICMP requests between Host1 and Host2 are successful, but not between Host3 and Host2 as the network is sliced into two separate domains, as seen in *Figure 3.90*.

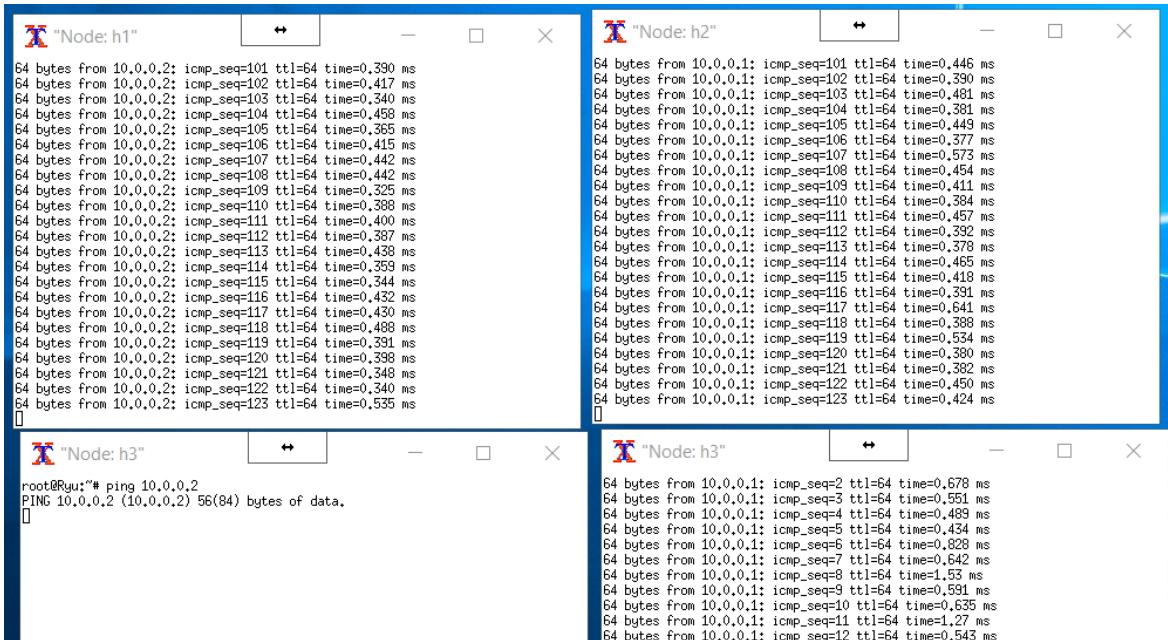


Figure 3.90: ICMP Requests between Hosts in Sliced QoS Topo.

With the experiments on the Meter Table we have proven that it's possible to use the

external controller to remark the traffic until some other app running on the NBI will take care of forwarding, while OF13 will be responsible for QoS rules injection via REST API and OFSoftSwitch13 will take over the role of remarking our DSCP classes bound to specific meters. Our tests cases were mostly based on Ryu, but this type of implementation isn't much different with other controllers. Kumar (2016) discussed how to use OF meters with ODL and OFSoftSwitch13 to create QoS policies with a drop rate of 10 Mbps between two hosts which is very like our scenarios.

Above tests allowed to proof that QoS in MPLS and OpenFlow is possible to achieve with use of traffic classification and DSCP markings. MPLS-TE doesn't have any inbuild mechanism to limit the bandwidth on specific interfaces rather than on the whole VPN channel to limit the overall available bandwidth to the customer, while OF can use HTBs and port numbers to place traffic into different queues.

3.5. Traffic Engineering

3.5.1. MPLS

The dynamic development of the Internet in recent years has caused the efficient use of network infrastructure become a very important issue. The concept of TE descriptively means that all activities are aimed at evaluating and optimizing the performance of computer networks. Internet services such a new kind of image transmission and sound in real-time or interactive online games led to the definition of the term of QoS. ISPs must meet the requirements of customers which is the main reason for the development of traffic engineering.

The problem, however, is the IP on which the Internet is based has a very weak set of tools for effective traffic control. This is one of the reasons why technologies such as MPLS and OpenFlow are designed and developed as this can extend the future possibilities in these terms. The remaining part will show how we can define a non-trivial rule to determinate the flow of traffic on the network.

In these experiments, we will show the possibilities that appear in the field of TE with the use of MPLS or OpenFlow. We will use different configuration environments which constitute a substitute for the real network. The idea is to move traffic to a single destination along different routes depending on its source of origin.

The relevant configurations which will allow replicating the test cases can be found in *Appendix 10*.

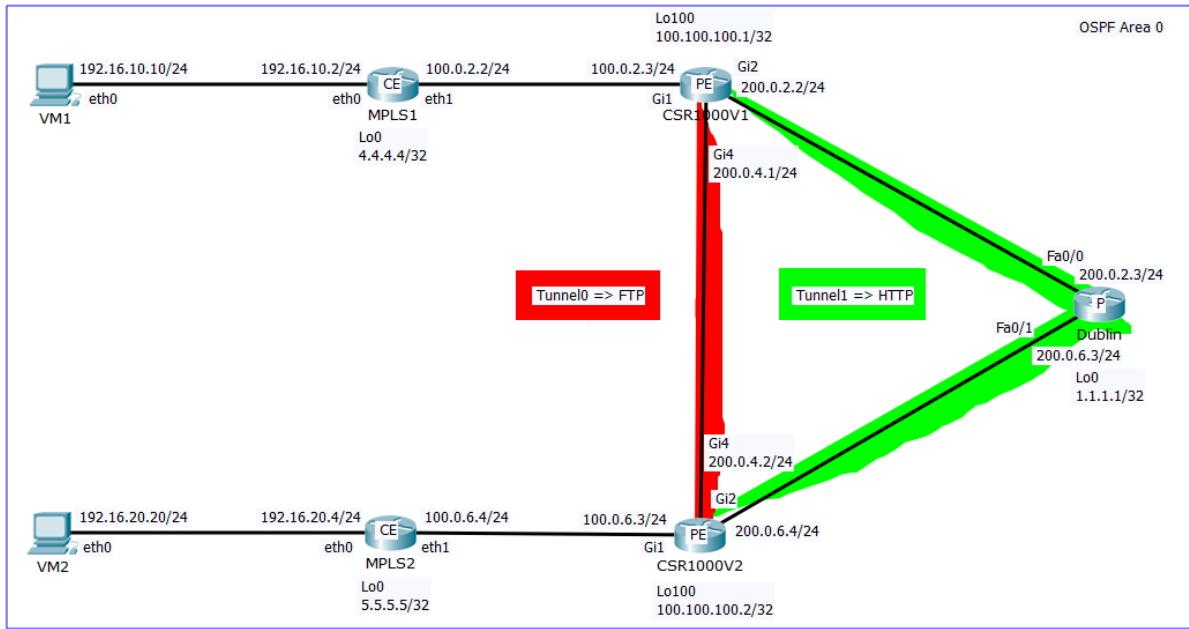


Figure 3.91: MPLS TE Topology.

This topology accurately meets all the assumptions that have been set before we were preparing for the tests. Nodes acting as end-user computers are working in two different subnets and there is no additional equipment to ensure the desired functionality except devices within the Topo. MPLS nodes with FRR run OSPF and LDP as well, as they set DSCP values for forwarded packets to specific port destinations with TCP before reaching Cisco CSRs. Cisco PE routers have two TE tunnels set up and they use PBR and Assured Forwarding (AF) classes to implement the route from an ingress interface to egress tunnels with explicit paths and extended ACLs, as seen in *Figure 3.91*. We also have used DSCP to EXP mappings to forward the packets via a deployed path, as seen in *Figure 3.92*.

Tunnel0	Tunnel1
DSCP: EF	DSCP: CS1
EXP: 5	EXP: 1
ToS: 184	ToS: 32

Figure 3.92: MPLS TE Topology DSCP to EXP Mappings.

To check whether the packets are marked correctly while entering the tunnels we have used ping and ToS values from VM1 to VM2 via `ping 192.16.20.20 -c 5 -Q 184` or `32`, as seen in *Figure 3.93*.

```

CSR1000V1#show route-map
route-map pbr, permit, sequence 10
  Match clauses:
    ip address (access-lists): EF
  Set clauses:
    interface Tunnel0
    Policy routing matches: 5 packets, 490 bytes
route-map pbr, permit, sequence 20
  Match clauses:
    ip address (access-lists): CS1
  Set clauses:
    interface Tunnel1
    Policy routing matches: 5 packets, 490 bytes
CSR1000V1#
CSR1000V2#show route-map
route-map pbr, permit, sequence 10
  Match clauses:
    ip address (access-lists): EF
  Set clauses:
    interface Tunnel0
    Policy routing matches: 5 packets, 490 bytes
route-map pbr, permit, sequence 20
  Match clauses:
    ip address (access-lists): CS1
  Set clauses:
    interface Tunnel1
    Policy routing matches: 5 packets, 490 bytes
CSR1000V2#

```

File Action Media Clipboard View Help

openflow1@openflow1:~

```

openflow1@openflow1:~$ ping 192.16.20.20 -c 5
PING 192.16.20.20 (192.16.20.20) 56(84) bytes of data.
64 bytes from 192.16.20.20: icmp_seq=1 ttl=59 time=1.32 ms
64 bytes from 192.16.20.20: icmp_seq=2 ttl=59 time=1.59 ms
64 bytes from 192.16.20.20: icmp_seq=3 ttl=59 time=1.94 ms
64 bytes from 192.16.20.20: icmp_seq=4 ttl=59 time=1.51 ms
64 bytes from 192.16.20.20: icmp_seq=5 ttl=59 time=1.14 ms

--- 192.16.20.20 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 1.146/1.504/1.947/0.274 ms
openflow1@openflow1:~$ ping 192.16.20.20 -c 5 -Q 184
PING 192.16.20.20 (192.16.20.20) 56(84) bytes of data.
64 bytes from 192.16.20.20: icmp_seq=1 ttl=60 time=1.00 ms
64 bytes from 192.16.20.20: icmp_seq=2 ttl=60 time=1.10 ms
64 bytes from 192.16.20.20: icmp_seq=3 ttl=60 time=0.769 ms
64 bytes from 192.16.20.20: icmp_seq=4 ttl=60 time=1.28 ms
64 bytes from 192.16.20.20: icmp_seq=5 ttl=60 time=0.889 ms

--- 192.16.20.20 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4008ms
rtt min/avg/max/mdev = 0.769/1.010/1.282/0.176 ms
openflow1@openflow1:~$ ping 192.16.20.20 -c 5 -Q 32
PING 192.16.20.20 (192.16.20.20) 56(84) bytes of data.
64 bytes from 192.16.20.20: icmp_seq=1 ttl=59 time=1.31 ms
64 bytes from 192.16.20.20: icmp_seq=2 ttl=59 time=1.71 ms
64 bytes from 192.16.20.20: icmp_seq=3 ttl=59 time=2.68 ms
64 bytes from 192.16.20.20: icmp_seq=4 ttl=59 time=8.74 ms
64 bytes from 192.16.20.20: icmp_seq=5 ttl=59 time=3.25 ms

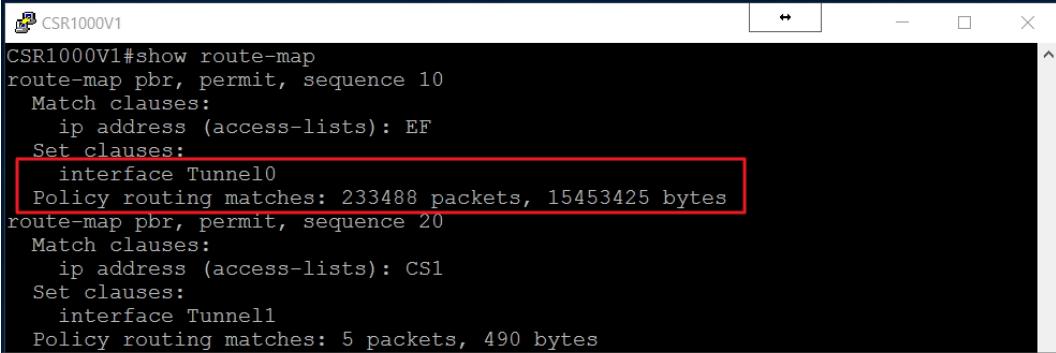
--- 192.16.20.20 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 1.318/3.544/8.746/2.689 ms
openflow1@openflow1:~$ 

```

Figure 3.93: ICMP Test with ToS and TE.

We can see that five packets were sent to ToS 184 and 32 were properly accounted against PBR mappings on both routers while the other traffic is traversing to the destination through the prioritized *Tunnel1*.

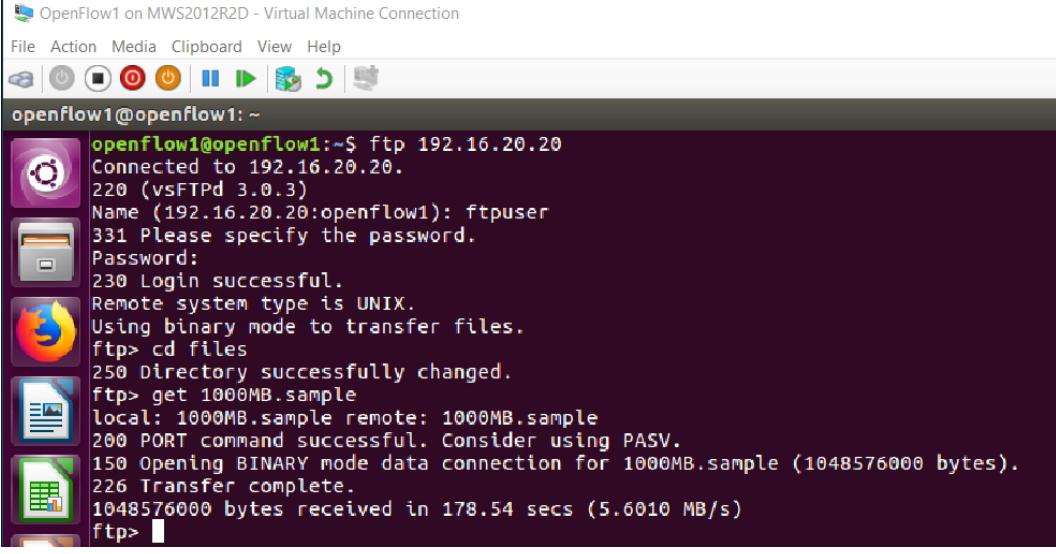
In further tests, we have downloaded a 1 GB sample file from VM2 out of FTP and the Web servers being hosted on this machine. Since the received packet was accounted on CSR1000V1 we were able to prove that the packets are correctly routed via mapped TE tunnel to a specific protocol, as seen in *Figure 3.94*.



```

CSR1000V1#show route-map
route-map pbr, permit, sequence 10
  Match clauses:
    ip address (access-lists): EF
  Set clauses:
    interface Tunnel0
  Policy routing matches: 233488 packets, 15453425 bytes
route-map pbr, permit, sequence 20
  Match clauses:
    ip address (access-lists): CS1
  Set clauses:
    interface Tunnell
  Policy routing matches: 5 packets, 490 bytes

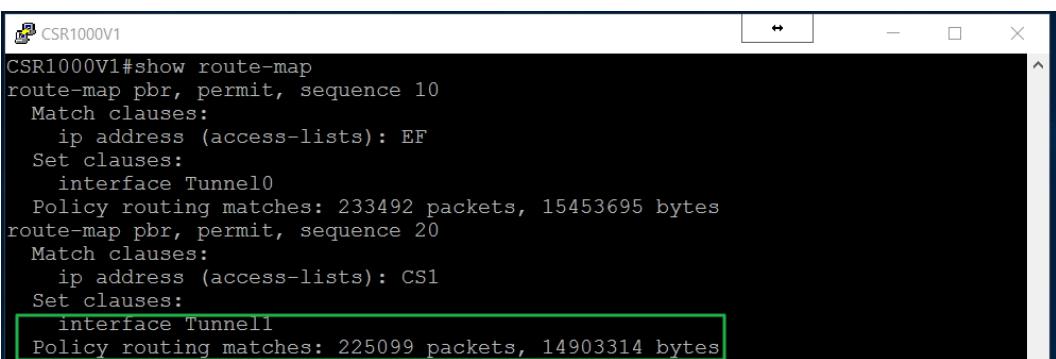
```



```

openflow1@openflow1:~$ ftp 192.16.20.20
Connected to 192.16.20.20.
220 (vsFTPd 3.0.3)
Name (192.16.20.20:openflow1): ftpuser
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd files
250 Directory successfully changed.
ftp> get 1000MB.sample
local: 1000MB.sample remote: 1000MB.sample
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for 1000MB.sample (1048576000 bytes).
226 Transfer complete.
1048576000 bytes received in 178.54 secs (5.6010 MB/s)
ftp>

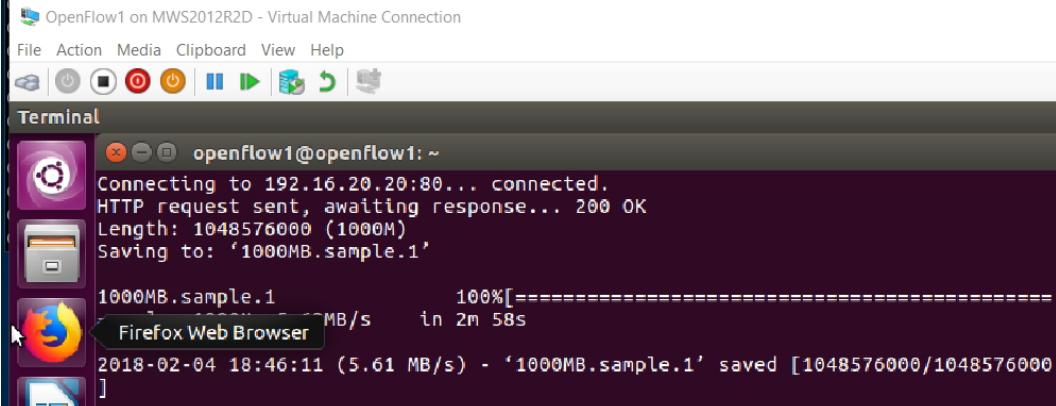
```



```

CSR1000V1#show route-map
route-map pbr, permit, sequence 10
  Match clauses:
    ip address (access-lists): EF
  Set clauses:
    interface Tunnel0
  Policy routing matches: 233492 packets, 15453695 bytes
route-map pbr, permit, sequence 20
  Match clauses:
    ip address (access-lists): CS1
  Set clauses:
    interface Tunnell
  Policy routing matches: 225099 packets, 14903314 bytes

```



```

Terminal
openflow1@openflow1:~
Connecting to 192.16.20.20:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1048576000 (1000M)
Saving to: '1000MB.sample.1'

1000MB.sample.1          100%[=====] 1048576000 5.61 MB/s
[Firefox Web Browser]   in 2m 58s
2018-02-04 18:46:11 (5.61 MB/s) - '1000MB.sample.1' saved [1048576000/1048576000]

```

Figure 3.94: FTP and HTTP Tests with DSCP to EXP Markings and TE.

To further investigate this, we have checked the DSCP class marking on MPLS1 and MPLS2 for forwarded TCP packets between CE and PE nodes, as seen in *Figure 3.95*.

```

[mpls1@mpls1:~]$ sudo iptables -t mangle -L -v
[sudo] password for mpls1:
Chain PREROUTING (policy ACCEPT 2140K packets, 4036M bytes)
pkts bytes target prot opt in     out      source          destination
Chain INPUT (policy ACCEPT 87823 packets, 6573K bytes)
pkts bytes target prot opt in     out      source          destination
Chain FORWARD (policy ACCEPT 2025K packets, 4027M bytes)
pkts bytes target prot opt in     out      source          destination
  342K  18M DSCP    tcp  --  any    any   anywhere        anywhere       tcp dpt:ftp-data DSCP set 0x2e
  157  9889 DSCP   tcp  --  any    any   anywhere        anywhere       tcp dpt:ftp DSCP set 0x2e
  512K  27M DSCP   tcp  --  any    any   anywhere        anywhere       tcp dpt:http DSCP set 0x08

[mpls2@mpls2:~]$ sudo iptables -t mangle -L -v
[sudo] password for mpls2:
Chain PREROUTING (policy ACCEPT 1576K packets, 4008M bytes)
pkts bytes target prot opt in     out      source          destination
Chain INPUT (policy ACCEPT 88180 packets, 6598K bytes)
pkts bytes target prot opt in     out      source          destination
Chain FORWARD (policy ACCEPT 1462K packets, 3999M bytes)
pkts bytes target prot opt in     out      source          destination
  342K  18M DSCP    tcp  --  any    any   anywhere        anywhere       tcp dpt:ftp-data DSCP set 0x2e
  157  9889 DSCP   tcp  --  any    any   anywhere        anywhere       tcp dpt:ftp DSCP set 0x2e
  512K  27M DSCP   tcp  --  any    any   anywhere        anywhere       tcp dpt:http DSCP set 0x08

```

Figure 3.95: DSCP Markings for Forwarded TCP Packets on CE Nodes.

By the number of logged packets and the data transferred across to the TCP destination ports, we can assume that DSCP is marked according to the specified protocols for FTP, FTP-Data, and HTTP.

The above test cases have proven that it's possible to effectively use Linux and FRR together with Cisco equipment, which uses PBR to perform MPLS TE with LDP and RSVP to route traffic via tunnels depending on the protocol type.

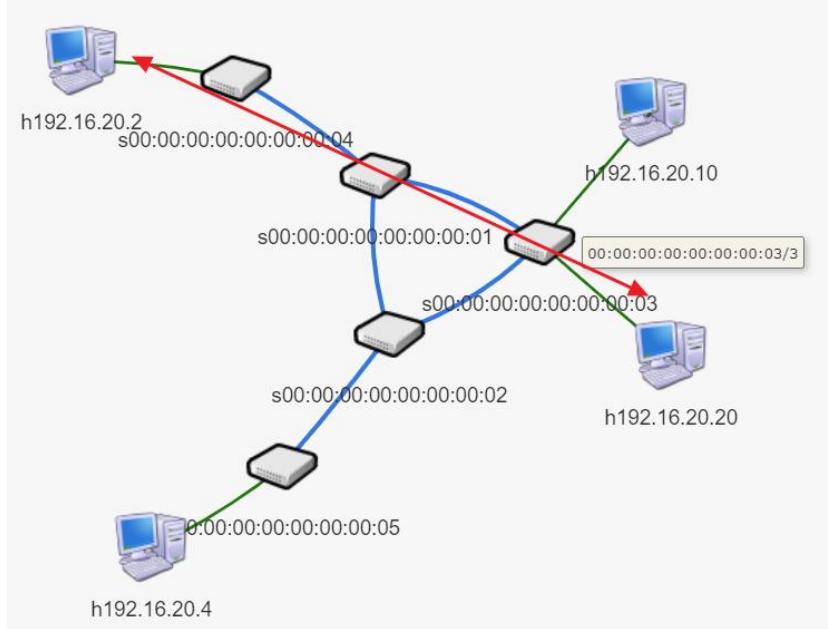
3.5.2. OpenFlow

The situation looks a little bit different when we want to test the same topology with OpenFlow. The devices acting as software switches are the VMs with the right set of tools and a sufficient number of vNICs.

It was necessary however to consider that the switches programmed using the OpenFlow protocol are working in the data link layer and this therefore requires that the communicating nodes were within the same subnet. In OF tests we will use non-commercial controller called Floodlight and commercial HPE controller previously discussed in *Chapter 2.6.3*. We are also going to discuss the potential use of other open source OpenDaylight controller for large-scale environment with Cisco applications.

In the presented topology we must look at the IP addresses assigned to the h192.16.20.10 and h192.16.20.20 on Port3 of Switch3. These IP addresses are the external hosts bridged via the eth1 interface in Mininet. Nodes h192.16.20.2 and h192.16.20.4, during this experiment, will

serve as internal end-users. Port3 of Switch3, in this case, plays the roles of the boundary for OpenFlow domain. Such a solution has been used because we are trying to achieve similar topologies to MPLS and at the same time reduce the number of physical devices used, without affecting the results obtained from the experiments.



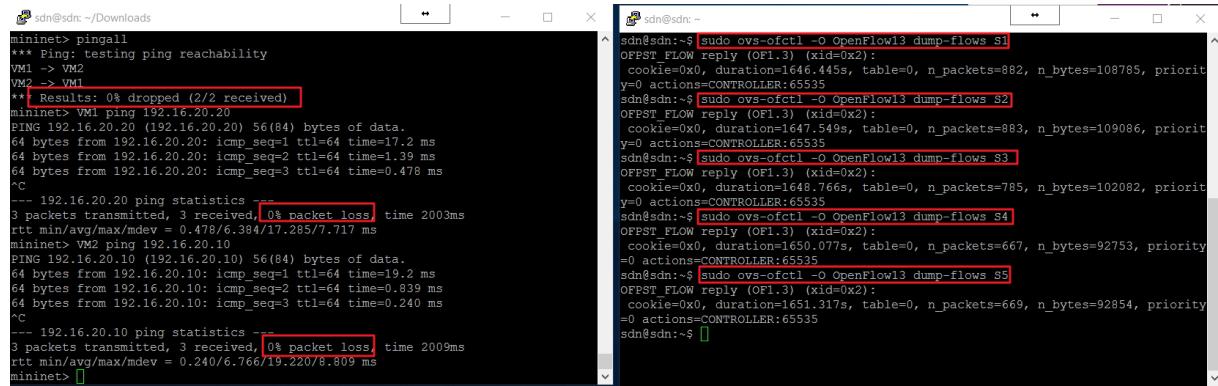
```
sdn@sdn:~/Downloads$ sudo ./OpenFlow3.py
*** Connecting to hw intf: eth1*** Checking eth1
*** Creating network
*** Adding hardware interface eth1 to switch S3
*** Note: Reconfigure the interfaces for the Mininet hosts:
[<Host VM1: VM1-eth0:None pid=36311> , <Host VM2: VM2-eth0:None pid=36313> ]
*** Configuring hosts
VM1 VM2
*** Starting controller
Floodlight
*** Starting 5 switches
S1 S2 S3 S4 S5 ...
sdn@sdn:~$ sudo ovs-vsctl show
f2408541-6908-48b7-bcdd-2843daca80e6
Bridge "S3"
    Controller "tcp:192.16.20.31:6653"
        is_connected: true
    fail_mode: secure
    Port "S3-eth2"
        Interface "S3-eth2"
    Port "S3"
        Interface "S3"
            type: internal
    Port "S3-eth1"
        Interface "S3-eth1"
    Port "eth1"
        Interface "eth1"
```

Figure 3.96: OF Topology with Floodlight without TE.

In this scenario, we are using OF13 and an external controller called Floodlight previously discussed in *Chapter 2.6.2* running on a separate VM with an IP address of 192.16.20.31

connected to the OVS via port 6653.

Since core switches have multiple links, STP will determinate the shortest path which in our experiment will be made during the requests from h192.16.20.2 to h192.16.20.20 and it will forward traffic from Switch4 to Switch1, then to Switch3 before reaching the destination. We can also see that there are no flow entries present on any switches except for an entry which forwards all requests to the controller and that we can reach all the hosts in the Topo, as seen in *Figure 3.97*.



The screenshot shows two terminal windows. The left window is a mininet terminal where the user runs pingall to test connectivity between VM1 and VM2. The right window is a host terminal (sdn) running the command sudo ovs-ofctl -O OpenFlow13 dump-flows S1 through S4. The output shows flow entries for each switch, each with a cookie of 0x0 and duration of 0. The flows forward packets to the controller (actions=CONTROLLER) and have priority 0.

```

mininet> pingall
*** Ping: testing ping reachability
VM1 -> VM2
VM2 -> VM1
** Results: 0% dropped (2/2 received)
mininet> VM1 ping 192.16.20.20
PING 192.16.20.20 (192.16.20.20) 56(84) bytes of data.
64 bytes from 192.16.20.20: icmp_seq=1 ttl=64 time=17.2 ms
64 bytes from 192.16.20.20: icmp_seq=2 ttl=64 time=1.39 ms
64 bytes from 192.16.20.20: icmp_seq=3 ttl=64 time=0.478 ms
^C
--- 192.16.20.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.478/6.384/17.285/7.717 ms
mininet> VM2 ping 192.16.20.10
PING 192.16.20.10 (192.16.20.10) 56(84) bytes of data.
64 bytes from 192.16.20.10: icmp_seq=1 ttl=64 time=19.2 ms
64 bytes from 192.16.20.10: icmp_seq=2 ttl=64 time=0.839 ms
64 bytes from 192.16.20.10: icmp_seq=3 ttl=64 time=0.240 ms
^C
--- 192.16.20.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2009ms
rtt min/avg/max/mdev = 0.240/6.766/19.220/8.809 ms
mininet> [REDACTED]

```

```

sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows S1
OFPT_FLOW_REPLY (OF1.3) (xid=0x2):
cookie=0x0, duration=1646.445s, table=0, n_packets=882, n_bytes=108785, priority=0 actions=CONTROLLER:65535
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows S2
OFPT_FLOW_REPLY (OF1.3) (xid=0x2):
cookie=0x0, duration=1647.549s, table=0, n_packets=883, n_bytes=109086, priority=0 actions=CONTROLLER:65535
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows S3
OFPT_FLOW_REPLY (OF1.3) (xid=0x2):
cookie=0x0, duration=1648.766s, table=0, n_packets=785, n_bytes=102082, priority=0 actions=CONTROLLER:65535
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows S4
OFPT_FLOW_REPLY (OF1.3) (xid=0x2):
cookie=0x0, duration=1650.077s, table=0, n_packets=667, n_bytes=92753, priority=0 actions=CONTROLLER:65535
sdn@sdn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows S5
OFPT_FLOW_REPLY (OF1.3) (xid=0x2):
cookie=0x0, duration=1651.317s, table=0, n_packets=669, n_bytes=92854, priority=0 actions=CONTROLLER:65535
sdn@sdn:~$ [REDACTED]

```

Figure 3.97: ICMP Requests between Hosts and Flow Entries on the Switches.

Now we are going to use iPerf to capture three measures of the UDP delay between these two hosts where results will be saved on the h192.16.20.20 (OpenFlow2) VM in the *INTtoEXT1* CSV file. To do this we need to execute the iPerf Server with *iperf -s -u -i 1 192.16.20.20 / tee INTtoEXT1.csv* and connect to it from VM1 via *iperf -c 192.16.20.20 -u -t 60* to perform the test for 60 seconds. The mean value of these tests was 0.136 ms with StDev of 0.06.

The discussed case was performed on a Topo without TE, so we have decided to use Static Entry Pusher API (Izard, 2017) to set up flows on switches as follows from both hosts:

- "ipv4_src":"192.16.20.2" to "ipv4_dst":"192.16.20.20" for "eth_type":"0x800" (IPv4)
S4-eth1 => S4-eth2 => S1-eth3 => S1-eth2 => S2-eth2 => S2-eth1 => S3-eth2 => S3-eth3
- "ipv4_src":"192.16.20.20" to "ipv4_dst":"192.16.20.2" "eth_type":"0x800" (IPv4)
S3-eth3 => S3-eth2 => S2-eth1 => S2-eth2 => S1-eth2 => S1-eth3 => S4-eth2 => S4-eth1

This will force all IPv4 traffic between these hosts to take a specific route through the Topo, as seen in *Figure 3.98*.

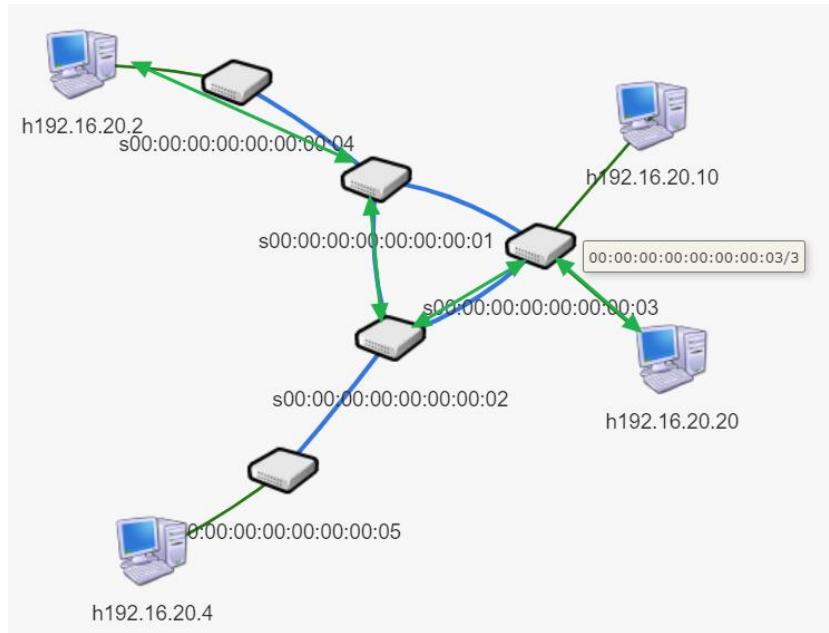


Figure 3.98: OF Topology with Floodlight with TE.

After the successful execution of the Python script, we can see all of the entries on the participating switches and the assigned priority of 600 which is higher than the entry responsible for pointing to the controller for forwarding, as seen in *Figure 3.99*.

Figure 3.99: Flow Entries on the Switches with TE.

We have also generated a high amount of traffic by downloading sample files out of FTP and HTTP servers hosted on h192.16.20.20 to make sure that entire flow will match their actions. This reassured us in the settings as we noticed a very high volume of packets being reported for Port2 on Switch3 which was caused by the request made from h192.16.20.2, a seen in *Figure 3.100*.

The screenshot displays three main sections of the Floodlight and TE interface:

- Switch3 Details:** A table showing hardware and software information for Switch3. The MAC address is highlighted with a red box.
- Flow Count:** A table showing various statistics for flows, including Flow Count (3), Packet Count (101148), Byte (4201222626), Flag (empty), Buffer (256), and Table Count (254).
- Port Table:** A table showing port statistics for the 'local' port. It includes columns for No, R. Packets, Tran. Packets, R. Bytes, Tran. Bytes, R. Dropped, Tran. Dropped, Coll., and Duration(s). The total duration is 7229ms.
- Flow Table:** A table showing static flow entries. Two entries are highlighted with red boxes: one for Table No 0x0 with Priority 600 and another for Table No 0x0 with Priority 600. Both entries have flags SEND_FLOW_Rem and output values 3 and 2 respectively.

Figure 3.100: Switch3 Details with Floodlight and TE.

Irrespective of the previous experiment, we have executed tests to measure the delay after TE was implemented to use a longer path to the destination, where the results were saved to a file named *INTtoEXT2*. The comparison of the results can be seen in *Figure 3.101*.

Method	Without TE (ms)	With TE (ms)
Mean	0.136	0.184
StDev	0.060	0.112

Figure 3.101: Comparison of Delay without and with TE.

From the results, we can observe that the delay is higher by 35% as expected due to a longer route to the destination by one hop count as the initial route had four hop counts between the hosts.

In the next scenario, we have used the HPE Aruba VAN controller to load the same Topo and to perform initial UDP delay tests since the STP algorithm has calculated an identical path to Switch3 from VM1 which is four hops away, as seen in *Figure 3.102*.

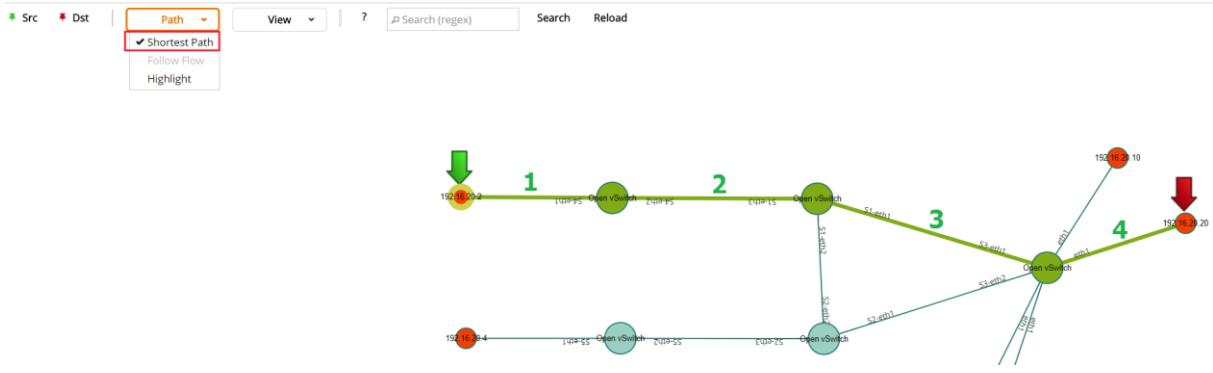


Figure 3.102: OF Topology with HPE Aruba VAN without TE.

This time the results were saved to the *INTtoEXT3* CSV file and the mean value was 0.126 ms with StDev of 0.043.

After that, we had to retrieve the token from the controller as discussed in *Chapter 1.4.10* before we were able to create a Bash script (HPE, 2016) which would alter the path of the IPv4 traffic via REST API to use a longer route to OpenFlow2 VM, as seen in *Figure 3.103*.

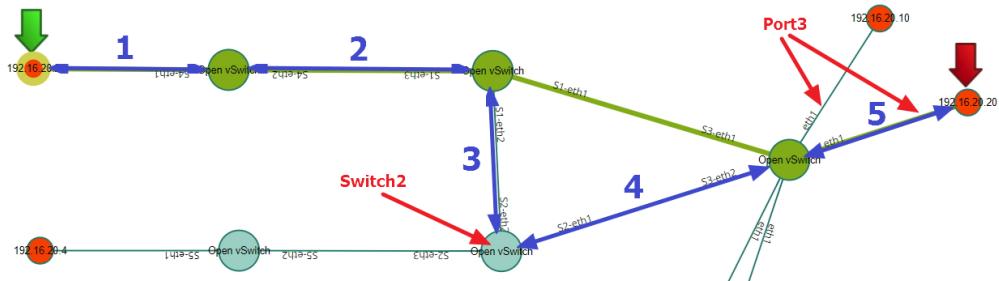


Figure 3.103: OF Topology with HPE Aruba VAN with TE.

Now traffic from 192.16.20.2 moves across the path where there are five hops to the destination IP address 192.16.20.20 and this could be validated by looking at the DPID 00:00:00:00:00:00:02 of Switch2 which normally wouldn't participate in the transfer of data between these two hosts, as seen in *Figure 3.104*.

Flows for Data Path ID: 00:00:00:00:00:00:02				
				Flows
Table ID	Flow Count	Table Name		Flow Class ID
0	26	classifier		
▶ 60001	90482	Packets Bytes 5248861804	Match eth_type: ipv4 ipv4_src: 192.16.20.20 ipv4_dst: 192.16.20.20	Actions/Instructions apply_actions: output: 2
▶ 60001	67139	50865581	eth_type: ipv4 ipv4_src: 192.16.20.2 ipv4_dst: 192.16.20.20	apply_actions: output: 1
▶ 60000	6	426	eth_type: bddp	apply_actions: output: com.hp.sdn.bddp.steal
▶ 50301	0	0	eth_type: ipv4 ip_proto: udp udp_dst: 53	apply_actions: output: CONTROLLER
▶ 50300	0	0	eth_type: ipv4 ip_proto: udp udp_dst: 53	apply_actions: output: CONTROLLER

Figure 3.104: Flow Entries with HPE Aruba VAN of Switch2 with TE.

From *Figure 3.104* we can see that Switch2 has our flow entry with a priority of 60001 and that the requests made from VM1 generated most of the traffic on the outgoing Port2 (S2-eth2).

However, when we inspect Switch3 we can distinguish that the port number displayed in the HPE GUI is different than the actual port as it's the bridged interface name rather than vNIC, as seen in *Figure 3.105*.

Flows for Data Path ID: 00:00:00:00:00:00:03						
	Table ID	Flow Count	Table Name	Match	Actions/Instructions	Flow Class ID
0	Priority	25	classifier	eth_type: ipv4 ip4_src: 192.16.20.20 ip4_dst: 192.16.20.2	apply_actions: output: 2	
	60001	90482	5248861804	eth_type: ipv4 ip4_src: 192.16.20.2 ip4_dst: 192.16.20.20	apply_actions: output: 3	
	60001	67139	50865581	eth_type: ipv4 ip4_src: 192.16.20.2 ip4_dst: 192.16.20.20	apply_actions: output: 3	
	60000	4	284	eth_type: bddp	apply_actions: output: CONTROLLER	corn.hp.sdn.bddp.steal
	50301	0	0	eth_type: ipv4 ip_proto: udp udp_dst: 53	apply_actions: output: CONTROLLER	
	50300	0	0	eth_type: ipv4 ip_proto: udp udp_dst: 53	apply_actions: output: CONTROLLER	

```

Bridge "S3"
Controller "tcp:192.16.20.254:6633"
is_connected: true
fail_mode: secure
Port "S3-eth1"
Interface "S3-eth1"
Port "eth1"
Interface "eth1"
Port "S3-eth2"
Interface "S3-eth2"
Port "S3"
Interface "S3"
type: internal
  
```

Figure 3.105: Flow Entries with HPE Aruba VAN of Switch3 with TE.

The table below displays the UDP delay between both SDN controllers with identical Topo and test cases.

Method	SDN Controller	Without TE (ms)	With TE (ms)
Mean	Floodlight	0.136	0.184
	HPE	0.126	0.130
StDev	Floodlight	0.060	0.112
	HPE	0.043	0.073

Figure 3.106: Comparison of Delay without and with TE between Floodlight and HPE Controllers.

By comparing both of the SDN controllers we can see that HPE performs better than Floodlight as both scenarios with and without TE resulted in a lower mean and StDev for the jitter parameter. HPE controller without TE appeared to be 7 % faster and 29 % more efficient with TE in comparison to Floodlight controller with a difference of one hop between the client and server. This could be caused because HPE is a commercial controller, however, this explains how different SDN implementations can impact the network performance on a larger scale.

Another good example of TE with a large-scale environment would be the use of an ODL controller and open source Pathman apps developed by Cisco used for the creation of

LSPs with BGP (Cisco DevNet GitHub, 2015) and Segment Routing (Cisco DevNet GitHub, 2016) to enforce the flow path through its domain at the ingress node via RESTCONF APIs on the NBI. Due to hardware limitations, to explain the use of Pathman apps with ODL and BGP-LS we are going to use Cisco dCloud demo (Cisco dCloud, 2017). In this Virtual Internet Routing Lab (VIRL) environment, we will be able to use common infrastructure used by the SPs with multiple links consisting of eight XRv routers (Cisco, 2014) which run within together with ODL Carbon SR1 release on separate VMs hosted in VMware ESXi hypervisor.

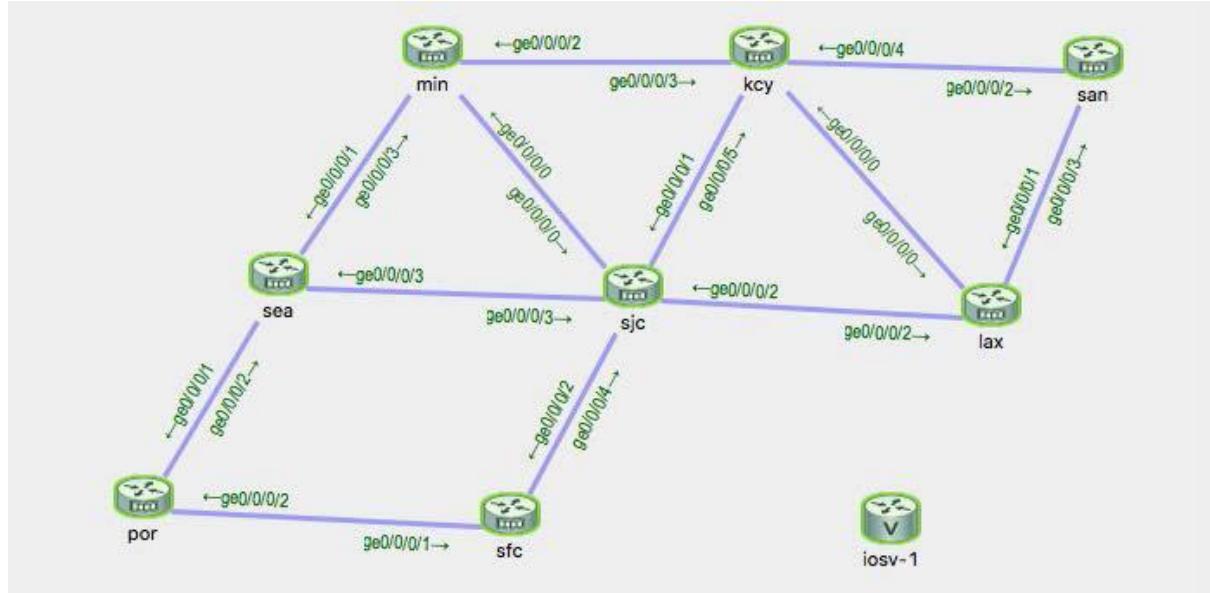


Figure 3.107: Demo SP Topology.

The creation of LSPs with Pathman apps through the Topo isn't complex as we only need to select a source and a destination to see the metrics based on the protocol or number of hops, so we are able to create a path between san and sfc nodes within seconds, as seen in *Figure 3.108*.

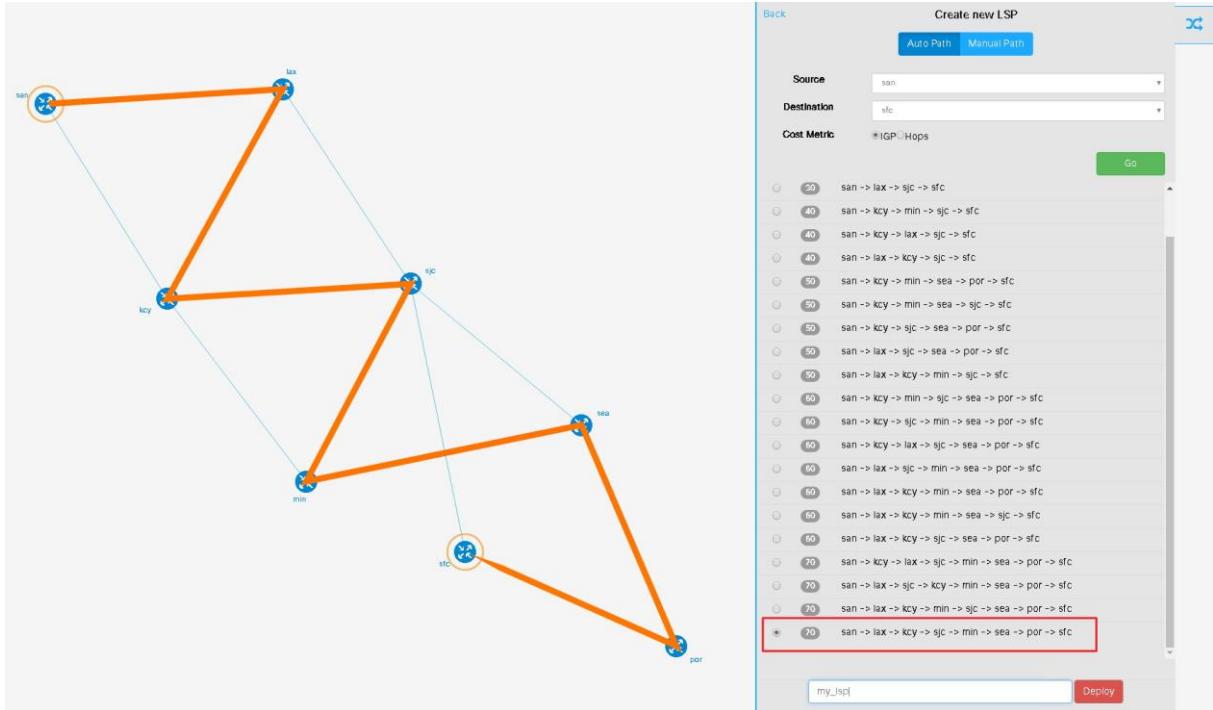


Figure 3.108: Creation of LSP with Pathman Apps and BGP.

To segment the network with TE tunnels and Pathman SR respectively we have selected a source and a destination, then the Path Computation Element Protocol (PCEP) was used to calculate the path between the endpoints before the deployment, as seen in *Figure 3.109*.

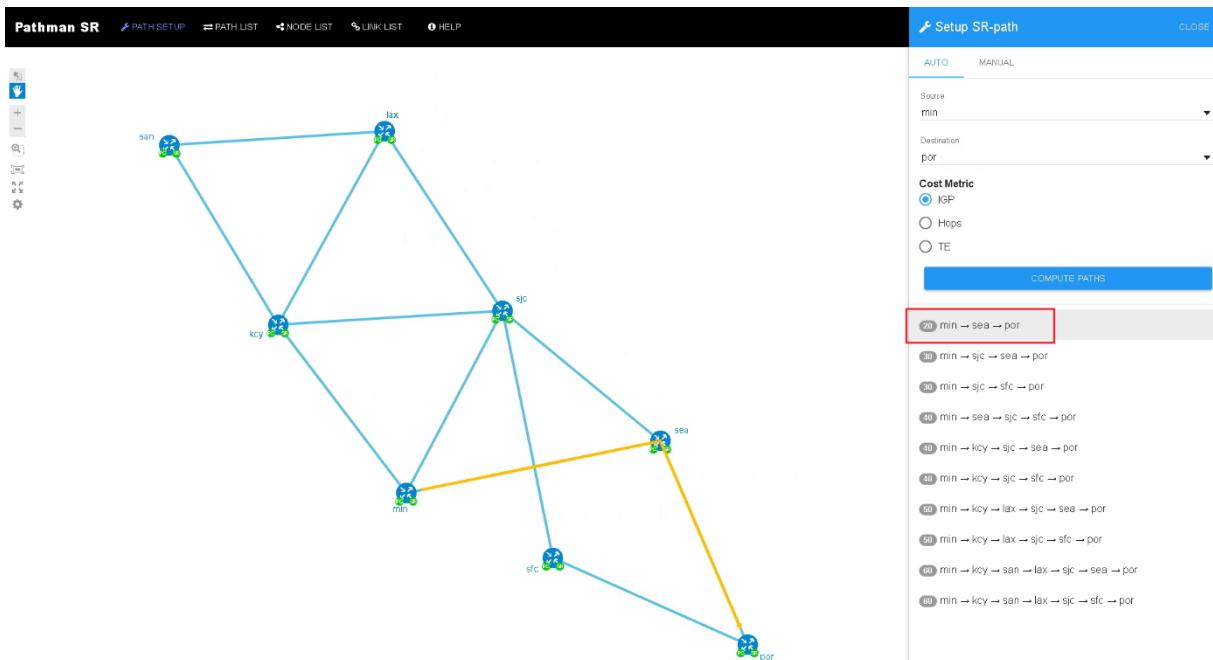


Figure 3.109: Point to Point Path Calculation with PCEP.

Pathman simply sends the path to the ODL and connects to the routers while the management of this tunnel is delegated to the controller. Segment IDs are the actual MPLS labels used for distribution of the labels via LDP, so in our example, the min uses label 16028 for sea router

and label 16026 is used for por node, as seen in *Figure 3.110*.

Summary			
Path name	min -> por		
Route	min → sea → por		
Route			MODIFY
Router	IP	SID	
min	198.19.1.24	-	
sea	198.19.1.28	16028	
por	198.19.1.26	16026	

```

RP/0/0/CPU0:min#show ip int brief
Mon Jan 29 18:29:03.264 UTC
Interface          IP-Address      Status    Protocol Vrf-Name
Loopback0          198.19.1.24    Up       default
tunnel-tel         198.19.1.24    Up       default
MgmtEth0/0/CPU0/0  198.18.1.32    Up       default
GigabitEthernet0/0/0/0 57.0.0.24    Up       default
GigabitEthernet0/0/0/1 40.0.0.24    Up       default
GigabitEthernet0/0/0/2 44.0.0.24    Up       default
GigabitEthernet0/0/0/3 51.0.0.24    Up       default

RP/0/0/CPU0:min#show mpls traffic-eng tunnels | include tunnel
Mon Jan 29 18:29:16.703 UTC
Name: tunnel-tel Destination: 198.19.1.26 Ifhandle:0x70 (auto-tunnel pcc)
RP/0/0/CPU0:min#show mpls traffic-eng tunnels | include Segment
Mon Jan 29 18:29:29.892 UTC
    path option 10, (Segment-Routing type explicit (autopcc_tel)) (Basis for Setup)
    Segment-Routing Path Info (PCE controlled)
        Segment0[Node]: 198.19.1.28, Label: 16028
        Segment1[Node]: 198.19.1.26, Label: 16026
RP/0/0/CPU0:min#show mpls forwarding labels 16028
Mon Jan 29 18:29:38.322 UTC
Local Outgoing Prefix           Outgoing      Next Hop      Bytes
Label Label or ID            Interface
-----+-----+-----+-----+-----+-----+
16028 Pop   SR Pfx (idx 28)   Gi0/0/0/3  51.0.0.28  1072
RP/0/0/CPU0:min#show mpls forwarding labels 16026
Mon Jan 29 18:29:43.772 UTC
Local Outgoing Prefix           Outgoing      Next Hop      Bytes
Label Label or ID            Interface
-----+-----+-----+-----+-----+
16026 16026 SR Pfx (idx 26)   Gi0/0/0/3  51.0.0.28  0
RP/0/0/CPU0:min#

```

Figure 3.110: MPLS Labels between Segmented Endpoints.

Since the segment tunnels are controlled by the ODL we are not able to see any information on min node, while we can still see the old tunnel created with the Pathamn apps called *my_lsp* which used the LSP approach where we would use next hop IP address on the node to program the path via the network, as seen in *Figure 3.111*.

```

RP/0/0/CPU0:sea#show mpls traffic-eng tunnel
Mon Jan 29 18:41:12.915 UTC

LSP Tunnel 198.19.1.27_1 [2] is signalled, Signaling State: up
Tunnel Name: my lsp Tunnel Role: Mid
InLabel: GigabitEthernet0/0/0/1, 24006
OutLabel: GigabitEthernet0/0/0/2, 24004
Signalling Info:
    Src 198.19.1.27 Dst 198.19.1.29, Tun ID 1, Tun Inst 2, Ext ID 198.19.1.27
    Router-IDs: upstream 198.19.1.24
                 local 198.19.1.28
                 downstream 198.19.1.26
    Bandwidth: 0 kbps (CT0) Priority: 7 7 DSTE-class: 0
    Soft Preemption: None
    SRLGs: not collected
    Path Info:
        Incoming Address: 51.0.0.28
        Incoming:
        Explicit Route:
            Strict, 51.0.0.28
            Strict, 53.0.0.26
            Strict, 54.0.0.29
        Outgoing:
            Explicit Route:
                Strict, 53.0.0.26
                Strict, 54.0.0.29
    Record Route: Disabled
    Tspec: avg rate=0 kbytes, burst=1000 bytes, peak rate=0 kbytes
    Session Attributes: Local Prot: Not Set, Node Prot: Not Set, BW Prot: Not Set
                         Soft Preemption Desired: Not Set
    Resv Info: None
    Record Route: Disabled
    Ftspec: avg rate=0 kbytes, burst=1000 bytes, peak rate=0 kbytes
    Displayed 0 (of 0) heads, 1 (of 1) midpoints, 0 (of 0) tails
    Displayed 0 up, 0 down, 0 recovering, 0 recovered heads

```

Figure 3.111: Signaling for *my_lsp* on sea Node.

To retrieve the information about PCEP topology we can use HTTP get method to the ODL controller, as seen in *Figure 3.112*.

```
},
"ip-address": "198.19.1.24",
"reported-lsp": [
{
  "name": "min -> por",
  "path": [
    {
      "lsp-id": 2,
      "ero": {
        "processing-rule": false,
        "subobject": [
          {
            "loose": false,
            "odl-pcep-segment-routing:m-flag": true,
            "odl-pcep-segment-routing:c-flag": false,
            "odl-pcep-segment-routing:sid": 16028,
            "odl-pcep-segment-routing:ip-address": "198.19.1.28",
            "odl-pcep-segment-routing:sid-type": "ipv4-node-id"
          },
          {
            "loose": false,
            "odl-pcep-segment-routing:m-flag": true,
            "odl-pcep-segment-routing:c-flag": false,
            "odl-pcep-segment-routing:sid": 16026,
            "odl-pcep-segment-routing:ip-address": "198.19.1.26",
            "odl-pcep-segment-routing:sid-type": "ipv4-node-id"
          }
        ]
      }
    }
  ]
}
```

Figure 3.112: PCEP Topology out of ODL Controller.

We also have added in a new TE tunnel to por's Loopback1 IP address 11.11.4.1/32 from min node to check whether the packets will follow the SR path and as we can see, traceroute min router has used the correct label 16026 to forward data to the GigabitEthernet0/0/0/1 interface, as seen in *Figure 3.113*.

```

RP/0/0/CPU0:min#show route | include tunnel
Mon Jan 29 18:32:31.184 UTC
RP/0/0/CPU0:min#configure terminal
Mon Jan 29 18:32:41.533 UTC
Current Configuration Session Line      User      Date          Lock
00000000-000cf123-00000002  netconf   cisco   Mon Jan 29 18:04:00 2018
RP/0/0/CPU0:min(config)#router static address-family ipv4 unicast 11.11.4.0/24 tunnel-te1
RP/0/0/CPU0:min(config)#commit
Mon Jan 29 18:33:28.521 UTC
RP/0/0/CPU0:min(config)#end
RP/0/0/CPU0:min#show route | include tunnel
Mon Jan 29 18:33:40.900 UTC
S 11.11.4.0/24 is directly connected, 00:00:12, tunnel-te1
RP/0/0/CPU0:min#

```

```

RP/0/0/CPU0:por#configure terminal
Mon Jan 29 18:36:07.365 UTC
Current Configuration Session Line      User      Date          Lock
00000000-000cf123-00000000  netconf   cisco   Mon Jan 29 16:03:38 2018
RP/0/0/CPU0:por(config)#interface loopback 1
RP/0/0/CPU0:por(config-if)#ipv4 address 11.11.4.1/32
RP/0/0/CPU0:por(config-if)#no shutdown
RP/0/0/CPU0:por(config-if)#commit
Mon Jan 29 18:36:47.650 UTC
RP/0/0/CPU0:por(config-if)#end
RP/0/0/CPU0:por#show ip int brief
Mon Jan 29 18:37:01.596 UTC

Interface          IP-Address      Status      Protocol Vrf-Name
Loopback0          198.19.1.26    Up         Up        default
Loopback1          11.11.4.1     Up         Up        default
MgmtEth0/0/CPU0/0  198.18.1.33    Up         Up        default
GigabitEthernet0/0/0/0 10.0.0.53    Up         Up        default
GigabitEthernet0/0/0/1 53.0.0.26    Up         Up        default
GigabitEthernet0/0/0/2 54.0.0.26    Up         Up        default

```

```

RP/0/0/CPU0:min#traceroute 11.11.4.1
Mon Jan 29 18:51:46.686 UTC

Type escape sequence to abort.
Tracing the route to 11.11.4.1

1 51.0.0.28 [MPLS: Label 16026 Exp 0] 19 msec 19 msec 19 msec
2 53.0.0.26 9 msec * 9 msec

```

Figure 3.113: SR Path between min and por Nodes.

Basically, Segment Routing Traffic Engineering (SR-TE) has no additional overhead except for a higher number of labels traversing over the network. There is no control-plane for tunnels created this way as TE LSP is encapsulated into a packet with an MPLS label.

3.6. Failover

Reliability of service delivery in today's world is the most important aspect of the evaluation of the quality of computer networks. The work of people in all industries to a greater or lesser extent is based on complex information systems and network infrastructure. Even momentary downtime caused by link failure can generate large losses for the company. To avoid these, engineers design the network with redundant links for spare connectivity while we talk about the business crucial applications. The remaining part of this chapter contains an overview of the solutions which can prevent downtime on our network against link failure.

3.6.1. MPLS

In *Figure 3.91* links between the end-users (VMs) are marked green for a longer route

and red for a shorter route. By default, communication takes place using the main connection depending on the traffic type and in case of failure, it will route packets via the active link no matter for the PBR.

This test case will allow checking whether the created configuration ensures continuous operation in accordance with the assumptions by simulating the main connection failure by shutting down ethernet interface on CSR1000V1 which connects to the Dublin router as well as it being the tail to both MPLS-TE tunnels.

When we executed *traceroute* from VM1 to VM2 we were able to see that CSR1000V1 makes the routing decision based on the shorter path (red), as seen in *Figure 3.114*.

```
openflow1@openflow1:~/Downloads$ traceroute 192.16.20.20
traceroute to 192.16.20.20 (192.16.20.20), 30 hops max, 60 byte packets
 1  192.16.10.2 (192.16.10.2)  0.678 ms  0.694 ms  0.687 ms
 2  100.8.2.3 (100.8.2.3)  1.207 ms  1.204 ms  1.196 ms
 3  200.0.4.2 (200.0.4.2)  3.434 ms  4.070 ms  4.067 ms
 4  100.8.6.4 (100.8.6.4)  4.960 ms  4.055 ms  4.051 ms
 5  192.16.20.20 (192.16.20.20)  4.838 ms  4.836 ms  4.830 ms
openflow1@openflow1:~/Downloads$
```

```
openflow1@openflow1:~$
64 bytes from 192.16.20.20: icmp_seq=609 ttl=59 time=1.26 ms
64 bytes from 192.16.20.20: icmp_seq=610 ttl=59 time=1.21 ms
64 bytes from 192.16.20.20: icmp_seq=611 ttl=59 time=3.39 ms
64 bytes from 192.16.20.20: icmp_seq=612 ttl=59 time=1.03 ms
64 bytes from 192.16.20.20: icmp_seq=613 ttl=59 time=0.979 ms
64 bytes from 192.16.20.20: icmp_seq=614 ttl=59 time=1.03 ms
64 bytes from 192.16.20.20: icmp_seq=615 ttl=59 time=8.09 ms
64 bytes from 192.16.20.20: icmp_seq=616 ttl=59 time=1.19 ms
64 bytes from 192.16.20.20: icmp_seq=617 ttl=59 time=3.80 ms
64 bytes from 192.16.20.20: icmp_seq=618 ttl=59 time=1.50 ms
64 bytes from 192.16.20.20: icmp_seq=619 ttl=59 time=1.63 ms
64 bytes from 192.16.20.20: icmp_seq=620 ttl=59 time=1.61 ms
64 bytes from 192.16.20.20: icmp_seq=621 ttl=59 time=1.81 ms
64 bytes from 192.16.20.20: icmp_seq=622 ttl=59 time=2.72 ms
64 bytes from 192.16.20.20: icmp_seq=623 ttl=59 time=3.75 ms
64 bytes from 192.16.20.20: icmp_seq=624 ttl=59 time=0.979 ms
64 bytes from 192.16.20.20: icmp_seq=625 ttl=59 time=0.980 ms
64 bytes from 192.16.20.20: icmp_seq=626 ttl=59 time=1.29 ms
64 bytes from 192.16.20.20: icmp_seq=627 ttl=59 time=1.61 ms
^[[21;6-64 bytes from 192.16.20.20: icmp_seq=628 ttl=59 time=1.06 ms
64 bytes from 192.16.20.20: icmp_seq=629 ttl=59 time=1.22 ms
64 bytes from 192.16.20.20: icmp_seq=630 ttl=59 time=1.58 ms
64 bytes from 192.16.20.20: icmp_seq=631 ttl=59 time=3.16 ms
```

Figure 3.114: Traceroute between VM1 and VM2 via Red Path.

Next, we have shutdown GigabitEthernet4 on CSR1000V1 to simulate the failure of the Tunnel0. This allowed us to see that the ICMP requests to VM2 were continuous and that *traceroute* returned the IP addresses of interfaces along the longer path (green), as seen in *Figure 3.115*.

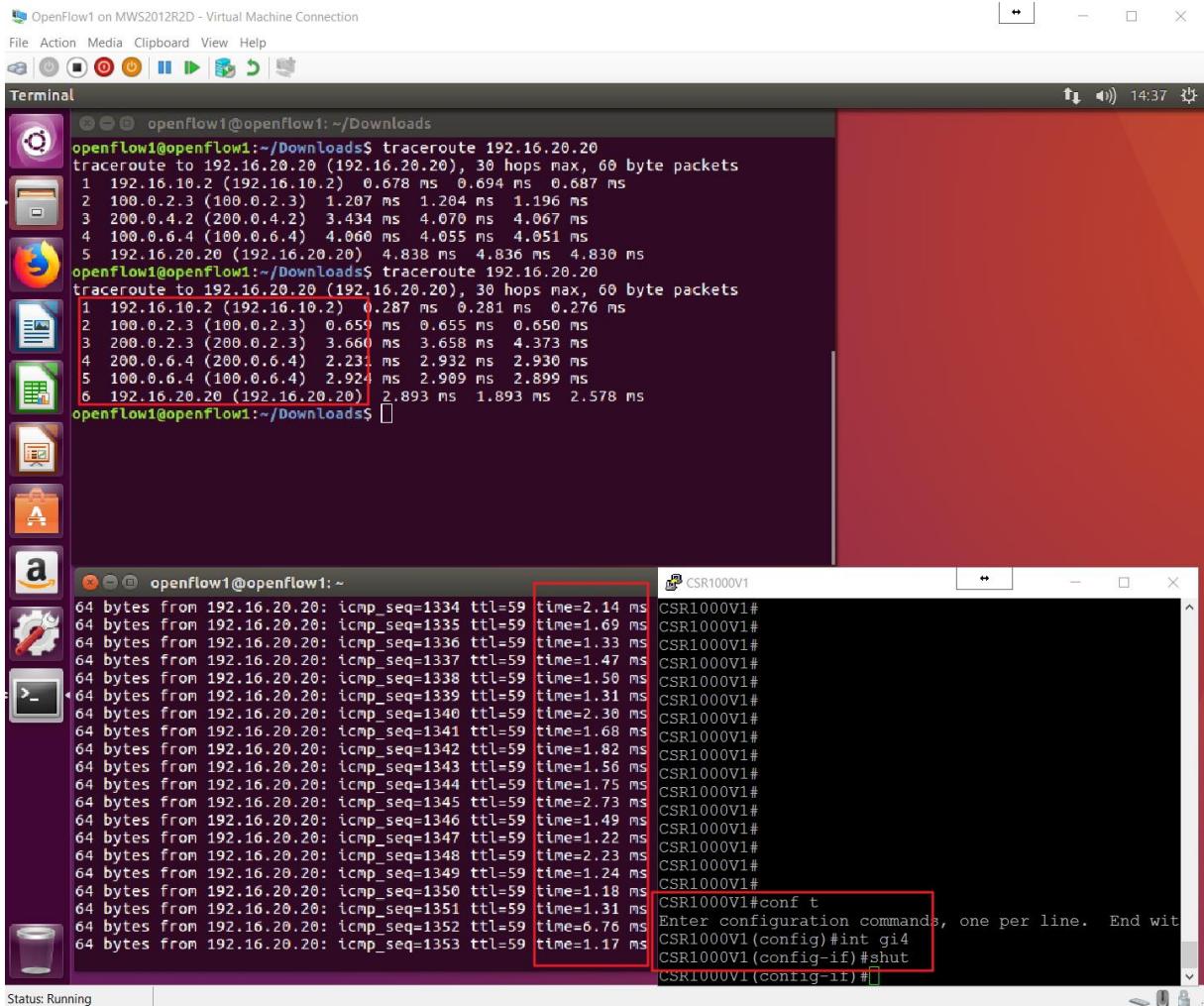


Figure 3.115: Traceroute between VM1 and VM2 via Green Path.

To test it even further we have downloaded a sample 1 GB file from FTP server on VM2.

This time we noticed that PBR hasn't been used as FTP data normally would traverse across the shorter path, but in this situation, it was transferred across via backup link on Tunnel1 and no packets were matched against policy routing, as seen in *Figure 3.116*.

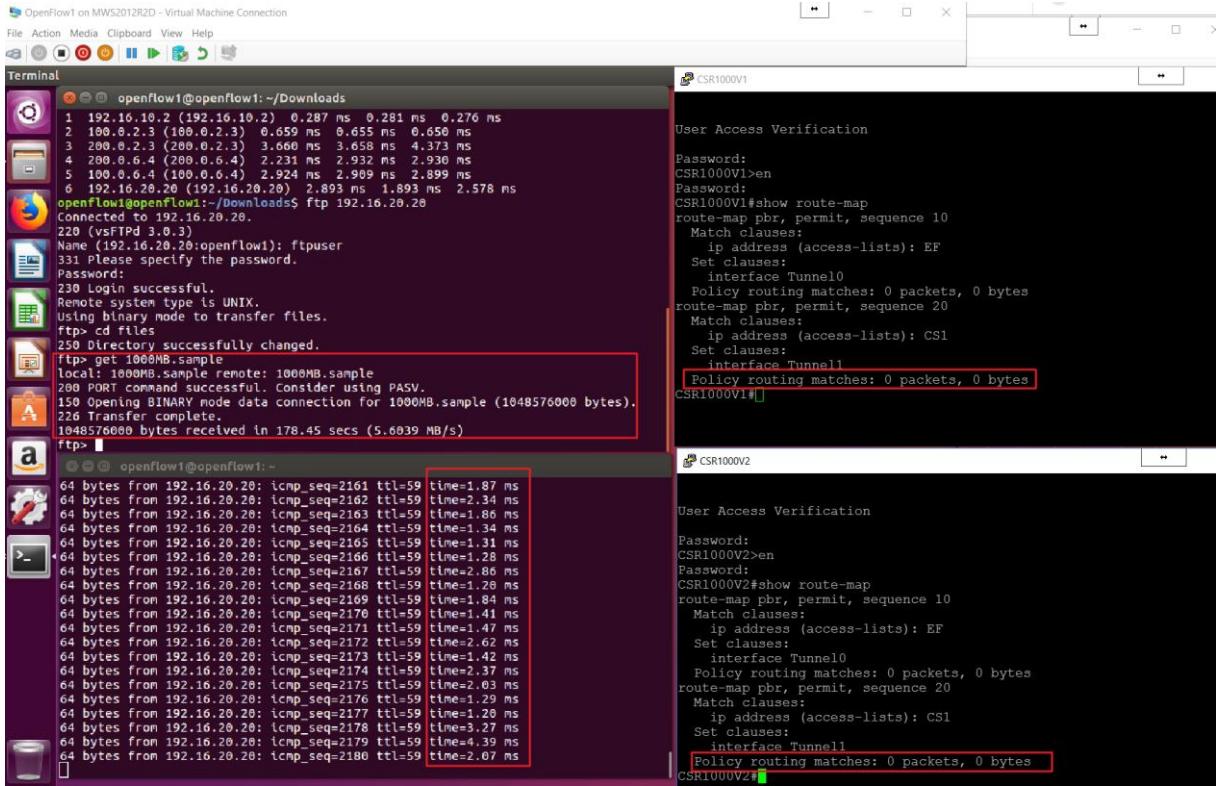


Figure 3.116: FTP Data Transfer from VM2 via Green Path.

3.6.2. OpenFlow

However, in terms of SDN we have used the datacentre Topo discussed in *Chapter 2.6.6* with OF13, but with the HPE Aruba VAN controller instead of Ryu to show the STP calculated path, as seen in *Figure 3.117*.

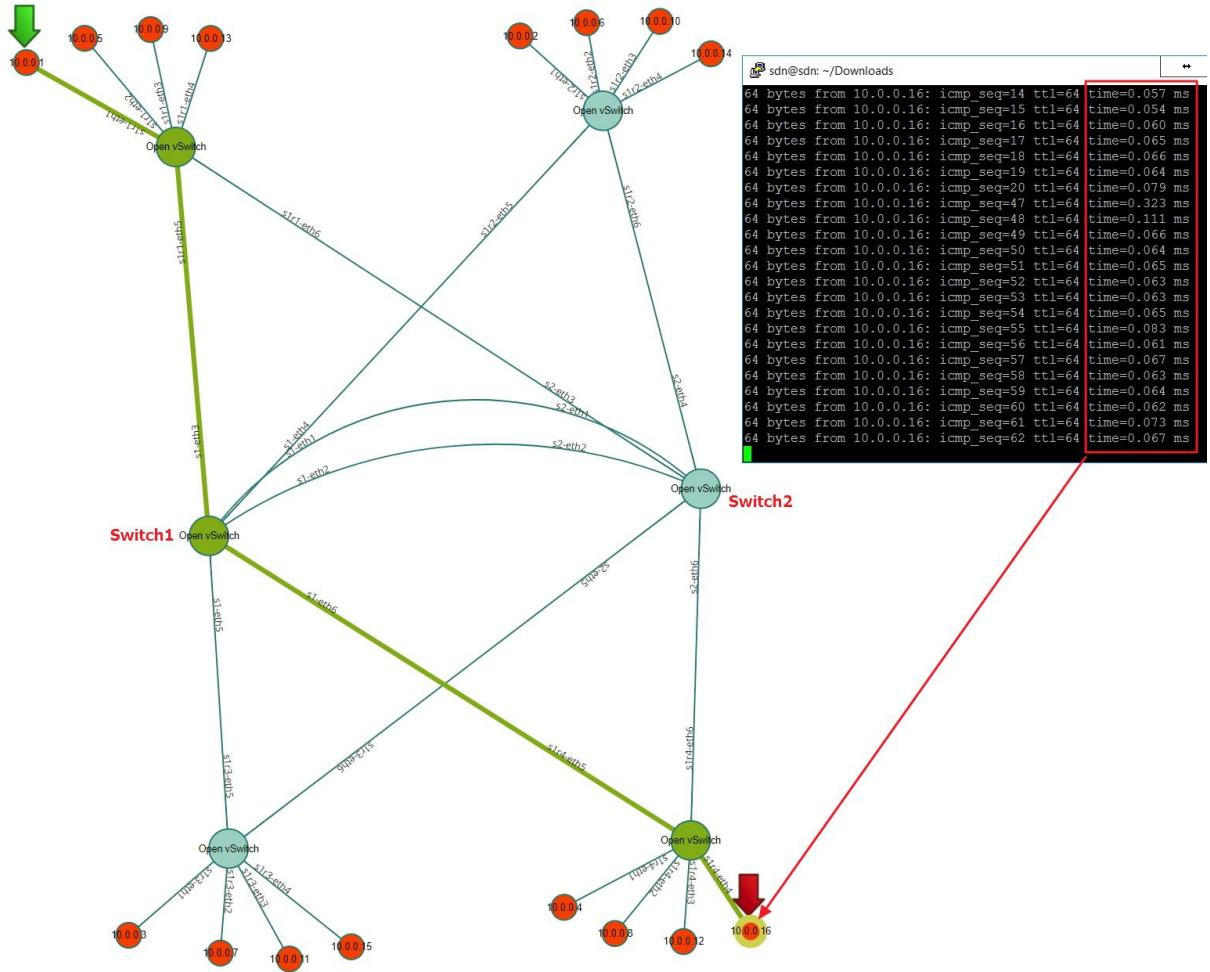


Figure 3.117: Datacentre Topo,4,4 in Mininet with HPE Aruba VAN Controller.

Next, we have shutdown s1-eth3 vNIC connected to Switch1 which resulted in a new path calculation which was used to forward packets via Switch2 instead, as seen in *Figure 3.118*.

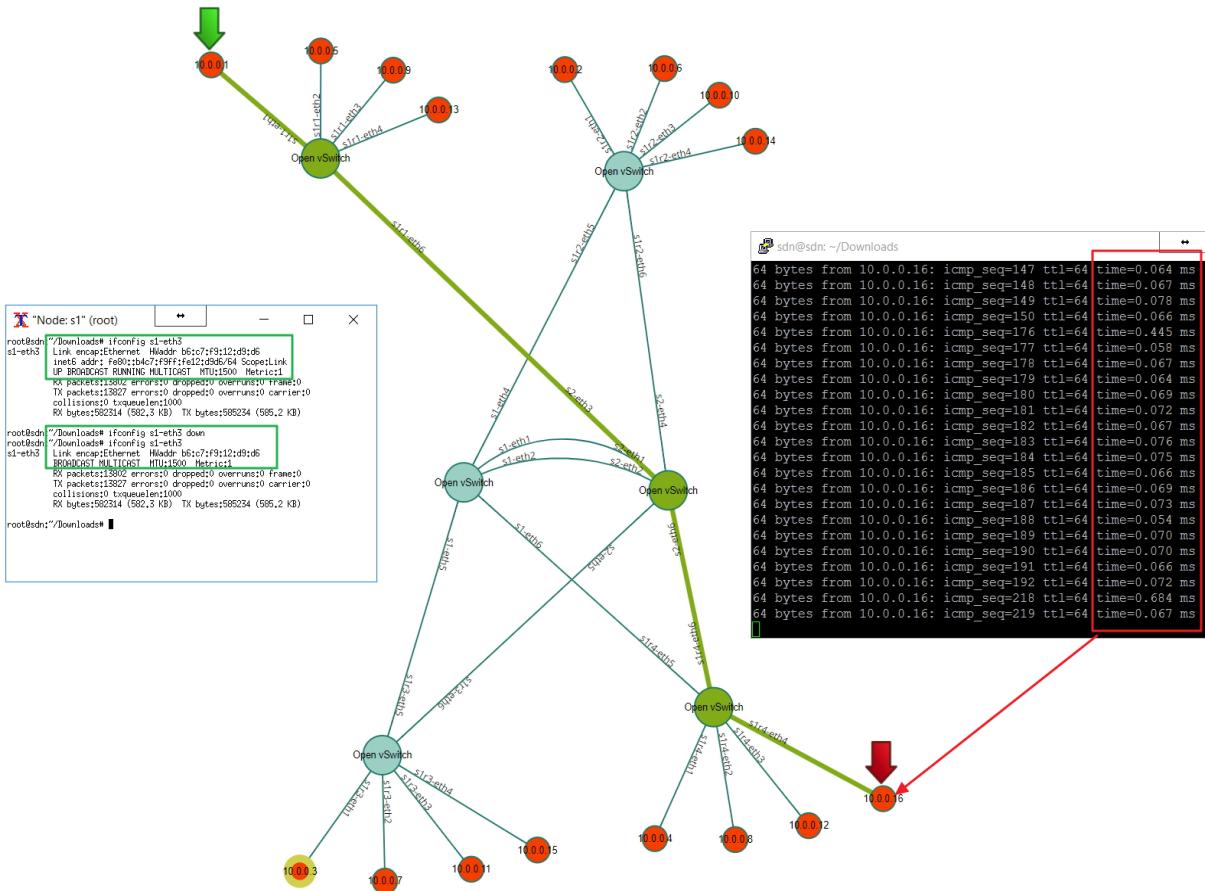


Figure 3.118: HPE Aruba VAN Controller and New Flow Path after vNIC Failure.

ICMP requests were successful and no major delay was identified during the failure of the link while the interface went down similar to the scenario when we tested failover with MPLS on Cisco routers and Linux FRR nodes.

We have tested that in the above discussed situations, the failover mechanism operates correctly. The reaction time in the solution based on the OF depends fully on the controllers' capabilities to learn the DPID or the amount of manually entered flow entries in the flow tables as well as their priorities. In the case of Cisco devices checking for the connection, the status is entirely the responsibility of the IOS system. It is exceptionally efficient while changing the packet forwarding route when the main connection is restored to the state before the failure. However, we have noticed that it takes much longer to diagnose that the transmission channel is not working properly. Additionally, in this experiment, we have used an open and commercial implementation of the MPLS protocol which showed us how the network copes with such configuration during link failure. This basically demonstrates how to use MPLS and SDN with failover mechanisms to detect the connection state and effectively configure the backup route in case of unavailability.

3.7. Summary of Tests

Chapter 3 was the most practical part of this research as it covered the configurations and results from a multitude of test cases which were aimed at investigating the hypothesis made at the beginning of it.

The experiments uncovered that Linux nodes implemented with MPLS acting as LSR cannot pop an outgoing label on the LSP, their throughput was the lowest between LERs, packet loss was high for small files as well as a delay for large packets. Linux and FRR with the mixed approach of using Cisco nodes resulted in lower response times in comparison to pure hardware, and it was also fully compatible while creating QoS policies when acting as LERs and during TE tests.

The deployed Cisco HW without MPLS in Linux nodes were obviously fully compatible between each other while exchanging label information, throughput and delay were lower than in OF, but the number of packets lost was lower, while the response times were still a lot higher for small packets than with OF. Like interoperability, QoS and TE were easily achievable after long and complex configuration of nodes which requires wide knowledge from the network administrator.

OpenFlow, however, resulted in lower throughputs even than a P2P link with slightly higher delays, but it outperformed all remaining tested technologies. It did perform worse when tested with large volumes of data during packet loss, but it achieved the smallest response times for small and large packet sizes. Scalable topology has proven that it's possible to scale-up network resources with minimal configuration, while QoS experiments with the Ryu controller provided an insight into per-flow policies, mapping of QoS classes to DSCP values and traffic remarking with the Meter Table. TE in OF tested with Floodlight and HPE Aruba VAN controllers on scaled-up topology have proven that SDN caters for centralized management to program the flow of data while it also has a mechanism for link failover which will respond rapidly after detecting that DPID is no longer available.

4. Summary

The last chapter of the thesis contains a summary of the work carried out to present the lessons learned in relation to the assumptions and hypothesis formulated in the initial part of the practical work in *Chapter 3*. It will also show the considerations in relation to issues that have arisen during the experiments and how we can fix them to remain within the subject matter.

4.1. Research Work

Introduction to main research concepts and theoretical review of MPLS, OpenFlow, and SDN was described in *Chapter 1*.

The experiments carried out were divided into six main stages which in turn are described in *Chapter 3.1* to *Chapter 3.6*. Before we were able to investigate the subject matter and study the efficiency of MPLS and OpenFlow we had to prepare the environment in which the performance tests were conducted as described in *Chapter 2*. This stage turned out to be as labour-intensive as the main part of the work. The reason for this was the need to configure multiple topologies operating within three different technologies, additionally with the use of a mixture of virtual environments and the physical equipment which sometimes refused to cooperate. The issues discussed were a completely new topic for me and it certainly had an impact on the time devoted to bringing the infrastructure available to the state that enabled the planned tests to be carried out.

It is worth mentioning that the whole process has been iterative. After a series of test scenarios and preliminary analysis of the results obtained, the test case modification was often made so that the final results were as valuable and reliable as possible. In *Chapter 2.7* we also had to decide on the methodology and solutions which were used to perform the experiments.

The comparison of interoperability, performance, and scalability of the described technologies consisted of planning and carrying out a series of tests and collecting structured results for analysis through *Chapter 3.1* to *Chapter 3.3*. Prepared infrastructures were used to verify the arguments formulated at the beginning to check whether the use of MPLS and OpenFlow helps in the effective use of network resources.

Additionally, the tested technologies were also implemented with a view to providing an overview of possibilities in the field of QoS and TE. Examples of their use in this aspect have been demonstrated in *Chapter 3.4* and *Chapter 3.5*.

The application of failover mechanism was reviewed in *Chapter 3.6* where we have tested both technologies and the reaction to link failure.

4.2. Conclusions

The aim of this thesis was to verify the voracity of the statement that using OpenFlow and MPLS protocols, we can manage traffic in a computer network more efficiently than it does in the situation of using the currently most widely used solution which is the IP protocol. Partial conclusions resulting from individual experiments were usually presented on a regular basis in the parts devoted to the description of the results obtained. In this chapter, we will present a summary of conclusions resulting from the work carried out.

The first important statement arising from this work is the compatibility of two different implementations of the MPLS protocol. We have checked that there is a possibility to provide internet services using a heterogeneous network using both routers based on MPLS in Linux and Cisco hardware routers. However, during tests with Linux router acting as LSR Kildare node couldn't pop out outgoing label to forward the packets on the LSP. This has proven that MPLS support in Linux wasn't fully compatible with software routers, where no issues were identified when same node acted as LER. Cisco devices had no issues during testing when they acted as LER or LSR and thus makes them fully interoperable with MPLS.

In relation to the main aspect of the work, the efficiency of OpenFlow and MPLS technologies, the obtained results are not satisfactory. Several factors contributed to this. The most serious problem turned out to be the compatibility of the used software traffic generators. Both with the use of the MGEN and different versions of iPerf we couldn't simulate the situation in which the used network infrastructure would operate within the limits of its capabilities. As it was already mentioned before, most likely the best selective tests were when we attempted to transfer very small packets that would almost saturate the total with a nominal bandwidth in *Chapter 3.2.2.3*. A hardware traffic generator such as Open Source Network Tester (OSNT) developed by Antichi (2017) could come here to help, but we did not have the opportunity to use such a tool during the experiments. The second problem, which is closely related to the first one, was the inability to show performance imperfections in the architecture based on the IP protocol. The assumption was to demonstrate such a situation and then to show what new opportunities are provided by OpenFlow and MPLS. Unfortunately, we were unable to simulate a situation in which the IP protocol and routing would be clearly worse. The third, but not so important problem was the size of the prepared test environments. Several nodes are not enough to fully show the advantages of using the technologies described. All of these contradictions mean that drawing unambiguous conclusions about the efficiency of an OpenFlow-based network or MPLS is very difficult. The conducted tests comparing the transmission parameters (throughput, packet loss, delay or variability of delay) depending on

the technology used, did not provide satisfactory results. They only show that under given conditions, virtually OF works faster than MPLS. OF scored the best results in throughput, delay and RTT, but it was outperformed during packet loss as its performance was reduced due to high frequency of packet-in messages send to the controller. This however can be improved by use of proactive approach where controller wouldn't participate in the forwarding as soon as flow entry is written to the Flow Table. A very interesting observation was identified in the OpenFlow provided in one of the tests described in *Chapter 3.2.2.1*. The results presented here show that the OF flow table operations are much faster than viewing the routing table when deciding to send the packet to the next node on the route. The total delay using MPLS in Linux and Cisco is, however, lower than in the case of IP forwarding because the time gained during the transition through the LSR is lost at the LER nodes. The operation of adding and removing the MPLS label takes longer than selecting the route based on the routing table. Having a test environment consisting of a larger number of nodes could be better highlighted in the MPLS protocol. However, due to the limitations of the equipment available in the laboratory, it was impossible.

OF also appeared to far easy to scale than MPLS as adding additional nodes only involved in altering the script when controller takes over the flow processing, while this process for MPLS requires all the configurations on each node individually. In terms of compatibility of scaled-up infrastructure in *Chapter 3.3*, LDP on both Cisco and FRR Linux nodes were functioning correctly, but Linux implementation resulted in lower delays, while OF was irrespectively the fastest.

Moving away from the aspect of the throughput of OpenFlow and MPLS in the work we presented examples of the use of these protocols in the field of QoS and TE. The first issue was the transmission of data on arbitrarily selected routes. The assumption was that the flow paths leading to one target point would depend on the source generating the traffic. The experiments described in detail in *Chapter 3.4* and *Chapter 3.5* showed that with each of the technologies studied we can obtain the expected results. It was possible to use traffic classification and DSCP markings for both technologies to provide QoS, but only OF has a mechanism which can be used to limit the bandwidth with use of Linux HTBs and port numbers to move packets into different queues. Major identified TE benefits in OF come from centralization of management which minimises all that administrator's burden while setting up the tunnel end-points, while in OF simply flow paths are hardcoded on the controller with flow entries to specific ports on each switch as it was discussed in *Chapter 3.5.2*.

The last topic of the paper was the question of securing a computer network against

the effects of a sudden connection failure. In *Chapter 3.6* we can see what possibilities in this respect are given to us with MPLS on Linux and Cisco as well as on the OF protocol itself. Both solutions are effective, but it seems to be better when using OF than Cisco or Linux nodes because the failure detection takes place there at the centralized external SDN controller. The administrator is required only to properly configure the backup flow entries or to simply use the learning capabilities of the developed controller and the apps running on it. This will allow for automatically redirected traffic in an emergency. Using hardware routers, failure detection takes place there at the OS level where with a software app we must additionally use tools that supervise the connection state such as a SDN controller using apps running in the background which respond appropriately to the existing situation. However, the Cisco solution is more natural and should use fewer resources.

4.3. Future Work

Considering the results obtained regarding the performance of the OpenFlow and MPLS protocols against the IP protocol, they leave a certain amount of insufficiency. The reason for this situation was explained above and this thesis could be a good starting point for continuing the research on the presented issue. To obtain reliable and more useful results would undoubtedly contribute to the use of the already mentioned OSNT. Large research centres dealing with the area of computer networks certainly have adequate and accurate measuring equipment which is necessary for this type of project. Also, creating a larger test environment should not be a problem for them. Presented in my work configurations, tests of network performance and scalability parameters, QoS and TE possibilities of OpenFlow and MPLS protocols as well as results obtained describing the speed of operation of these technologies can be a good starting point for research covering these issues on a much larger scale.

So far, software providers and users of the Internet of Things (IoT) systems have not paid attention to the issues of network communication which takes place between endpoints and central databases or data processing applications. In simple IoT applications, communication is rare where data transfer happens only from time to time and it involves the transmission of small amounts of data. In this case, the network traffic parameters may be low due to the large delay of transmitted data packets and the completely missed requirements in relation to the encryption of transmitted data or the guarantee of transmission speed. However, because of new, emerging technologies, IoT systems and their higher level of complexity require communication changes. Modern services based on data obtained in IoT systems require efficient computer networks that meet specific QoS requirements such as very short delays in data transmission. A new approach to the creation and management of a network infrastructure

with the use of SDN and OF can face this challenge.

It is possible to create ecosystems with SDN network infrastructure and cloud computing which the main purpose would be to automatically control the transmission of data obtained in IoT systems to meet the requirements of end-users. NoviFlow has demonstrated a particularly interesting ecosystem because it integrates various technologies in a coherent and efficient manner including LXC containers, NoviSwitch network devices, Spirent network traffic generator, Libelium IoT system, SDN controller called Ryu and ThingSpeak visualization application (LeClerc, 2016). Similar extensions of the OpenFlow protocol related to the inspection of transmitted data packets and the way they are modified on the switch could be used to recognize and classify network traffic to modify it for subsequent management.

For example, the IoT application could be used to store, visualize and analyse data from several sets of sensors to measure: noise level, temperature, air humidity and lighting levels where some form of generator could be used to emulate the traffic coming from the IoT ecosystems on the macro scale with various sets of sensors sending random data. This can bring closer the advantages of integrating SDN computer networks and IoT systems.

List of Abbreviations

ACL	Access Control List
AF	Assured Forwarding
API	Application Programming Interface
API	Application Programming Interface
APIC-EN	Application Policy Infrastructure Controller Enterprise Module
App	Application
AS	Autonomous System
ASICs	Application Specific Integrated Circuits
B	Byte
BDDP	Broadcast Domain Discovery Protocol
BE	Best Effort
BFD	Bidirectional Forwarding Detection
BGP	Border Gateway Protocol
BGP-LS	Border Gateway Protocol Link-State
BPDUs	Bridge Protocol Data Units
Bps	Bit per second
Capex	Capital Expenditure
CBWFQ	Class-Based Weighted Fair Queuing
CE	Customer Edge
CF	Compact Flash
CLI	Command Line Interface
CORD	Central Office Re-architected as a Datacentre
CoS	Class of Service
CPU	Central Processing Unit
CRUD	Create, Retrieve, Update, Delete
CSR	Cloud Services Router
CSV	Comma Separated Values
DAST	Distributed Applications Support Team

DDoS	Distributed Denial of Service
DiffServ	Differentiated Services
DoS	Denial of Service
DPCtl	Data Path Controller
DPID	Datapath ID
DRAM	Dynamic Random-Access Memory
DSCP	Differentiated Services Codepoint
EIGRP	Enhanced Interior Gateway Routing Protocol
EOS	Extensible Operating System
EXP	Experimental
FBOSS	Facebook Open Switching System
FEC	Forwarding Equivalence Class
FFR	FFRouting
FTP	File Transfer Protocol
FWA	Fixed Wireless Access
GB	Gigabyte
Gbps	Gigabit per second
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
GW	Gateway
HTTP	Hypertext Transfer Protocol
HPE	Hewlett Packard Enterprise
HTB	Hierarchical Token Bucket
HW	Hardware
HWIC-1FE	1-Port Fast Ethernet layer 3 card
HWIC-2FE	2-Port Fast Ethernet layer 3 card
IAB	Internet Architecture Board
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol

IOS	Internetwork Operating System
IoT	Internet of Things
IP	Internet Protocol
IS-IS	Intermediate System to Intermediate System
ISP	Service Provider
IT	Information Technology
KB	Kilobyte
kB/s	Kilobyte per second
Kbps	Kilobit per second
LAN	Local Area Network
LDP	Label Distribution Protocol
LDPd	Label Distribution Protocol daemon
LER	Label Edge Router
LIB	Label Information Base
LLDP	Link Layer Discovery Protocol
Lo	Loopback
LSP	Label Switching Path
LSR	Label Switched Router
MAC	Media Access Control
MB	Megabyte
MB/s	Megabyte per second
Mbps	Megabit per second
MD-SAL	Model-Driven Service Adoption Layer
MGEN	Multi-Generator
MIB	Management Information Base
MITM	Man in the Middle
MP-BGP	Multiprotocol Border Gateway Protocol
MPLS	Multiprotocol Label Switching
MPLS-TP	Multiprotocol Label Switching Transport Profile
ms	Millisecond

NA	Not Applicable
NAT	Network Address Translation
NBAR	Network-Based Application Recognition
NBI	Northbound Interface
NETCONF	Network Configuration Protocol
NFV	Network Functions Virtualization
NGN	Next Generation Network
NIC	Network Interface Card
NLANR	National Laboratory for Applied Network Research
NOS	Network Operating System
NRL	Naval Research Laboratory
NSA	National Security Agency
ODL	OpenDaylight
OF	OpenFlow
OF-Config	OpenFlow Configuration
ONF	Open Networking Foundation
ONL	Open Network Linux
ONOS	Open Network Operating System
Opex	Operating Expenditure
OF10	OpenFlow version 1.0
OF13	OpenFlow version 1.3
OF14	OpenFlow version 1.4
OF15	OpenFlow version 1.5
OFM	OpenFlow Manager
OS	Operating System
OSI	Open Systems Interconnection
OSNT	Open Source Network Tester
OSPF	Open Shortest Path First
OSPF	Open Shortest Path First
OVS	Open Virtual Switch

OVSDB	Open Virtual Switch Database Management Protocol
P	Provider
P2P	Point to Point
PBR	Policy Based Routing
PC	Personal Computer
PCEP	Path Computation Element Protocol
PE	Provider Edge
PPA	Personal Package Archive
QoS	Quality of Service
RAM	Random Access Memory
RBAC	Role-Based Access Control
RD	Route Distinguisher
REST	Representational State Transfer
RfC	Request for Comment
RPFC	Reverse Path Forwarding Check
RSVP	Resource Reservation Protocol
RT	Route Target
RTT	Round-Trip Time
S	Subscriber
SBI	Southbound Interface
St	Stack
STP	Spanning Tree Protocol
SCMS	Source Code Management System
SDN	Software Defined Networking
SD-WAN	Software Defined Wide Area Network
SNMP	Simple Network Management Protocol
SONET	Synchronous Optical Networking
SP	Service Provider
SR	Stable Release
SR-TE	Segment Routing Traffic Engineering

SSH	Secure Shell
SSL	Secure Socket Layer
StDev	Standard Deviation
SW	Software
TAR	Tape Archive
TCP	Transmission Control Protocol
TE	Traffic Engineering
TLS	Transport Layer Security
Topo	Topology
ToS	Type of Service
TPT	Third-Party Provider
TTL	Time to Live
UDP	User Datagram Protocol
UKUU	Ubuntu Kernel Update Utility
USB	Universal Serial Bus
VA	Virtual Appliance
VAN	Virtual Application Network
VIRL	Virtual Internet Routing Lab
vCPU	Virtual Central Processing Unit
VEth	Virtual Ethernet
vHost	Virtual Host
VM	Virtual Machine
VMM	Virtual Machine Monitor
vNIC	Virtual Network Interface Card
VoIP	Voice over Internet Protocol
VOSE	Virtual Operating System Environment
VPN	Virtual Private Network
vPort	Virtual Port
VRF	Virtual Routing and Forwarding
vSwitch	Virtual Switch

WAN	Wide Area Network
WAWIT	Warsaw University of Technology
X64	64-bit Architecture
X86	32-bit Architecture
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

List of Figures

- Figure 1.1: OSI Model and MPLS.
- Figure 1.2: MPLS Label Format.
- Figure 1.3: MPLS Label Switching.
- Figure 1.4: VRF and VPN.
- Figure 1.5: OpenFlow Components.
- Figure 1.6: OpenFlow Flow Tables.
- Figure 1.7: OpenFlow Controller to Switch Secure Channel Communication.
- Figure 1.8: Components of OpenFlow Switch.
- Figure 1.9: Single Topo with HPE Aruba VAN Controller.
- Figure 1.10: Table-miss Flow Entry with HPE Aruba VAN Controller.
- Figure 1.11: New Flow Entry with HPE Aruba VAN Controller.
- Figure 1.12: Fields to Match against Flow Entries.
- Figure 1.13: Reserved Ports with HPE Aruba VAN Controller.
- Figure 1.14: Hybrid Switch Secure Mode with Running HPE Aruba VAN Controller.
- Figure 1.15: Hybrid Switch Secure Mode with Stopped HPE Aruba VAN Controller.
- Figure 1.16: Pure OF Switch Secure Mode with Stopped HPE Aruba VAN Controller.
- Figure 1.17: OVS and Standalone Mode with ODL Controller.
- Figure 1.18: OVS Standalone Mode and Switch Flow Entries with Stopped ODL Controller.
- Figure 1.19: OpenFlow Controller to Switch Session.
- Figure 1.20: Negotiation of OpenFlow Version.
- Figure 1.21: OF Version Negotiation Failure.
- Figure 1.22: Features Reply Message from the Switch.
- Figure 1.23: Multipart Reply Message from the Switch.
- Figure 1.24: Multipart Port Description Reply Message from the Switch.
- Figure 1.25: Packet-In Message from Switch to the ODL.
- Figure 1.26: Packet-Out Message from ODL to the Switch.
- Figure 1.27: SDN Architecture.

- Figure 1.28: VMware NSX.
- Figure 1.29: ODL Controller Architecture.
- Figure 1.30: Cisco APIC-EN Controller Architecture.
- Figure 1.31: SDN Northbound and Southbound Interfaces.
- Figure 1.32: Intelligent Offloading of Traffic.
- Figure 1.33: SDN-Toolkit Attempt to Download Flows from Floodlight Controller.
- Figure 1.34: SDN-Toolkit Attempt to Download Flows from ODL Controller.
- Figure 1.35: Authentication Token with HPE Aruba VAN Controller.
- Figure 1.36: Failed Authentication with HPE Aruba VAN Controller.
- Figure 1.37: Successful Authentication with HPE Aruba VAN Controller.
- Figure 1.38: DPID Information Retrieval via API with HPE Aruba VAN Controller.
- Figure 1.39: Traditional Forwarding.
- Figure 1.40: OpenFlow Forwarding.
- Figure 1.41: Reactive Flow Entry.
- Figure 1.42: Proactive Flow Entry.
- Figure 1.43: Proactive Flow Path.
- Figure 2.1: Cisco 2800 Series Routers.
- Figure 2.2: Cisco 2801 Router IOS Version.
- Figure 2.3: Port Mapping between Hyper-V NICs and Mininet VM.
- Figure 2.4: Communication between Host2 and Hyper-V Host.
- Figure 2.5: View of Devices in External Network via GUI with HPE Controller.
- Figure 2.6: Linear Topo with ODL Controller.
- Figure 2.7: Flow Table with ODL Controller.
- Figure 2.8: Port Entries with ODL Controller of Switch1.
- Figure 2.9: Flow Entries with ODL Controller of Switch1.
- Figure 2.10: OFM Flow Entry to Block Traffic on Switch1.
- Figure 2.11: Single Topo with Floodlight Controller.
- Figure 2.12: Flow Table with Floodlight Controller.
- Figure 2.13: Flow Entries with Floodlight Controller of Switch1.

- Figure 2.14: Tree Topo with HPE Controller.
- Figure 2.15: Port Entries with HPE Controller of Switch1.
- Figure 2.16: Flow Entries with HPE Controller of Switch1.
- Figure 2.17: Linear Topo with ONOS Controller.
- Figure 2.18: Port Entries with ONOS Controller of Switch2.
- Figure 2.19: Flow Entries with HPE Controller of Switch2.
- Figure 2.20: Flow Entries with POX Controller of Switch1.
- Figure 2.21: Example Torus Topo View from ONOS Controller.
- Figure 2.22: STP POX Controller Command for Torus Topo.
- Figure 2.23: ICMP Packets between Two Hosts with POX Controller for Torus Topo.
- Figure 2.24: Linear Topo with Ryu Controller.
- Figure 2.25: Flow Entries with Ryu Controller of Switch1 with different OF Versions.
- Figure 2.26: OF15 and OVS Packet-Out Message issue with Ryu Controller.
- Figure 2.27: Datacentre Topo,4,4 in Mininet with Ryu Controller and in OFM.
- Figure 2.28: DPIDs and BPDUs Exchange between Ryu Controller and Switches.
- Figure 2.29: Switch1 Flow Entries and Ports with Datacentre Topo and Ryu Controller.
- Figure 2.30: Kernel Upgrade in UKUU.
- Figure 3.1: MPLS Compatibility Topology.
- Figure 3.2: Virtual Interface Mapping to Kildare and Laois MPLS VMs.
- Figure 3.3: Static Routes on OpenFlow VMs to Kildare and Laois MPLS VMs.
- Figure 3.4: Dublin Cisco MPLS as LER.
- Figure 3.5: VM1 ICMP Request to VM2.
- Figure 3.6: VM2 ICMP Request to VM1.
- Figure 3.7: Cisco Router Outgoing and Incoming MPLS Labels as LSR.
- Figure 3.8: Cisco Router Outgoing and Incoming MPLS Labels as LER.
- Figure 3.9: Kildare Router ICMP Reply to VM1.
- Figure 3.10: Kildare Router ICMP Reply to VM2.
- Figure 3.11: LSP from VM2 to VM1's GW between both LERs.

- Figure 3.12: LSP from VM1 to VM2's GW between both LERs.
- Figure 3.13: OpenFlow Performance Topology.
- Figure 3.14: OVS Bridge and Default Local Controller in Mininet.
- Figure 3.15: DPCTL Switch Commands.
- Figure 3.16: P2P Connection on Internal vSwitch.
- Figure 3.17: VM2 to VM1 P2P Throughput Test.
- Figure 3.18: Network Throughput Depending on the IP Technology.
- Figure 3.19: GUI of jPerf2 for Initial P2P Test.
- Figure 3.20: Syntax of UDP MPLS in Linux Tests with iPerf3.
- Figure 3.21: Network Delay Depending on the IP Technology.
- Figure 3.22: Packets Received in Comparison to Total Packets Transmitted.
- Figure 3.23: Correctly Delivered Data Depending on the IP Technology Used for Individual Test Cases.
- Figure 3.24: RTT Results in Milliseconds.
- Figure 3.25: RTT Comparison in Percentage Values.
- Figure 3.26: Three Cisco MPLS LSR Nodes and Two LER Nodes.
- Figure 3.27: Three Cisco MPLS LSR Nodes and Two MPLS in Linux LER Nodes.
- Figure 3.28: Mininet OpenFlow Scaled-Up Topology.
- Figure 3.29: MPLS Forwarding Tables and LDP Binding Information on Cisco Nodes.
- Figure 3.30: MPLS Forwarding Tables and LDP Binding Information in Linux Environment.
- Figure 3.31: RTT Results for Scaled-Up Environment.
- Figure 3.32: RTT Comparison in Percentage Values for Scaled-Up Environment.
- Figure 3.33: Two Cisco PE MP-BGP Nodes and Two CE EIGRP Nodes.
- Figure 3.34: MPLS Domain Label Bindings.
- Figure 3.35: BGP VPN Label Bindings.
- Figure 3.36: Mean Values of Throughput Test of FTP and HTTP Servers.
- Figure 3.37: Mappings between DSCP and MPLS EXP Bits
- Figure 3.38: MPLS-TE Tunnels with RSVP Topology.
- Figure 3.39: MPLS-TE Tunnels Signalling.

- Figure 3.40: Linux HTB Queues for FTP and Web Server Scenario.
- Figure 3.41: Connecting Controller to OVSDB.
- Figure 3.42: List of QoS and Queue Tables and S1 Flow Entries.
- Figure 3.43: HTB Queue Settings.
- Figure 3.44: QoS Settings and S1 Flow Entries.
- Figure 3.45: Topo for per Class with DSCP QoS.
- Figure 3.46: Linux HTB Queues and DSCP Mapping for Cloud Scenario.
- Figure 3.47: Configuration of Bridges, Routes and IP Addresses of Hosts.
- Figure 3.48: Switches Joining QoS and HTB Queue Creation.
- Figure 3.49: HTB with Settings of QoS Transfer Limits.
- Figure 3.50: Flow Table Entries with IP Addresses and Default Routes.
- Figure 3.51: QoS Classes with DSCP Settings on Switch1.
- Figure 3.52: Default Policy Class and Captured Packets.
- Figure 3.53: Custom QoS Topo without Separation.
- Figure 3.54: HTB Queue and QoS Rules on Switch1.
- Figure 3.55: QoS Rules on Switch1 after Switches connected to the REST QoS API with OVS.
- Figure 3.56: Linux HTB Queues and DSCP Mapping for Unsliced QoS Topo.
- Figure 3.57: QoS Meters and Rules on Switch2 and Switch3 with OVS.
- Figure 3.58: Custom QoS Topo with Separation.
- Figure 3.59: QoS Rules on Switch1 after Switches connected to the REST QoS API with OFSoftSwitch13.
- Figure 3.60: QoS Meters and Rules on Switch2 and Switch3 with OFSoftSwitch13.
- Figure 3.61: Retrieval of QoS Meters and Rules on Switch2 and Switch3 with OFSoftSwitch13.
- Figure 3.62: Default Policy Class and Captured Packets.
- Figure 3.63: ICMP Wireshark Capture.
- Figure 3.64: FTP Policy Class and Captured Packets.
- Figure 3.65: FTP Wireshark Capture.
- Figure 3.66: Web Policy Class and Captured Packets.

- Figure 3.67: HTTP Wireshark Capture.
- Figure 3.68: NBAR Protocol Discovery with 10 MB Sample for FTP and 100 MB Sample for HTTP.
- Figure 3.69: NBAR Protocol Discovery with 100 MB Sample for FTP and 1000 MB Sample for HTTP.
- Figure 3.70: Link Congestion with 100 MB Sample for FTP and 1000 MB Sample for HTTP.
- Figure 3.71: FTP Transfer via TE Tunnel Bandwidth Test.
- Figure 3.72: NBAR for FTP and ICMP of PE MPLS Domain.
- Figure 3.73: NBAR for HTTP of PE MPLS Domain and HTTP Transfer via TE Tunnel Bandwidth Test.
- Figure 3.74: Web and Data Traffic via TE Tunnel Bandwidth Test.
- Figure 3.75: Maximum Link Bandwidth Test for Web and Data Traffic.
- Figure 3.76: Wireshark Capture of VPN Label for FTP on NIC1.
- Figure 3.77: Wireshark Capture of VPN Label for HTTP on NIC3.
- Figure 3.78: Server for iPerf on Ports 5000, 21 and 80.
- Figure 3.79: Simultaneous Requests to Ports 5000, 21 and 80.
- Figure 3.80: Requests to Port 80 for Web Server.
- Figure 3.81: Requests to Port 5000 for UDP Traffic and to Port 21 for FTP Server.
- Figure 3.82: Simultaneous Requests to Ports 5000, 21 and 80 with DSCP.
- Figure 3.83: Requests to Port 21 and to Port 80 with DSCP.
- Figure 3.84: Requests to Port 5000 and Port 80 vs Port 5000 and Port 21 with DSCP.
- Figure 3.85: Retrieval of QoS Rules on Switch1 and Switch2.
- Figure 3.86: ICMP Requests between Hosts in Unsliced QoS Topo.
- Figure 3.87: Requests to Port 5001 (BE) and Port 5002 (AF42) from Host2.
- Figure 3.88: Requests to Port 5003 (BE) from Host3 and Port 5002 (AF42) from Host2.
- Figure 3.89: Retrieval of QoS Meters on Switch2 and Switch3 with OFSoftSwitch13.
- Figure 3.90: ICMP Requests between Hosts in Sliced QoS Topo.
- Figure 3.91: MPLS TE Topology.

- Figure 3.92: MPLS TE Topology DSCP to EXP Mappings.
- Figure 3.93: ICMP Test with ToS and TE.
- Figure 3.94: FTP and HTTP Tests with DSCP to EXP Markings and TE.
- Figure 3.95: DSCP Markings for Forwarded TCP Packets on CE Nodes.
- Figure 3.96: OF Topology with Floodlight without TE.
- Figure 3.97: ICMP Requests between Hosts and Flow Entries on the Switches.
- Figure 3.98: OF Topology with Floodlight with TE.
- Figure 3.99: Flow Entries on the Switches with TE.
- Figure 3.100: Switch3 Details with Floodlight and TE.
- Figure 3.101: Comparison of Delay without and with TE.
- Figure 3.102: OF Topology with HPE Aruba VAN without TE.
- Figure 3.103: OF Topology with HPE Aruba VAN with TE.
- Figure 3.104: Flow Entries with HPE Aruba VAN of Switch2 with TE.
- Figure 3.105: Flow Entries with HPE Aruba VAN of Switch3 with TE.
- Figure 3.106: Comparison of Delay without and with TE between Floodlight and HPE Controllers.
- Figure 3.107: Demo SP Topology.
- Figure 3.108: Creation of LSP with Pathman Apps and BGP.
- Figure 3.109: Point to Point Path Calculation with PCEP.
- Figure 3.110: MPLS Labels between Segmented Endpoints.
- Figure 3.111: Signaling for my_lsp on sea Node.
- Figure 3.112: PCEP Topology out of ODL Controller.
- Figure 3.113: SR Path between min and por Nodes.
- Figure 3.114: Traceroute between VM1 and VM2 via Red Path.
- Figure 3.115: Traceroute between VM1 and VM2 via Green Path.
- Figure 3.116: FTP Data Transfer from VM2 via Green Path.
- Figure 3.117: Datacentre Topo,4,4 in Mininet with HPE Aruba VAN Controller.
- Figure 3.118: HPE Aruba VAN Controller and New Flow Path after vNIC Failure.

List of Appendices

Appendix 1

MPLS1/2

Install Linux Kernel 4.12.0 via Ubuntu Kernel Update Utility:

```
sudo add-apt-repository ppa:teejee2008/ppa  
sudo apt-get update  
sudo apt-get install ukuu
```

Update Required Package Dependencies:

```
sudo apt-get update  
sudo apt-get upgrade  
sudo apt-get install flex bison
```

Download IPRoute 4.12.0 from <https://www.kernel.org/pub/linux/utils/net/iproute2/>:

```
tar xf iproute2-4.12.0.tar.xz  
cd iproute2-4.12.0  
.configure  
make  
sudo make install
```

Install Traceroute:

```
sudo apt-get install traceroute
```

Load MPLS:

```
sudo modprobe mpls_router  
sudo modprobe mpls_gso  
sudo modprobe mpls_iptunnel  
sudo sysctl -w net.mpls.conf.eth1.input=1  
sudo sysctl -w net.mpls.platform_labels=1048575
```

Install MGEN 5.02:

```
sudo apt-get update  
sudo apt-get install libc6 libgcc1 libstdc++6  
sudo apt-get install mgen
```

Install iPerf 3.0.11

```
sudo apt-get update  
sudo apt-get install iperf3
```

Install iPerf 2.0.5

```
sudo apt-get update  
sudo apt-get install iperf
```

Install Java 1.8.0

```
sudo apt-get update  
sudo apt-get install default-jre
```

Download jPerf 2.0.2 from <https://code.google.com/archive/p/xjperf/downloads>:

```
cd iperf-2.0.2  
sudo chmod +x jperf.sh  
sudo ./jperf.sh
```

Remove avahi-daemon:

```
sudo apt-get remove avahi-daemon  
sudo apt-get install sysv-rc-conf  
sudo sysv-rc-conf avahi-daemon off
```

Appendix 2

OpenFlow1/2/3

Install Linux Kernel 4.12.0 via Ubuntu Kernel Update Utility:

```
sudo add-apt-repository ppa:teejee2008/ppa  
sudo apt-get update  
sudo apt-get install ukuu
```

Update Required Package Dependencies:

```
sudo apt-get update  
sudo apt-get upgrade  
sudo apt-get install flex bison
```

Download IPRoute 4.12.0 from <https://www.kernel.org/pub/linux/utils/net/iproute2/>:

```
tar xf iproute2-4.12.0.tar.xz  
cd iproute2-4.12.0  
.configure  
make  
sudo make install
```

Install Traceroute:

```
sudo apt-get install traceroute
```

Install Git:

```
sudo apt-get install git
```

Install OpenFlow:

```
git clone git://gitosis.stanford.edu/openflow.git  
cd openflow  
git tag  
git checkout -b openflow-1.0.0
```

```
sudo apt-get install autoconf  
.boot.sh  
.configure  
make  
sudo make install
```

Install Wireshark:

```
sudo add-apt-repository ppa:wireshark-dev/stable  
sudo apt-get update  
sudo apt-get install wireshark  
sudo chmod +x /usr/bin/dumpcap
```

Remove avahi-daemon:

```
sudo apt-get remove avahi-daemon  
sudo apt-get install sysv-rc-conf  
sudo sysv-rc-conf avahi-daemon off
```

Disable IPv6:

```
sudo nano /etc/sysctl.conf  
net.ipv6.conf.all.disable_ipv6 = 1  
net.ipv6.conf.default.disable_ipv6 = 1  
sudo nano /etc/modprobe.d/blacklist.conf  
blacklist net-pf-10  
blacklist ipv6  
sudo shutdown -r now
```

Install MGGEN 5.02:

```
sudo apt-get update  
sudo apt-get install libc6 libgcc1 libstdc++6  
sudo apt-get install mgen
```

Install iPerf 3.0.11

```
sudo apt-get update
```

```
sudo apt-get install iperf3
```

Install iPerf 2.0.5

```
sudo apt-get update
```

```
sudo apt-get install iperf
```

Install Java 1.8.0

```
sudo apt-get update
```

```
sudo apt-get install default-jre
```

Download jPerf 2.0.2 from <https://code.google.com/archive/p/xjperf/downloads>:

```
cd iperf-2.0.2
```

```
sudo chmod +x jperf.sh
```

```
sudo ./jperf.sh
```

Appendix 3

VM1 (OpenFlow1)

```
sudo ifconfig eth0 192.16.10.10 netmask 255.255.255.0 up
```

```
sudo ip route add 192.16.20.0/24 via 192.16.10.2
```

```
sudo ip route add default via 192.16.10.2 dev eth0
```

VM2 (OpenFlow2)

```
sudo ifconfig eth0 192.16.20.20 netmask 255.255.255.0 up
```

```
sudo ip route add 192.16.10.0/24 via 192.16.20.4
```

```
sudo ip route add default via 192.16.20.4 dev eth0
```

Dublin Cisco MPLS - LSR

```
enable
```

```
configure terminal
```

```
no mpls ldp advertise-labels
```

```

mpls label range 10001 1048575 static 16 10000
mpls static binding ipv4 192.16.10.0 255.255.255.0 2001
mpls static binding ipv4 192.16.20.0 255.255.255.0 1001
mpls static binding ipv4 192.16.10.0 255.255.255.0 output 100.0.2.2 1001
mpls static binding ipv4 192.16.20.0 255.255.255.0 output 100.0.6.4 2001
ip route 192.16.10.0 255.255.255.0 FastEthernet0/1
ip route 192.16.20.0 255.255.255.0 FastEthernet0/0
end
configure terminal
mpls ip
interface FastEthernet0/1
ip address 100.0.2.3 255.255.255.0
mpls ip
no shutdown
end
configure terminal
mpls ip
interface FastEthernet0/0
ip address 100.0.6.3 255.255.255.0
mpls ip
no shutdown
end
copy running-config startup-config

```

Kildare (MPLS1) - LER

```

sudo modprobe mpls_router
sudo modprobe mpls_gso
sudo modprobe mpls_iptunnel
sudo sysctl -w net.mpls.conf.eth1.input=1
sudo sysctl -w net.mpls.platform_labels=1048575
sudo ifconfig eth0 192.16.10.2 netmask 255.255.255.0 up

```

```
sudo ifconfig eth1 100.0.2.2 netmask 255.255.255.0 up
sudo bash -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
sudo bash -c "echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter"
sudo ip route add 192.16.20.0/24 encap mpls 2001 via inet 100.0.2.3
sudo ip route add default via 100.0.2.2 dev eth1
```

Laois (MPLS2) - LER

```
sudo modprobe mpls_router
sudo modprobe mpls_gso
sudo modprobe mpls_iptunnel
sudo sysctl -w net.mpls.conf.eth1.input=1
sudo sysctl -w net.mpls.platform_labels=1048575
sudo ifconfig eth0 192.16.20.4 netmask 255.255.255.0 up
sudo ifconfig eth1 100.0.6.4 netmask 255.255.255.0 up
sudo bash -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
sudo bash -c "echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter"
sudo ip route add 192.16.10.0/24 encap mpls 1001 via inet 100.0.6.3
sudo ip route add default via 100.0.6.4 dev eth1
```

Dublin Cisco MPLS - LER

```
enable
configure terminal
hostname Dublin
no mpls ldp advertise-labels
mpls label range 10001 1048575 static 16 10000
mpls static binding ipv4 192.16.20.0 255.255.255.0 1001
mpls static binding ipv4 192.16.20.0 255.255.255.0 output 100.0.2.3 2001
ip route 100.0.6.0 255.255.255.0 FastEthernet0/1
ip route 192.16.20.0 255.255.255.0 FastEthernet0/1
end
configure terminal
```

```
mpls ip
interface FastEthernet0/0
ip address 192.16.10.2 255.255.255.0
no shutdown
end
configure terminal
mpls ip
interface FastEthernet0/1
ip address 100.0.2.2 255.255.255.0
mpls ip
no shutdown
end
copy running-config startup-config
```

Kildare (MPLS1) - LSR

```
sudo modprobe mpls_router
sudo modprobe mpls_gso
sudo modprobe mpls_iptunnel
sudo sysctl -w net.mpls.conf.eth0.input=1
sudo sysctl -w net.mpls.conf.eth1.input=1
sudo sysctl -w net.mpls.platform_labels=1048575
sudo ifconfig eth0 100.0.2.3 netmask 255.255.255.0 up
sudo ifconfig eth1 100.0.6.3 netmask 255.255.255.0 up
sudo bash -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
sudo bash -c "echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter"
sudo ip route add 192.16.10.0/24 encap mpls 1001 via inet 100.0.2.2
sudo ip route add 192.16.20.0/24 encap mpls 2001 via inet 100.0.6.4
sudo ip -f mpls route add 1001 via inet 100.0.2.3
sudo ip -f mpls route add 2001 via inet 100.0.6.3
```

Laois (MPLS2) - LER

```
sudo modprobe mpls_router  
sudo modprobe mpls_gso  
sudo modprobe mpls_iptunnel  
sudo sysctl -w net.mpls.conf.eth1.input=1  
sudo sysctl -w net.mpls.platform_labels=1048575  
sudo ifconfig eth0 192.16.20.4 netmask 255.255.255.0 up  
sudo ifconfig eth1 100.0.6.4 netmask 255.255.255.0 up  
sudo bash -c "echo 1 > /proc/sys/net/ipv4/ip_forward"  
sudo bash -c "echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter"  
sudo ip route add 192.16.10.0/24 encap mpls 1001 via inet 100.0.6.3  
sudo ip -f mpls route add 2001 via inet 100.0.6.4  
sudo ip route add default via 100.0.6.3 dev eth0
```

Appendix 4

VM1 (OpenFlow1) - P2P

```
sudo ifconfig eth0 192.16.10.10 netmask 255.255.255.0 up
```

VM2 (OpenFlow2) - P2P

```
sudo ifconfig eth0 192.16.10.20 netmask 255.255.255.0 up
```

VM1 (OpenFlow1) - IP Forwarding

```
sudo ifconfig eth0 192.16.10.10 netmask 255.255.255.0 up
```

```
sudo ip route add 192.16.20.0/24 via 192.16.10.2
```

```
sudo ip route add default via 192.16.10.2 dev eth0
```

VM2 (OpenFlow2) - IP Forwarding

```
sudo ifconfig eth0 192.16.20.20 netmask 255.255.255.0 up
```

```
sudo ip route add 192.16.10.0/24 via 192.16.20.4
```

```
sudo ip route add default via 192.16.20.4 dev eth0
```

Dublin - IP Forwarding

```
enable
```

```
configure terminal
```

```
hostname Dublin
```

```

no ip domain-lookup

ip route 192.16.10.0 255.255.255.0 FastEthernet0/1

ip route 192.16.20.0 255.255.255.0 FastEthernet0/0

end

configure terminal

interface FastEthernet0/1

ip address 100.0.2.3 255.255.255.0

no shutdown

end

configure terminal

interface FastEthernet0/0

ip address 100.0.6.3 255.255.255.0

no shutdown

end

copy running-config startup-config

```

Kildare (MPLS1) - IP Forwarding

```

sudo ifconfig eth0 192.16.10.2 netmask 255.255.255.0 up

sudo ifconfig eth1 100.0.2.2 netmask 255.255.255.0 up

sudo bash -c "echo 1 > /proc/sys/net/ipv4/ip_forward"

sudo bash -c "echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter"

sudo ip route add 192.16.20.0/24 encaps mpls 2001 via inet 100.0.2.3

sudo ip route add default via 100.0.2.2 dev eth1

sudo ip route add 100.0.6.0/24 via 100.0.2.3 dev eth1

sudo ip route add 192.16.20.0/24 via 100.0.2.3 dev eth1

```

Laois (MPLS2) - IP Forwarding

```

sudo ifconfig eth0 192.16.20.4 netmask 255.255.255.0 up

sudo ifconfig eth1 100.0.6.4 netmask 255.255.255.0 up

sudo bash -c "echo 1 > /proc/sys/net/ipv4/ip_forward"

sudo bash -c "echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter"

```

```
sudo ip route add 192.16.10.0/24 encap mpls 1001 via inet 100.0.6.3  
sudo ip route add default via 100.0.6.4 dev eth1  
sudo ip route add 100.0.2.0/24 via 100.0.6.3 dev eth1  
sudo ip route add 192.16.10.0/24 via 100.0.6.3 dev eth1
```

Dublin Cisco MPLS - LSR

```
enable  
configure terminal  
no mpls ldp advertise-labels  
mpls label range 10001 1048575 static 16 10000  
mpls static binding ipv4 192.16.10.0 255.255.255.0 2001  
mpls static binding ipv4 192.16.20.0 255.255.255.0 1001  
mpls static binding ipv4 192.16.10.0 255.255.255.0 output 100.0.2.2 1001  
mpls static binding ipv4 192.16.20.0 255.255.255.0 output 100.0.6.4 2001  
ip route 192.16.10.0 255.255.255.0 FastEthernet0/0  
ip route 192.16.20.0 255.255.255.0 FastEthernet0/1  
end  
configure terminal  
mpls ip  
interface FastEthernet0/1  
ip address 100.0.6.3 255.255.255.0  
mpls ip  
no shutdown  
end  
configure terminal  
mpls ip  
interface FastEthernet0/0  
ip address 100.0.2.3 255.255.255.0  
mpls ip  
no shutdown  
end
```

```
copy running-config startup-config
```

Kildare Cisco MPLS - LER

```
enable
```

```
configure terminal
```

```
hostname Kildare
```

```
no ip domain-lookup
```

```
no mpls ldp advertise-labels
```

```
mpls label range 10001 1048575 static 16 10000
```

```
mpls static binding ipv4 192.16.20.0 255.255.255.0 1001
```

```
mpls static binding ipv4 192.16.20.0 255.255.255.0 output 100.0.2.3 2001
```

```
ip route 0.0.0.0 0.0.0.0 FastEthernet0/1
```

```
end
```

```
configure terminal
```

```
mpls ip
```

```
interface FastEthernet0/0
```

```
ip address 192.16.10.2 255.255.255.0
```

```
no shutdown
```

```
end
```

```
configure terminal
```

```
interface FastEthernet0/1
```

```
ip address 100.0.2.2 255.255.255.0
```

```
mpls ip
```

```
no shutdown
```

```
end
```

```
copy running-config startup-config
```

Laois Cisco MPLS - LER

```
enable
```

```
configure terminal
```

```
hostname Laois
```

```

no ip domain-lookup
no mpls ldp advertise-labels
mpls label range 10001 1048575 static 16 10000
mpls static binding ipv4 192.16.10.0 255.255.255.0 2001
mpls static binding ipv4 192.16.10.0 255.255.255.0 output 100.0.6.3 1001
ip route 0.0.0.0 0.0.0.0 FastEthernet0/1
end
configure terminal
mpls ip
interface FastEthernet0/0
ip address 192.16.20.4 255.255.255.0
no shutdown
end
configure terminal
interface FastEthernet0/1
ip address 100.0.6.4 255.255.255.0
mpls ip
no shutdown
end
copy running-config startup-config

```

Mininet OpenFlow Performance Topology - OpenFlow

```

sudo ./OpenFlow1.py

```

```

#!/usr/bin/python

import re
import sys

from mininet.cli import CLI
from mininet.log import setLogLevel, info, error

```

```

from mininet.net import Mininet
from mininet.link import Intf
from mininet.topolib import TreeTopo
from mininet.util import quietRun

def checkIntf( intf ):
    "Make sure intf exists and is not configured"
    config = quietRun( 'ifconfig %s 2>/dev/null' % intf, shell=True )
    if not config:
        error( 'Error:', intf, 'does not exist!\n' )
        exit( 1 )
    ips = re.findall( r'\d+\.\d+\.\d+\.\d+', config )
    if ips:
        error( 'Error:', intf, 'has an IP address,'
               'and is probably in use!\n' )
        exit( 1 )

if __name__ == '__main__':
    setLogLevel( 'info' )

    # Gets HW interface from the command line
    intfName = sys.argv[ 1 ] if len( sys.argv ) > 1 else 'enp0s8'
    info( '*** Connecting to hw intf: %s' % intfName )

    info( '*** Checking', intfName, '\n' )
    checkIntf( intfName )

    info( '*** Creating network\n' )
    net = Mininet()
    Controller = net.addController('C0',ip='127.0.0.1')

```

```

# Creates two hosts

VM1 = net.addHost('VM1',ip='192.16.20.2/24', defaultRoute='via 192.16.20.1')
VM2 = net.addHost('VM2',ip='192.16.20.4/24', defaultRoute='via 192.16.20.1')

# Creates three switches

S1 = net.addSwitch('S1')
S2 = net.addSwitch('S2')
S3 = net.addSwitch('S3')

# Adds links between the switches and each host

net.addLink(VM1,S1)
net.addLink(S1,S3)
net.addLink(S3,S2)
net.addLink(VM2,S2)

switch = net.switches[ 0 ]
info( "*** Adding hardware interface", intfName, 'to switch',
      S3, '\n' )
_intf = Intf( intfName, node=S3 )

info( "*** Note: Reconfigure the interfaces for '
      'the Mininet hosts:\n', net.hosts, '\n' )

net.start()
CLI( net )
net.stop()

```

xterm S1 - OpenFlow

```

dpctl add-flow tcp:127.0.0.1:6653 in_port=1,idle_timeout=0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6653 in_port=2,idle_timeout=0,actions=output:1

```

xterm S3 - OpenFlow

```
dpctl add-flow tcp:127.0.0.1:6653 in_port=1,idle_timeout=0,actions=output:2  
dpctl add-flow tcp:127.0.0.1:6653 in_port=2,idle_timeout=0,actions=output:1
```

xterm S2 - OpenFlow

```
dpctl add-flow tcp:127.0.0.1:6653 in_port=1,idle_timeout=0,actions=output:2  
dpctl add-flow tcp:127.0.0.1:6653 in_port=2,idle_timeout=0,actions=output:1
```

SDN - OpenFlow

```
sudo nano /etc/network/interfaces
```

```
auto eth0
```

```
iface eth0 inet static
```

```
    address 192.16.20.30/24
```

```
    gateway 192.16.20.1
```

```
auto eth1
```

```
iface eth1 inet manual
```

```
pre-up ifconfig eth1 up
```

```
sudo nano /etc/resolvconf/resolv.conf.d/base
```

```
nameserver 8.8.8.8
```

```
nameserver 8.8.4.4
```

VM1 (OpenFlow1) - OpenFlow

```
sudo nano /etc/network/interfaces
```

```
auto eth0
```

```
iface eth0 inet static
```

```
    address 192.16.20.10/24
```

```
    gateway 192.16.20.1
```

```
sudo nano /etc/resolvconf/resolv.conf.d/base
```

```
nameserver 8.8.8.8
```

```
nameserver 8.8.4.4
```

VM2 (OpenFlow2) - OpenFlow

```
sudo nano /etc/network/interfaces
```

```
auto eth0
```

```
iface eth0 inet static
```

```
    address 192.16.20.20/24
```

```
    gateway 192.16.20.1
```

```
sudo nano /etc/resolvconf/resolv.conf.d/base
```

```
nameserver 8.8.8.8
```

```
nameserver 8.8.4.4
```

Appendix 5

To start the MG5 Server run: mgen input (*TEST*) output *x-output-TEST-Y.csv*

VM2 MG5 Server (1-listen-50bm-a.mgn) - P2P

```
LISTEN UDP 10001-10003 INTERFACE eth0 RXBUFFER 13107200
```

VM2 MG5 Server (2-listen-100bm-a.mgn) - P2P

```
LISTEN UDP 20001-20003 INTERFACE eth0 RXBUFFER 13107200
```

VM2 MG5 Server (3-listen-1000bh-a.mgn) - P2P

```
LISTEN UDP 30001-30003 INTERFACE eth0 RXBUFFER 13107200
```

VM2 MG5 Server (4-listen-100bl-a.mgn) - P2P

```
LISTEN UDP 40001-40003 INTERFACE eth0 RXBUFFER 13107200
```

VM2 MG5 Server (1-listen-50bm-b.mgn) - IP Forwarding

```
LISTEN UDP 10001-10003 INTERFACE eth0 RXBUFFER 13107200
```

VM2 MG5 Server (2-listen-100bm-b.mgn) - IP Forwarding

```
LISTEN UDP 20001-20003 INTERFACE eth0 RXBUFFER 13107200
```

VM2 MG5 Server (3-listen-1000bh-b.mgn) - IP Forwarding

```
LISTEN UDP 30001-30003 INTERFACE eth0 RXBUFFER 13107200
```

VM2 MG5 Server (4-listen-100bl-b.mgn) - IP Forwarding

```
LISTEN UDP 40001-40003 INTERFACE eth0 RXBUFFER 13107200
```

VM2 MG5 Server (1-listen-50bm-c.mgn) - MPLS in Linux

LISTEN UDP 10001-10003 INTERFACE eth0 RXBUFFER 13107200

VM2 MG5 Server (2-listen-100bm-c.mgn) - MPLS in Linux

LISTEN UDP 20001-20003 INTERFACE eth0 RXBUFFER 13107200

VM2 MG5 Server (3-listen-1000bh-c.mgn) - MPLS in Linux

LISTEN UDP 30001-30003 INTERFACE eth0 RXBUFFER 13107200

VM2 MG5 Server (4-listen-100bl-c.mgn) - MPLS in Linux

LISTEN UDP 40001-40003 INTERFACE eth0 RXBUFFER 13107200

VM2 MG5 Server (1-listen-50bm-d.mgn) - Cisco MPLS

LISTEN UDP 10001-10003 INTERFACE eth0 RXBUFFER 13107200

VM2 MG5 Server (2-listen-100bm-d.mgn) - Cisco MPLS

LISTEN UDP 20001-20003 INTERFACE eth0 RXBUFFER 13107200

VM2 MG5 Server (3-listen-1000bh-d.mgn) - Cisco MPLS

LISTEN UDP 30001-30003 INTERFACE eth0 RXBUFFER 13107200

VM2 MG5 Server (4-listen-100bl-d.mgn) - Cisco MPLS

LISTEN UDP 40001-40003 INTERFACE eth0 RXBUFFER 13107200

SDN MG5 Server (1-listen-50bm-e.mgn) - OpenFlow

LISTEN UDP 10001-10003 INTERFACE eth0 RXBUFFER 13107200

SDN MG5 Server (2-listen-100bm-e.mgn) - OpenFlow

LISTEN UDP 20001-20003 INTERFACE eth0 RXBUFFER 13107200

SDN MG5 Server (3-listen-1000bh-e.mgn) - OpenFlow

LISTEN UDP 30001-30003 INTERFACE eth0 RXBUFFER 13107200

SDN MG5 Server (4-listen-100bl-e.mgn) - OpenFlow

LISTEN UDP 40001-40003 INTERFACE eth0 RXBUFFER 13107200

To start the MG5 Client run: mgen input (*TEST*)

VM1 MG5 Client (1-input-50bm-a.mgn) - P2P

0.0 ON 1 UDP DST 192.16.10.20/10001 PERIODIC [25000.0 50] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.10.20/10002 PERIODIC [25000.0 50] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.10.20/10003 PERIODIC [25000.0 50] TXBUFFER 4369067
10.0 OFF 1
10.0 OFF 2
10.0 OFF 3

VM1 MGEN5 Client (2-input-100bm-a.mgn) - P2P

0.0 ON 1 UDP DST 192.16.10.20/20001 PERIODIC [12500.0 100] TXBUFFER 4369066
0.0 ON 2 UDP DST 192.16.10.20/20002 PERIODIC [12500.0 100] TXBUFFER 4369067
0.0 ON 3 UDP DST 192.16.10.20/20003 PERIODIC [12500.0 100] TXBUFFER 4369067
10.0 OFF 1
10.0 OFF 2
10.0 OFF 3

VM1 MGEN5 Client (3-input-1000bh-a.mgn) - P2P

0.0 ON 1 UDP DST 192.16.10.20/30001 PERIODIC [3000.0 1000] TXBUFFER 4369066
0.0 ON 2 UDP DST 192.16.10.20/30002 PERIODIC [3000.0 1000] TXBUFFER 4369067
0.0 ON 3 UDP DST 192.16.10.20/30003 PERIODIC [3000.0 1000] TXBUFFER 4369067
10.0 OFF 1
10.0 OFF 2
10.0 OFF 3

VM1 MGEN5 Client (4-input-100bl-a.mgn) - P2P

0.0 ON 1 UDP DST 192.16.10.20/40001 PERIODIC [6000.0 100] TXBUFFER 4369066
0.0 ON 2 UDP DST 192.16.10.20/40002 PERIODIC [6000.0 100] TXBUFFER 4369067
0.0 ON 3 UDP DST 192.16.10.20/40003 PERIODIC [6000.0 100] TXBUFFER 4369067
10.0 OFF 1
10.0 OFF 2
10.0 OFF 3

VM1 MGEN5 Client (1-input-50bm-b.mgn) - IP Forwarding

0.0 ON 1 UDP DST 192.16.20.20/10001 PERIODIC [25000.0 50] TXBUFFER 4369066
0.0 ON 2 UDP DST 192.16.20.20/10002 PERIODIC [25000.0 50] TXBUFFER 4369067
0.0 ON 3 UDP DST 192.16.20.20/10003 PERIODIC [25000.0 50] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MG5 Client (2-input-100bm-b.mgn) - IP Forwarding

0.0 ON 1 UDP DST 192.16.20.20/20001 PERIODIC [12500.0 100] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.20/20002 PERIODIC [12500.0 100] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.20/20003 PERIODIC [12500.0 100] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MG5 Client (3-input-1000bh-b.mgn) - IP Forwarding

0.0 ON 1 UDP DST 192.16.20.20/30001 PERIODIC [3000.0 1000] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.20/30002 PERIODIC [3000.0 1000] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.20/30003 PERIODIC [3000.0 1000] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MG5 Client (4-input-100bl-b.mgn) - IP Forwarding

0.0 ON 1 UDP DST 192.16.20.20/40001 PERIODIC [6000.0 100] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.20/40002 PERIODIC [6000.0 100] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.20/40003 PERIODIC [6000.0 100] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MG5 Client (1-input-50bm-c.mgn) - MPLS in Linux

0.0 ON 1 UDP DST 192.16.20.20/10001 PERIODIC [25000.0 50] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.20/10002 PERIODIC [25000.0 50] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.20/10003 PERIODIC [25000.0 50] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MG5 Client (2-input-100bm-c.mgn) - MPLS in Linux

0.0 ON 1 UDP DST 192.16.20.20/20001 PERIODIC [12500.0 100] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.20/20002 PERIODIC [12500.0 100] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.20/20003 PERIODIC [12500.0 100] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MG5 Client (3-input-1000bh-c.mgn) - MPLS in Linux

0.0 ON 1 UDP DST 192.16.20.20/30001 PERIODIC [3000.0 1000] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.20/30002 PERIODIC [3000.0 1000] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.20/30003 PERIODIC [3000.0 1000] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MG5 Client (4-input-100bl-c.mgn) - MPLS in Linux

0.0 ON 1 UDP DST 192.16.20.20/40001 PERIODIC [6000.0 100] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.20/40002 PERIODIC [6000.0 100] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.20/40003 PERIODIC [6000.0 100] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MG5 Client (1-input-50bm-d.mgn) - Cisco MPLS

0.0 ON 1 UDP DST 192.16.20.20/10001 PERIODIC [25000.0 50] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.20/10002 PERIODIC [25000.0 50] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.20/10003 PERIODIC [25000.0 50] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MG5 Client (2-input-100bm-d.mgn) - Cisco MPLS

0.0 ON 1 UDP DST 192.16.20.20/20001 PERIODIC [12500.0 100] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.20/20002 PERIODIC [12500.0 100] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.20/20003 PERIODIC [12500.0 100] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MG5 Client (3-input-1000bh-d.mgn) - Cisco MPLS

0.0 ON 1 UDP DST 192.16.20.20/30001 PERIODIC [3000.0 1000] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.20/30002 PERIODIC [3000.0 1000] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.20/30003 PERIODIC [3000.0 1000] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MG5 Client (4-input-100bl-d.mgn) - Cisco MPLS

0.0 ON 1 UDP DST 192.16.20.20/40001 PERIODIC [6000.0 100] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.20/40002 PERIODIC [6000.0 100] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.20/40003 PERIODIC [6000.0 100] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MG5 Client (1-input-50bm-e.mgn) - OpenFlow

0.0 ON 1 UDP DST 192.16.20.4/10001 PERIODIC [25000.0 50] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.4/10002 PERIODIC [25000.0 50] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.4/10003 PERIODIC [25000.0 50] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MGEN5 Client (2-input-100bm-e.mgn) - OpenFlow

0.0 ON 1 UDP DST 192.16.20.4/20001 PERIODIC [12500.0 100] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.4/20002 PERIODIC [12500.0 100] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.4/20003 PERIODIC [12500.0 100] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MGEN5 Client (3-input-1000bh-e.mgn) - OpenFlow

0.0 ON 1 UDP DST 192.16.20.4/30001 PERIODIC [3000.0 1000] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.4/30002 PERIODIC [3000.0 1000] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.4/30003 PERIODIC [3000.0 1000] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

VM1 MGEN5 Client (4-input-100bl-e.mgn) - OpenFlow

0.0 ON 1 UDP DST 192.16.20.4/40001 PERIODIC [6000.0 100] TXBUFFER 4369066

0.0 ON 2 UDP DST 192.16.20.4/40002 PERIODIC [6000.0 100] TXBUFFER 4369067

0.0 ON 3 UDP DST 192.16.20.4/40003 PERIODIC [6000.0 100] TXBUFFER 4369067

10.0 OFF 1

10.0 OFF 2

10.0 OFF 3

Appendix 6

VM1 (OpenFlow1) - IP Forwarding

sudo ifconfig eth0 192.16.10.10 netmask 255.255.255.0 up

sudo ip route add 192.16.20.0/24 via 192.16.10.2

```
sudo ip route add default via 192.16.10.2 dev eth0
```

VM2 (OpenFlow2) - IP Forwarding

```
sudo ifconfig eth0 192.16.20.20 netmask 255.255.255.0 up
```

```
sudo ip route add 192.16.10.0/24 via 192.16.20.4
```

```
sudo ip route add default via 192.16.20.4 dev eth0
```

Dublin - IP Forwarding

```
enable
```

```
configure terminal
```

```
hostname Dublin
```

```
no ip domain lookup
```

```
ip route 192.16.10.0 255.255.255.0 FastEthernet0/0
```

```
ip route 192.16.20.0 255.255.255.0 FastEthernet0/1
```

```
ip route 100.0.2.0 255.255.255.0 FastEthernet0/0
```

```
ip route 100.0.6.0 255.255.255.0 FastEthernet0/1
```

```
interface FastEthernet0/0
```

```
ip address 200.0.2.3 255.255.255.0
```

```
no shutdown
```

```
interface FastEthernet0/1
```

```
ip address 200.0.6.3 255.255.255.0
```

```
no shutdown
```

```
end
```

```
copy running-config startup-config
```

CSR1000V1 - IP Forwarding

```
enable
```

```
configure terminal
```

```
hostname CSR1000V1
```

```
no ip domain lookup
```

```
ip route 192.16.10.0 255.255.255.0 GigabitEthernet1
```

```
ip route 192.16.20.0 255.255.255.0 GigabitEthernet2
```

```
ip route 200.0.6.0 255.255.255.0 GigabitEthernet2
ip route 100.0.6.0 255.255.255.0 GigabitEthernet2
interface GigabitEthernet1
ip address 100.0.2.3 255.255.255.0
no shutdown
interface GigabitEthernet2
ip address 100.0.6.3 255.255.255.0
no shutdown
end
copy running-config startup-config
```

CSR1000V2 - IP Forwarding

```
enable
configure terminal
hostname CSR1000V2
no ip domain lookup
ip route 192.16.10.0 255.255.255.0 GigabitEthernet2
ip route 192.16.20.0 255.255.255.0 GigabitEthernet1
ip route 200.0.2.0 255.255.255.0 GigabitEthernet2
ip route 100.0.2.0 255.255.255.0 GigabitEthernet2
interface GigabitEthernet1
ip address 100.0.6.3 255.255.255.0
no shutdown
interface GigabitEthernet2
ip address 200.0.6.4 255.255.255.0
no shutdown
end
copy running-config startup-config
```

CSR1000V3 - IP Forwarding

```
enable
```

```
configure terminal  
hostname CSR1000V3  
no ip domain lookup  
ip route 192.16.20.0 255.255.255.0 GigabitEthernet1  
ip route 200.0.2.0 255.255.255.0 GigabitEthernet1  
ip route 200.0.6.0 255.255.255.0 GigabitEthernet1  
ip route 100.0.6.0 255.255.255.0 GigabitEthernet1  
interface GigabitEthernet1  
ip address 100.0.2.2 255.255.255.0  
no shutdown  
interface GigabitEthernet2  
ip address 192.16.10.2 255.255.255.0  
no shutdown  
end  
copy running-config startup-config
```

CSR1000V4 - IP Forwarding

```
enable  
configure terminal  
hostname CSR1000V4  
no ip domain lookup  
ip route 192.16.10.0 255.255.255.0 GigabitEthernet1  
ip route 200.0.6.0 255.255.255.0 GigabitEthernet1  
ip route 200.0.2.0 255.255.255.0 GigabitEthernet1  
ip route 100.0.2.0 255.255.255.0 GigabitEthernet1  
interface GigabitEthernet1  
ip address 100.0.6.4 255.255.255.0  
no shutdown  
interface GigabitEthernet2  
ip address 192.16.20.4 255.255.255.0  
no shutdown
```

```
end  
copy running-config startup-config
```

VM1 (OpenFlow1) - Cisco MPLS

```
sudo ifconfig eth0 192.16.10.10 netmask 255.255.255.0 up  
sudo ip route add 192.16.20.0/24 via 192.16.10.2  
sudo ip route add default via 192.16.10.2 dev eth0
```

VM2 (OpenFlow2) - Cisco MPLS

```
sudo ifconfig eth0 192.16.20.20 netmask 255.255.255.0 up  
sudo ip route add 192.16.10.0/24 via 192.16.20.4  
sudo ip route add default via 192.16.20.4 dev eth0
```

Dublin - Cisco MPLS

```
enable  
configure terminal  
hostname Dublin  
no ip domain lookup  
interface Loopback0  
ip address 1.1.1.1 255.255.255.255  
ip ospf 1 area 0  
interface FastEthernet0/0  
ip address 200.0.2.3 255.255.255.0  
ip ospf 1 area 0  
no shutdown  
interface FastEthernet0/1  
ip address 200.0.6.3 255.255.255.0  
ip ospf 1 area 0  
no shutdown  
exit  
router ospf 1  
mpls ldp autoconfig
```

```
end  
copy running-config startup-config
```

CSR1000V1 - Cisco MPLS

```
enable  
configure terminal  
hostname CSR1000V1  
no ip domain lookup  
interface Loopback0  
ip address 2.2.2.2 255.255.255.255  
ip ospf 1 area 0  
interface GigabitEthernet1  
ip address 100.0.2.3 255.255.255.0  
ip ospf 1 area 0  
no shutdown  
interface GigabitEthernet2  
ip address 100.0.6.3 255.255.255.0  
ip ospf 1 area 0  
no shutdown  
exit  
router ospf 1  
mpls ldp autoconfig  
end  
copy running-config startup-config
```

CSR1000V2 - Cisco MPLS

```
enable  
configure terminal  
hostname CSR1000V2  
no ip domain lookup  
interface Loopback0
```

```
ip address 3.3.3.3 255.255.255.255
ip ospf 1 area 0
interface GigabitEthernet1
ip address 100.0.6.3 255.255.255.0
ip ospf 1 area 0
no shutdown
interface GigabitEthernet2
ip address 200.0.6.4 255.255.255.0
ip ospf 1 area 0
no shutdown
exit
router ospf 1
mpls ldp autoconfig
end
copy running-config startup-config
```

CSR1000V3 - Cisco MPLS

```
enable
configure terminal
hostname CSR1000V3
no ip domain lookup
interface Loopback0
ip address 4.4.4.4 255.255.255.255
ip ospf 1 area 0
interface GigabitEthernet1
ip address 100.0.2.2 255.255.255.0
ip ospf 1 area 0
no shutdown
interface GigabitEthernet2
ip address 192.16.10.2 255.255.255.0
ip ospf 1 area 0
```

```
no shutdown  
exit  
router ospf 1  
mpls ldp autoconfig  
end  
copy running-config startup-config
```

CSR1000V4 - Cisco MPLS

```
enable  
configure terminal  
hostname CSR1000V4  
no ip domain lookup  
interface Loopback0  
ip address 5.5.5.5 255.255.255.255  
ip ospf 1 area 0  
interface GigabitEthernet1  
ip address 100.0.6.4 255.255.255.0  
ip ospf 1 area 0  
no shutdown  
interface GigabitEthernet2  
ip address 192.16.20.4 255.255.255.0  
ip ospf 1 area 0  
no shutdown  
router ospf 1  
mpls ldp autoconfig  
end  
copy running-config startup-config
```

VM1 (OpenFlow1) - MPLS in Linux (LDPinLinux.sh)

```
sudo ifconfig eth0 192.16.10.10 netmask 255.255.255.0 up  
sudo ip route add 192.16.20.0/24 via 192.16.10.2
```

```
sudo ip route add default via 192.16.10.2 dev eth0
```

VM2 (OpenFlow2) - MPLS in Linux (LDPinLinux.sh)

```
sudo ifconfig eth0 192.16.20.20 netmask 255.255.255.0 up
```

```
sudo ip route add 192.16.10.0/24 via 192.16.20.4
```

```
sudo ip route add default via 192.16.20.4 dev eth0
```

MPLS1 - MPLS in Linux (LDPinLinux.sh)

```
sudo modprobe mpls_router
```

```
sudo modprobe mpls_gso
```

```
sudo modprobe mpls_iptunnel
```

```
sudo sysctl -w net.mpls.conf.eth1.input=1
```

```
sudo sysctl -w net.mpls.platform_labels=1048575
```

```
sudo ifconfig eth0 192.16.10.2 netmask 255.255.255.0 up
```

```
sudo ifconfig eth1 100.0.2.2 netmask 255.255.255.0 up
```

```
sudo bash -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
```

```
sudo bash -c "echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter"
```

```
sudo ip link add name lo0 type dummy
```

```
sudo ip link set dev lo0 up
```

```
sudo ip addr add 4.4.4.4/32 dev lo0
```

MPLS2 - MPLS in Linux (LDPinLinux.sh)

```
sudo modprobe mpls_router
```

```
sudo modprobe mpls_gso
```

```
sudo modprobe mpls_iptunnel
```

```
sudo sysctl -w net.mpls.conf.eth1.input=1
```

```
sudo sysctl -w net.mpls.platform_labels=1048575
```

```
sudo ifconfig eth0 192.16.20.4 netmask 255.255.255.0 up
```

```
sudo ifconfig eth1 100.0.6.4 netmask 255.255.255.0 up
```

```
sudo bash -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
```

```
sudo bash -c "echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter"
```

```
sudo ip link add name lo0 type dummy
```

```
sudo ip link set dev lo0 up  
sudo ip addr add 5.5.5.5/32 dev lo0
```

MPLS1/2 - MPLS in Linux

Remove Quagga 0.99.24.1, Install Nmap 7.01 and FRRouting 3.1:

```
sudo apt-get remove --auto-remove quagga  
sudo apt-get purge --auto-remove quagga  
sudo rm -r /usr/local  
sudo apt-get update  
sudo apt-get upgrade  
sudo apt-get install git autoconf automake libtool make gawk libreadline-dev texinfo dejagnu  
pkg-config libpam0g-dev libjson-c-dev bison flex python-pytest libc-ares-dev python3-dev  
libsystemd-dev python-ipaddr nmap  
sudo groupadd -g 92 frr  
sudo groupadd -r -g 85 frrvty  
sudo adduser --system --ingroup frr --home /var/run/frr/ --gecos "FRR suite" --shell  
/sbin/nologin frr  
sudo usermod -a -G frrvty frr  
git clone https://github.com/frrouting/frr.git frr  
cd frr  
.bootstrap.sh  
.configure --prefix=/usr --enable-exemplifiedir=/usr/share/doc/frr/examples/ --  
localstatedir=/var/run/frr --sbindir=/usr/lib/frr --sysconfdir=/etc/frr --enable-pimd --enable-  
watchfrr --enable-ospfclient=yes --enable-ospfapi=yes --enable-multipath=64 --enable-  
user=frr --enable-group=frr --enable-vty-group=frrvty --enable-configfile-mask=0640 --  
enable-logfile-mask=0640 --enable-rtadv --enable-fpm --enable-systemd=yes --with-pkg-git-  
version --with-pkg-extra-version=-MyOwnFRR Version  
make  
make check  
sudo make install  
sudo install -m 755 -o frr -g frr -d /var/log/frr  
sudo install -m 775 -o frr -g frrvty -d /etc/frr
```

```

sudo install -m 640 -o frr -g frr /dev/null /etc/frr/zebra.conf
sudo install -m 640 -o frr -g frr /dev/null /etc/frr/bgpd.conf
sudo install -m 640 -o frr -g frr /dev/null /etc/frr/ospfd.conf
sudo install -m 640 -o frr -g frr /dev/null /etc/frr/ospf6d.conf
sudo install -m 640 -o frr -g frr /dev/null /etc/frr/isisd.conf
sudo install -m 640 -o frr -g frr /dev/null /etc/frr/ripd.conf
sudo install -m 640 -o frr -g frr /dev/null /etc/frr/ripngd.conf
sudo install -m 640 -o frr -g frr /dev/null /etc/frr/pimd.conf
sudo install -m 640 -o frr -g frr /dev/null /etc/frr/ldpd.conf
sudo install -m 640 -o frr -g frr /dev/null /etc/frr/nhrpd.conf
sudo install -m 640 -o frr -g frrvty /dev/null /etc/frr/vtysh.conf
sudo install -m 644 tools/frr.service /etc/systemd/system/frr.service
sudo install -m 644 tools/etc/default/frr /etc/default/frr
sudo install -m 644 tools/etc/frr/daemons /etc/frr/daemons
sudo install -m 644 tools/etc/frr/daemons.conf /etc/frr/daemons.conf
sudo install -m 644 tools/etc/frr/frr.conf /etc/frr/frr.conf
sudo install -m 644 -o frr -g frrvty tools/etc/frr/vtysh.conf /etc/frr/vtysh.conf
ll /etc/frr (check permissions)
sudo nano /etc/frr/daemons (edit: zebra=yes, ospfd=yes, ldpd=yes)
sudo systemctl enable frr
sudo systemctl start frr
nmap localhost (check listening ports: 2601 - zebra, 2604 - ospfd)
systemctl status frr

```

MPLS1 - MPLS in Linux (frr.conf)

frr version 3.1-dev-MyOwnFRRVersion

frr defaults traditional

hostname mpls1

no ipv6 forwarding

username cumulus nopassword

!

```
service integrated-vtysh-config
```

```
!
```

```
log syslog informational
```

```
!
```

```
interface eth0
```

```
ip ospf area 0.0.0.0
```

```
!
```

```
interface eth1
```

```
ip ospf area 0.0.0.0
```

```
!
```

```
interface lo0
```

```
ip ospf area 0.0.0.0
```

```
!
```

```
router ospf
```

```
!
```

```
mpls ldp
```

```
router-id 4.4.4.4
```

```
!
```

```
address-family ipv4
```

```
discovery transport-address 4.4.4.4
```

```
!
```

```
interface eth1
```

```
!
```

```
exit-address-family
```

```
!
```

```
!
```

```
line vty
```

```
!
```

MPLS2 - MPLS in Linux (frr.conf)

frr version 3.1-dev-MyOwnFRRVersion

```
frr defaults traditional
hostname mpls2
no ipv6 forwarding
username cumulus nopassword
!
service integrated-vtysh-config
!
log syslog informational
!
interface eth0
ip ospf area 0.0.0.0
!
interface eth1
ip ospf area 0.0.0.0
!
interface lo0
ip ospf area 0.0.0.0
!
router ospf
!
mpls ldp
router-id 5.5.5.5
!
address-family ipv4
discovery transport-address 5.5.5.5
!
interface eth1
!
exit-address-family
!
```

!

line vty

!

Mininet OpenFlow Scaled-Up Topology - OpenFlow

sudo ./OpenFlow2.py

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

```
import re
```

```
import sys
```

```
from mininet.cli import CLI
```

```
from mininet.log import setLogLevel, info, error
```

```
from mininet.net import Mininet
```

```
from mininet.link import Intf
```

```
from mininet.topolib import TreeTopo
```

```
from mininet.util import quietRun
```

```
def checkIntf( intf ):
```

```
    "Make sure intf exists and is not configured"
```

```
    config = quietRun( 'ifconfig %s 2>/dev/null' % intf, shell=True )
```

```
    if not config:
```

```
        error( 'Error:', intf, 'does not exist!\n' )
```

```
        exit( 1 )
```

```
    ips = re.findall( r'\d+\.\d+\.\d+\.\d+', config )
```

```
    if ips:
```

```
        error( 'Error:', intf, 'has an IP address,' )
```

```
        'and is probably in use!\n' )
```

```
        exit( 1 )
```

```

if __name__ == '__main__':
    setLogLevel( 'info' )

# Gets HW interface from the command line
intfName = sys.argv[ 1 ] if len( sys.argv ) > 1 else 'enp0s8'
info( '*** Connecting to hw intf: %s' % intfName )

info( '*** Checking', intfName, '\n' )
checkIntf( intfName )

info( '*** Creating network\n' )
net = Mininet()
Controller = net.addController('C0',ip='127.0.0.1')

# Creates two hosts
VM1 = net.addHost('VM1',ip='192.16.20.2/24', defaultRoute='via 192.16.20.1')
VM2 = net.addHost('VM2',ip='192.16.20.4/24', defaultRoute='via 192.16.20.1')

# Creates five switches
S1 = net.addSwitch('S1')
S2 = net.addSwitch('S2')
S3 = net.addSwitch('S3')
S4 = net.addSwitch('S4')
S5 = net.addSwitch('S5')

# Adds links between the switches and each host
net.addLink(VM1,S4)
net.addLink(S4,S1)
net.addLink(S1,S3)
net.addLink(S3,S2)

```

```

net.addLink(S2,S5)

net.addLink(VM2,S5)

switch = net.switches[ 0 ]

info( "*** Adding hardware interface', intfName, 'to switch',
      S3, '\n' )

_intf = Intf( intfName, node=S3 )

info( "*** Note: Reconfigure the interfaces for '
      'the Mininet hosts:\n', net.hosts, '\n' )

net.start()

CLI( net )

net.stop()

```

xterm S1 - OpenFlow

```

dpctl add-flow tcp:127.0.0.1:6653 in_port=1,idle_timeout=0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6653 in_port=2,idle_timeout=0,actions=output:1

```

xterm S3 - OpenFlow

```

dpctl add-flow tcp:127.0.0.1:6653 in_port=1,idle_timeout=0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6653 in_port=2,idle_timeout=0,actions=output:1

```

xterm S2 - OpenFlow

```

dpctl add-flow tcp:127.0.0.1:6653 in_port=1,idle_timeout=0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6653 in_port=2,idle_timeout=0,actions=output:1

```

xterm S4 - OpenFlow

```

dpctl add-flow tcp:127.0.0.1:6653 in_port=1,idle_timeout=0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6653 in_port=2,idle_timeout=0,actions=output:1

```

xterm S5 - OpenFlow

```

dpctl add-flow tcp:127.0.0.1:6653 in_port=1,idle_timeout=0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6653 in_port=2,idle_timeout=0,actions=output:1

```

SDN - OpenFlow

```
sudo nano /etc/network/interfaces
```

```
auto eth0
```

```
iface eth0 inet static
```

```
address 192.16.20.30/24
```

```
gateway 192.16.20.1
```

```
auto eth1
```

```
iface eth1 inet manual
```

```
pre-up ifconfig eth1 up
```

```
sudo nano /etc/resolvconf/resolv.conf.d/base
```

```
nameserver 8.8.8.8
```

```
nameserver 8.8.4.4
```

VM1 (OpenFlow1) - OpenFlow

```
sudo nano /etc/network/interfaces
```

```
auto eth0
```

```
iface eth0 inet static
```

```
address 192.16.20.10/24
```

```
gateway 192.16.20.1
```

```
sudo nano /etc/resolvconf/resolv.conf.d/base
```

```
nameserver 8.8.8.8
```

```
nameserver 8.8.4.4
```

VM2 (OpenFlow2) - OpenFlow

```
sudo nano /etc/network/interfaces
```

```
auto eth0
```

```
iface eth0 inet static
```

```
address 192.16.20.20/24
```

```
gateway 192.16.20.1
```

```
sudo nano /etc/resolvconf/resolv.conf.d/base
```

```
nameserver 8.8.8.8
```

```
nameserver 8.8.4.4
```

Appendix 7

VM1 (OpenFlow1) - Cisco MPLS (QoSInCisco.sh)

```
sudo ifconfig eth0 192.16.10.10 netmask 255.255.255.0 up
```

```
sudo ip route add 192.16.10.0/24 via 192.16.10.2
```

```
sudo ip route add default via 192.16.10.2 dev eth0
```

VM2 (OpenFlow2) - Cisco MPLS (QoSInCisco.sh)

```
sudo ifconfig eth0 192.16.20.20 netmask 255.255.255.0 up
```

```
sudo ip route add 192.16.20.0/24 via 192.16.20.4
```

```
sudo ip route add default via 192.16.20.4 dev eth0
```

Install vsFTPD 3.0.3:

```
sudo apt-get update
```

```
sudo apt-get install vsftpd
```

```
sudo cp /etc/vsftpd.conf /etc/vsftpd.conf.bak (backup of configuration)
```

```
sudo ufw status (firewall check)
```

```
sudo ufw allow 20/tcp (FTP port)
```

```
sudo ufw allow 21/tcp (FTP port)
```

```
sudo ufw allow 990/tcp (TLS port)
```

```
sudo ufw allow 40000:50000/tcp (Passive ports)
```

```
sudo adduser ftpuser (Password: password)
```

```
sudo mkdir /home/ftpuser/ftp
```

```
sudo chown nobody:nogroup /home/ftpuser/ftp
```

```
sudo chmod a-w /home/ftpuser/ftp
```

```
sudo ls -la /home/ftpuser/ftp (permissions check)
```

```
sudo mkdir /home/ftpuser/ftp/files
```

```
sudo chown ftpuser:ftpuser /home/ftpuser/ftp/files
```

```
sudo ls -la /home/ftpuser/ftp (permissions check)
```

```
echo "VSFTPD Test File" | sudo tee /home/ftpuser/ftp/files/test.txt (test file)
sudo nano /etc/vsftpd.conf (Uncomment: write_enable=YES, chroot_local_user=YES / Add:
user_sub_token=$USER, local_root=/home/$USER/ftp, pasv_min_port=40000,
pasv_max_port=50000, userlist_enable=YES, userlist_file=/etc/vsftpd.userlist,
userlist_deny=NO)
echo "ftpuser" | sudo tee -a /etc/vsftpd.userlist (adds user)
cat /etc/vsftpd.userlist (user check)
sudo systemctl restart vsftpd
namp 192.16.20.20 (open ports check: 21 FTP)
ftp 192.16.20.20
cd files
get test.txt (ftp download test)
put test.txt upload.txt (ftp upload test)
bye
```

Create Sample Files:

```
truncate -s 1000M /home/ftpuser/ftp/files/1000M.sample (1000 MB Sample File)
truncate -s 100M /home/ftpuser/ftp/files/1000M.sample (100 MB Sample File)
truncate -s 10M /home/ftpuser/ftp/files/1000M.sample (10 MB Sample File)
truncate -s 1M /home/ftpuser/ftp/files/1000M.sample (1 MB Sample File)
```

Install Apache 2.4.18:

```
sudo apt-get update
sudo apt-get install apache2
sudo ufw app list (application ports: Apache 80, Apache Full 80 & 443, Apache Secure 443)
sudo ufw allow 'Apache Full'
sudo ufw status (firewall check)
sudo systemctl restart apache2
sudo systemctl status apache2 (status check)
namp 192.16.20.20 (open ports check: FTP 21, HTTP 80)
```

```
sudo cp /home/openflow2/Downloads/index.html /var/www/html  
sudo cp /home/openflow2/Downloads/index.html /var/www/html (sample test page)  
sudo cp /home/ftpuser/ftp/files/*.sample /var/www/html (sample files)
```

Alter Sample Test Page (index.html):

```
sudo nano /var/www/html/index.html  
  
<div class="section_header">  
    <div id="about"></div>  
    <a href="1000MB.sample" >1000 MB Sample File</a>  
</div>  
  
<div class="section_header">  
    <div id="changes"></div>  
    <a href="100MB.sample" >100 MB Sample File</a>  
</div>  
  
<div class="section_header">  
    <div id="docroot"></div>  
    <a href="10MB.sample" >10 MB Sample File</a>  
</div>  
  
<div class="section_header">  
    <div id="bugs"></div>  
    <a href="1MB.sample" >1 MB Sample File</a>  
</div>
```

Dublin - Cisco MPLS

```
enable  
configure terminal  
hostname Dublin  
no ip domain lookup  
mpls label range 2000 2999  
line console 0  
exec-timeout 0
```

```
exit
interface FastEthernet0/0
ip address 200.0.2.3 255.255.255.0
mpls ip
no shutdown
exit
interface FastEthernet0/1
ip address 200.0.6.3 255.255.255.0
mpls ip
no shutdown
exit
router ospf 1
log-adjacency-changes
network 0.0.0.0 255.255.255.255 area 0
end
copy running-config startup-config
configure terminal
class-map match-all EXP1
match mpls experimental topmost 1
exit
policy-map QOS
class EXP1
police cir 256000 bc 32000 be 32000 conform-action transmit exceed-action drop
end
configure terminal
interface FastEthernet0/0
service-policy input QOS
ip nbar protocol-discovery
end
copy running-config startup-config
```

CSR1000V1 - Cisco MPLS

```
enable
configure terminal
license boot level premium
exit
write
reload
enable
configure terminal
hostname CSR1000V1
no ip domain lookup
enable secret csr
line vty 0 4
password csr
logging synchronous
exec-timeout 0
login
exit
ip vrf cust
rd 100:1
route-target export 100:1
route-target import 100:1
exit
mpls label range 1000 1999
interface Loopback100
ip address 100.100.100.1 255.255.255.255
exit
interface GigabitEthernet1
ip vrf forwarding cust
ip address 100.0.2.3 255.255.255.0
```

```
no shutdown
exit
interface GigabitEthernet2
ip address 200.0.2.2 255.255.255.0
mpls ip
no shutdown
exit
interface GigabitEthernet3
ip address 192.168.2.241 255.255.255.0
description TELNET -> Management Access
no shutdown
exit
router eigrp 65000
no auto-summary
address-family ipv4 vrf cust autonomous-system 100
eigrp log-neighbor-changes
no eigrp log-neighbor-warnings
redistribute bgp 100 metric 1 1 1 1 1
network 100.0.0.0
no auto-summary
exit-address-family
exit
router ospf 1
log-adjacency-changes
network 0.0.0.0 255.255.255.255 area 0
exit
router bgp 100
bgp log-neighbor-changes
no bgp default ipv4-unicast
neighbor 100.100.100.2 remote-as 100
```

```
neighbor 100.100.100.2 update-source Loopback100
address-family vpng4
neighbor 100.100.100.2 activate
neighbor 100.100.100.2 send-community extended
exit-address-family
address-family ipv4 vrf cust
redistribute eigrp 100
no synchronization
exit-address-family
end
copy running-config startup-config
configure terminal
class-map match-all FTP
match mpls experimental topmost 5
class-map match-all Web
match mpls experimental topmost 4
policy-map QOS
class FTP
bandwidth 1024
police cir 1024000 bc 128000 be 128000 conform-action transmit exceed-action set-mpls-
exp-topmost-transmit 1
class Web
priority 1024 128000
end
configure terminal
interface GigabitEthernet2
bandwidth 6144
load-interval 30
service-policy output QOS
ip nbar protocol-discovery
```

```
end  
copy running-config startup-config
```

CSR1000V2 - Cisco MPLS

```
enable  
configure terminal  
license boot level premium  
exit  
write  
reload  
enable  
configure terminal  
hostname CSR1000V2  
no ip domain lookup  
enable secret csr  
line vty 0 4  
password csr  
logging synchronous  
exec-timeout 0  
login  
exit  
ip vrf cust  
rd 100:1  
route-target export 100:1  
route-target import 100:1  
exit  
mpls label range 3000 3999  
interface Loopback100  
ip address 100.100.100.2 255.255.255.255  
exit  
interface GigabitEthernet1
```

```
ip vrf forwarding cust
ip address 100.0.6.3 255.255.255.0
no shutdown
exit
interface GigabitEthernet2
ip address 200.0.6.4 255.255.255.0
mpls ip
no shutdown
exit
interface GigabitEthernet3
ip address 192.168.2.242 255.255.255.0
description TELNET -> Management Access
no shutdown
exit
router eigrp 65000
no auto-summary
address-family ipv4 vrf cust autonomous-system 100
eigrp log-neighbor-changes
no eigrp log-neighbor-warnings
redistribute bgp 100 metric 1 1 1 1 1
network 100.0.0.0
no auto-summary
exit-address-family
exit
router ospf 1
log-adjacency-changes
network 0.0.0.0 255.255.255.255 area 0
exit
router bgp 100
bgp log-neighbor-changes
```

```
no bgp default ipv4-unicast
neighbor 100.100.100.1 remote-as 100
neighbor 100.100.100.1 update-source Loopback100
address-family vpng4
neighbor 100.100.100.1 activate
neighbor 100.100.100.1 send-community extended
exit-address-family
address-family ipv4 vrf cust
redistribute eigrp 100
no synchronization
exit-address-family
end
copy running-config startup-config
configure terminal
class-map match-all FTP
match mpls experimental topmost 5
class-map match-all Web
match mpls experimental topmost 4
policy-map QOS
class FTP
bandwidth 1024
police cir 1024000 bc 128000 be 128000 conform-action transmit exceed-action set-mpls-
exp-topmost-transmit 1
class Web
priority 1024 128000
end
configure terminal
interface GigabitEthernet1
bandwidth 6144
load-interval 30
```

```
service-policy output QOS
ip nbar protocol-discovery
end
copy running-config startup-config
```

CSR1000V3 - Cisco MPLS

```
enable
configure terminal
license boot level premium
exit
write
reload
enable
configure terminal
hostname CSR1000V3
no ip domain lookup
enable secret csr
line vty 0 4
password csr
logging synchronous
exec-timeout 0
login
exit
interface GigabitEthernet1
ip address 100.0.2.2 255.255.255.0
no shutdown
exit
interface GigabitEthernet2
ip address 192.16.10.2 255.255.255.0
no shutdown
exit
```

```
interface GigabitEthernet3
ip address 192.168.2.243 255.255.255.0
description TELNET -> Management Access
no shutdown
exit
router eigrp 100
eigrp log-neighbor-changes
no eigrp log-neighbor-warnings
network 192.16.10.2 0.0.0.255
network 100.0.2.2 0.0.0.255
passive-interface GigabitEthernet3
no auto-summary
end
copy running-config startup-config
configure terminal
class-map match-all FTP
match protocol ftp-data
class-map match-all Web
match protocol http
policy-map QOS
class FTP
bandwidth 1024
police cir 1024000 bc 128000 be 128000 conform-action transmit exceed-action drop
set dscp ef
class Web
priority 1024
set dscp af41
end
configure terminal
interface GigabitEthernet1
```

```
bandwidth 6144  
load-interval 30  
service-policy output QOS  
ip nbar protocol-discovery  
end  
copy running-config startup-config
```

CSR1000V4 - Cisco MPLS

```
enable  
configure terminal  
license boot level premium  
exit  
write  
reload  
enable  
configure terminal  
hostname CSR1000V4  
no ip domain lookup  
enable secret csr  
line vty 0 4  
password csr  
logging synchronous  
exec-timeout 0  
login  
exit  
interface GigabitEthernet1  
ip address 100.0.6.4 255.255.255.0  
no shutdown  
exit  
interface GigabitEthernet2  
ip address 192.16.20.4 255.255.255.0
```

```
no shutdown
exit
interface GigabitEthernet3
ip address 192.168.2.244 255.255.255.0
description TELNET -> Management Access
no shutdown
exit
router eigrp 100
eigrp log-neighbor-changes
no eigrp log-neighbor-warnings
network 192.16.20.4 0.0.0.255
network 100.0.6.4 0.0.0.255
passive-interface GigabitEthernet3
no auto-summary
end
copy running-config startup-config
configure terminal
class-map match-all FTP
match protocol ftp-data
class-map match-all Web
match protocol http
policy-map QOS
class FTP
bandwidth 1024
police cir 1024000 bc 128000 be 128000 conform-action transmit exceed-action drop
set dscp ef
class Web
priority 1024 128000
set dscp af41
end
```

```
configure terminal  
interface GigabitEthernet1  
bandwidth 6144  
load-interval 30  
service-policy output QOS  
ip nbar protocol-discovery  
end  
copy running-config startup-config
```

VM1 (OpenFlow1) - MPLS in Linux (QoSinLinux.sh)

```
sudo ifconfig eth0 192.16.10.10 netmask 255.255.255.0 up  
sudo ip route add 192.16.20.0/24 via 192.16.10.2  
sudo ip route add default via 192.16.10.2 dev eth0
```

VM2 (OpenFlow2) - MPLS in Linux (QoSinLinux.sh)

```
sudo ifconfig eth0 192.16.20.20 netmask 255.255.255.0 up  
sudo ip route add 192.16.10.0/24 via 192.16.20.4  
sudo ip route add default via 192.16.20.4 dev eth0
```

Dublin - MPLS in Linux

```
enable  
configure terminal  
hostname Dublin  
no ip domain lookup  
mpls label range 2000 2999  
mpls traffic-eng tunnels  
line console 0  
exec-timeout 0  
exit  
interface Loopback0  
ip address 1.1.1.1 255.255.255.255  
exit
```

```
interface FastEthernet0/0
ip address 200.0.2.3 255.255.255.0
ip nbar protocol-discovery
mpls ip
mpls traffic-eng tunnels
ip rsvp bandwidth 6144 6144
no shutdown
exit
interface FastEthernet0/1
ip address 200.0.6.3 255.255.255.0
ip nbar protocol-discovery
mpls ip
mpls traffic-eng tunnels
ip rsvp bandwidth 6144 6144
no shutdown
exit
router ospf 1
log-adjacency-changes
router-id 1.1.1.1
network 1.1.1.1 0.0.0.0 area 0
network 200.0.2.0 0.0.0.255 area 0
network 200.0.6.0 0.0.0.255 area 0
mpls traffic-eng router-id Loopback0
mpls traffic-eng area 0
end
copy running-config startup-config
```

CSR1000V1 - MPLS in Linux

```
enable
configure terminal
license boot level premium
```

```
exit
write
reload
enable
configure terminal
hostname CSR1000V1
no ip domain lookup
enable secret csr
line vty 0 4
password csr
logging synchronous
exec-timeout 0
login
exit
mpls label range 1000 1999
mpls traffic-eng tunnels
mpls traffic-eng reoptimize timers frequency 10
interface Loopback100
ip address 100.100.100.1 255.255.255.255
exit
interface GigabitEthernet1
ip address 100.0.2.3 255.255.255.0
mpls ip
mpls traffic-eng tunnels
ip rsvp bandwidth 6144 6144
no shutdown
exit
interface GigabitEthernet2
ip address 200.0.2.2 255.255.255.0
load-interval 30
```

```
ip nbar protocol-discovery
mpls ip
mpls traffic-eng tunnels
ip rsvp bandwidth 6144 6144
no shutdown
exit
interface GigabitEthernet3
ip address 192.168.2.241 255.255.255.0
description TELNET -> Management Access
no shutdown
exit
interface Tunnel1
ip unnumbered Loopback100
tunnel mode mpls traffic-eng
tunnel destination 100.100.100.2
tunnel mpls traffic-eng autoroute announce
tunnel mpls traffic-eng priority 1 1
tunnel mpls traffic-eng bandwidth 1024
tunnel mpls traffic-eng path-option 1 explicit name CSR1-Dublin-CSR2
exit
router ospf 1
log-adjacency-changes
passive-interface GigabitEthernet3
router-id 100.100.100.1
network 100.100.100.1 0.0.0.0 area 0
network 100.0.2.0 0.0.0.255 area 0
network 200.0.2.0 0.0.0.255 area 0
mpls traffic-eng router-id Loopback100
mpls traffic-eng area 0
exit
```

```
ip explicit-path name CSR1-Dublin-CSR2 enable  
next-address 200.0.2.3  
next-address 200.0.6.4  
end  
copy running-config startup-config
```

CSR1000V2 - MPLS in Linux

```
enable  
configure terminal  
license boot level premium  
exit  
write  
reload  
enable  
configure terminal  
hostname CSR1000V2  
no ip domain lookup  
enable secret csr  
line vty 0 4  
password csr  
logging synchronous  
exec-timeout 0  
login  
exit  
mpls label range 3000 3999  
mpls traffic-eng tunnels  
mpls traffic-eng reoptimize timers frequency 10  
interface Loopback100  
ip address 100.100.100.2 255.255.255.255  
exit  
interface GigabitEthernet1
```

```
ip address 100.0.6.3 255.255.255.0
mpls ip
mpls traffic-eng tunnels
ip rsvp bandwidth 6144 6144
no shutdown
exit
interface GigabitEthernet2
ip address 200.0.6.4 255.255.255.0
load-interval 30
ip nbar protocol-discovery
mpls ip
mpls traffic-eng tunnels
ip rsvp bandwidth 6144 6144
no shutdown
exit
interface GigabitEthernet3
description TELNET -> Management Access
ip address 192.168.2.242 255.255.255.0
no shutdown
exit
interface Tunnel1
ip unnumbered Loopback100
tunnel mode mpls traffic-eng
tunnel destination 100.100.100.1
tunnel mpls traffic-eng autoroute announce
tunnel mpls traffic-eng priority 1 1
tunnel mpls traffic-eng bandwidth 1024
tunnel mpls traffic-eng path-option 1 explicit name CSR2-Dublin-CSR1
exit
router ospf 1
```

```

log-adjacency-changes
passive-interface GigabitEthernet3
router-id 100.100.100.2
network 100.100.100.2 0.0.0.0 area 0
network 100.0.6.0 0.0.0.255 area 0
network 200.0.6.0 0.0.0.255 area 0
mpls traffic-eng router-id Loopback100
mpls traffic-eng area 0
exit
ip explicit-path name CSR2-Dublin-CSR1 enable
next-address 200.0.6.3
next-address 200.0.2.2
end
copy running-config startup-config

```

MPLS1 - MPLS in Linux (QoSInLinux.sh)

```

sudo modprobe mpls_router
sudo modprobe mpls_gso
sudo modprobe mpls_iptunnel
sudo sysctl -w net.mpls.conf.eth1.input=1
sudo sysctl -w net.mpls.platform_labels=1048575
sudo ifconfig eth0 192.16.10.2 netmask 255.255.255.0 up
sudo ifconfig eth1 100.0.2.2 netmask 255.255.255.0 up
sudo bash -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
sudo bash -c "echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter"
sudo ip link add name lo0 type dummy
sudo ip link set dev lo0 up
sudo ip addr add 4.4.4.4/32 dev lo0

```

```

frr version 3.1-dev-MyOwnFRRVersion
frr defaults traditional

```

```
hostname mpls1
no ipv6 forwarding
username cumulus nopassword
!
service integrated-vtysh-config
!
log syslog informational
!
router ospf
ospf router-id 4.4.4.4
log-adjacency-changes
network 4.4.4.4/32 area 0
network 100.0.2.0/24 area 0
network 192.16.10.0/24 area 0
!
mpls ldp
router-id 4.4.4.4
!
address-family ipv4
discovery transport-address 4.4.4.4
!
interface eth1
!
exit-address-family
!
mpls label global-block 4000 4999
!
line vty
!
```

MPLS2 - MPLS in Linux (QoSInLinux.sh)

```
sudo modprobe mpls_router
sudo modprobe mpls_gso
sudo modprobe mpls_iptunnel
sudo sysctl -w net.mpls.conf.eth1.input=1
sudo sysctl -w net.mpls.platform_labels=1048575
sudo ifconfig eth0 192.16.20.4 netmask 255.255.255.0 up
sudo ifconfig eth1 100.0.6.4 netmask 255.255.255.0 up
sudo bash -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
sudo bash -c "echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter"
sudo ip link add name lo0 type dummy
sudo ip link set dev lo0 up
sudo ip addr add 5.5.5.5/32 dev lo0
```

```
frr version 3.1-dev-MyOwnFRRVersion
```

```
frr defaults traditional
```

```
hostname mpls2
```

```
no ipv6 forwarding
```

```
username cumulus nopassword
```

```
!
```

```
service integrated-vtysh-config
```

```
!
```

```
log syslog informational
```

```
!
```

```
router ospf
```

```
ospf router-id 5.5.5.5
```

```
log-adjacency-changes
```

```
network 5.5.5.5/32 area 0
```

```
network 100.0.6.0/24 area 0
```

```
network 192.16.20.0/24 area 0
```

```
!
mpls ldp
router-id 5.5.5.5
!
address-family ipv4
discovery transport-address 5.5.5.5
!
interface eth1
!
exit-address-family
!
!
mpls label global-block 5000 5999
!
line vty
!
```

Appendix 8

OpenDaylight INSTALLATION

Download ODL VM =>

https://wiki.opendaylight.org/view/CrossProject:Integration_Group:Test_VMs

Convert OVA to VHD => <http://blog.worldofjani.com/?p=991>

Unpack "odl-test-desktop-4-disk1.vmdk"

Install Oracle VirtualBox => <https://www.virtualbox.org/wiki/Downloads>

Execute as Administrator => "c:\Program Files\Oracle\VirtualBox\VBoxManage.exe"

clonemedium --format vhd odl-test-desktop-4-disk1.vmdk OpenDaylight.vhd

Build VM with 2 GB of RAM, 1 vCPU and NAT or Bridged vNIC and use
"OpenDaylight.vhd"

Login with mininet/mininet

Configure network interfaces => sudo nano /etc/network/interfaces

```
auto eth0
```

```
iface eth0 inet static
    address 192.16.20.32
    netmask 255.255.255.0
    network 192.16.20.0
    broadcast 192.16.20.255
    gateway 192.16.20.1
    dns-nameservers 8.8.8.8
```

Now we can use SSH to connect via PuTTY on Port 22

Update Java JDK to v1.8:

```
sudo apt-get install software-properties-common -y
sudo add-apt-repository ppa:webupd8team/java -y
sudo apt-get update
sudo apt-get install oracle-java8-installer oracle-java8-set-default -y
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

Check Java version => java -version

Install latest ODL Carbon SR2 release:

```
wget
https://nexus.opendaylight.org/content/repositories/public/org/opendaylight/integration/distribution-karaf/0.6.2-Carbon/distribution-karaf-0.6.2-Carbon.zip
unzip distribution-karaf-0.6.2-Carbon.zip
cd distribution-karaf-0.6.2-Carbon
```

Run ODL => cd distribution-karaf-0.6.2-Carbon && bin/karaf

Adding Karaf features => feature:install odl-restconf odl-l2switch-switch-ui odl-mdsal-apidocs odl-dluxapps-applications odl-restconf-all odl-openflowplugin-flow-services-ui odl-l2switch-all odl-mdsal-all odl-yangtools-common

ODL Web GUI => 192.16.20.32:8181/index.html

Login with admin/admin

Install OFM:

```
sudo apt-get update
git clone https://github.com/CiscoDevNet/OpenDaylight-Openflow-App
```

```
sed -i 's/localhost/192.16.20.32/g' ./OpenDaylight-Openflow-  
App/ofm/src/common/config/env.module.js
```

```
sudo apt-get install nodejs-legacy  
sudo apt-get install npm  
sudo npm install -g grunt-cli  
cd OpenDaylight-Openflow-App
```

Run OFM => cd OpenDaylight-Openflow-App && grunt (in case already running run: fuser -n tcp -k 9000)

OFM Web GUI => 192.16.20.32:9000

Testing remote controller with Mininet: sudo mn --controller=remote,ip=192.16.20.32 --topo=linear,2 --switch=ovsk,protocols=OpenFlow13 --mac

Floodlight INSTALLATION

Download Floodlight VM => <http://opennetlinux.org/binaries/floodlight-vm.zip>

Convert OVA to VHD => <http://blog.worldofjani.com/?p=991>

Install Oracle VirtualBox => <https://www.virtualbox.org/wiki/Downloads>

Execute as Administrator => "c:\Program Files\Oracle\VirtualBox\VBoxManage.exe"
clonemedium --format vhd Floodlight-v1.1+Mininet.vmdk Floodlight.vhd

Build VM with 2 GB of RAM, 1 vCPU and NAT or Bridged vNIC and use "Floodlight.vhd"

Login with floodlight/floodlight

Configure network interfaces => sudo nano /etc/network/interfaces

```
auto eth0  
  
iface eth0 inet static  
  
    address 192.16.20.31  
    netmask 255.255.255.0  
    network 192.16.20.0  
    broadcast 192.16.20.255  
    gateway 192.16.20.1  
    dns-nameservers 8.8.8.8
```

Now we can use SSH to connect via PuTTY on Port 22

Update Java JDK to v1.8:

```
sudo apt-get install software-properties-common -y  
sudo add-apt-repository ppa:webupd8team/java -y  
sudo apt-get update  
sudo apt-get install oracle-java8-installer oracle-java8-set-default -y  
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

Check Java version => java -version

Install Floodlight master dependencies: sudo apt-get install build-essential ant maven python-dev

Install latest Floodlight master release v1.2:

```
mv floodlight floodlight-old  
git clone git://github.com/floodlight/floodlight.git  
cd floodlight  
git tag master  
git submodule init  
git submodule update  
ant  
sudo mkdir /var/lib/floodlight  
sudo chmod 777 /var/lib/floodlight
```

Run Floodlight => java -jar target/floodlight.jar

Web GUI => http://192.16.20.31:8080/ui/pages/index.html

Login with floodlight/floodlight

Testing remote controller with Mininet: sudo mn --
controller=remote,ip=192.16.20.31,port=6653 --topo=single,3 --
switch=ovsk,protocols=OpenFlow13 --mac

HPVANSNDN INSTALLATION

Download Ubuntu v14.04.5 LTS (Trusty Tahr) Server X64 =>
<http://releases.ubuntu.com/14.04.5/ubuntu-14.04.5-server-amd64.iso>

Build VM with 2 GB of RAM, 1 vCPU and NAT or Bridged vNIC, use minimum of 20 GB for VHD

Install OS with credentials as sdn/skyline and install OpenSSH Server

Configure network interfaces => sudo nano /etc/network/interfaces

```
auto eth0
iface eth0 inet static
    address 192.16.20.254
    netmask 255.255.255.0
    network 192.16.20.0
    broadcast 192.16.20.255
    gateway 192.16.20.1
    dns-nameservers 8.8.8.8
```

Download VAN SDN Controller v2.8.8.0366 ZIP =>

<https://h10145.www1.hp.com/downloads/SoftwareReleases.aspx?ProductNumber=J9863AA&E=&lang=&cc=&prodSeriesId=&SoftwareReleaseUId=10608&SerialNumber=&PurchaseDate=>

Download WinSCP v5.11.3 => <https://winscp.net/download/WinSCP-5.11.3-Setup.exe>

Install WinSCP and upload "hp-sdn-ctl-2.8.8.0366_amd64.deb" to VM via SFTP Port 22

Now we can use SSH to connect via PuTTY on Port 22

Install Aruba VAN SDN Controller v2.8 =>

https://support.hpe.com/hpsc/doc/public/display?docId=a00003658en_us

```
sudo apt-get update
sudo apt-get install python-software-properties
sudo apt-get install ubuntu-cloud-keyring
sudo add-apt-repository cloud-archive:juno
sudo apt-get update
sudo apt-get install keystone
touch /tmp/override.txt
sudo dpkg --unpack hp-sdn-ctl-2.8.8.0366_amd64.deb
sudo apt-get install -f
sudo dpkg -l hp-sdn-ctl
sudo service sdnc status
sudo /opt/sdn/admin/config_local_keystone
```

Get license => http://h20628.www2.hp.com/km-ext/kmcstdirect/emr_na-a00003662en_us-1.pdf

Install license =>

<https://community.arubanetworks.com/aruba/attachments/aruba/SDN/11/1/VAN%20SDN%20Licensing%20with%20Demo.pdf>

Web GUI => <https://192.16.20.254:8443/sdn/ui/app/index>

Login with sdn/skyline

Download Northbound Networks Flow Maker v1.1.1 =>

<https://northboundnetworks.com/products/flow-maker-deluxe>

Install application by going to => Applications->New->Select "flow-maker.zip"->Deploy

Testing remote controller with Mininet: sudo mn --controller=remote,ip=192.16.20.254 --topo=tree,depth=3,fanout=2 --switch=ovsk,protocols=OpenFlow10 --mac

ONOS INSTALLATION

Download Ubuntu v16.04.3 LTS (Xenial Xerus) Desktop X64 =>

<http://releases.ubuntu.com/xenial/ubuntu-16.04.3-desktop-amd64.iso>

Build VM with 2 GB of RAM, 1 vCPU and NAT or Bridged vNIC, use minimum of 20 GB for VHD

Install OS with credentials as onos/onos

Configure network interfaces => sudo nano /etc/network/interfaces

```
auto eth0
iface eth0 inet static
    address 192.16.20.33
    netmask 255.255.255.0
    network 192.16.20.0
    broadcast 192.16.20.255
    gateway 192.16.20.1
    dns-nameservers 8.8.8.8
```

Install SSH Server => sudo apt install openssh-server

Now we can use SSH to connect via PuTTY on Port 22

Requirements for ONOS => <https://wiki.onosproject.org/display/ONOS/Requirements>

```

sudo apt-get install software-properties-common -y && \
sudo add-apt-repository ppa:webupd8team/java -y && \
sudo apt-get update && \
echo "oracle-java8-installer shared/accepted-oracle-license-v1-1 select true" | \
sudo debconf-set-selections && \
sudo apt-get install oracle-java8-installer oracle-java8-set-default -y
sudo apt-get install curl

```

Install latest ONOS Magpie release v1.12.0 as Single Machine =>

<https://wiki.onosproject.org/display/ONOS/Installing+on+a+single+machine>

```

sudo mkdir /opt
cd /opt
sudo wget -c http://downloads.onosproject.org/release/onos-1.12.0.tar.gz
sudo tar xzf onos-1.12.0.tar.gz
sudo mv onos-1.12.0 onos
/opt/onos/bin/onos-service start

```

Web GUI => <http://192.16.20.33:8181/onos/ui/index.html>

Login with onos/rocks

Configure ONOS as Service =>

<https://wiki.onosproject.org/display/ONOS/Running+ONOS+as+a+service>

```

sudo cp /opt/onos/init/onos.initd /etc/init.d/onos
sudo cp /opt/onos/init/onos.conf /etc/init/onos.conf
sudo cp /opt/onos/init/onos.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable onos
sudo reboot -h now
sudo service onos {start|stop|status} or sudo systemctl {start|stop|status|restart}
onos.service

```

Configure ONOS Applications => sudo nano /opt/onos/options

ONOS_USER=root

ONOS_APPS=openflow-

message,drivers,hostprovider,mobility,lldpprovider,linkdiscovery,ofagent,openflow-base,openflow,optical-model,proxyarp,fwd

Change file permissions => sudo chmod 777 /opt/onos/options

Testing remote controller with Mininet: sudo mn --controller=remote,ip=192.16.20.33 --topo=linear,3 --switch=ovsk,protocols=OpenFlow13 --mac

POX INSTALLATION

Download All-in-one SDN =>

http://yuba.stanford.edu/~srini/tutorial/SDN_tutorial_VM_64bit.ova

Convert OVA to VHD => <http://blog.worldofjani.com/?p=991>

Unpack "SDN Hub tutorial VM 64-bit with Docker-disk1.vmdk"

Install Oracle VirtualBox => <https://www.virtualbox.org/wiki/Downloads>

Execute as Administrator => "c:\Program Files\Oracle\VirtualBox\VBoxManage.exe" clonemedium --format vhd "SDN Hub tutorial VM 64-bit with Docker-disk1.vmdk" POX.vhd

Build VM with 2 GB of RAM, 1 vCPU and NAT or Bridged vNIC and use "POX.vhd"

Login with ubuntu/ubuntu

Configure network interfaces => sudo nano /etc/network/interfaces

```
auto eth0
iface eth0 inet static
    address 192.16.20.34
    netmask 255.255.255.0
    network 192.16.20.0
    broadcast 192.16.20.255
    gateway 192.16.20.1
    dns-nameservers 8.8.8.8
```

Now we can use SSH to connect via PuTTY on Port 22

Run POX Eel release v0.5.0 with Stock Components =>

<https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-StockComponents> (in case already running run: fuser -n tcp -k 6633)

L2 Learning Switch: cd /home/ubuntu/pox && ./pox.py log.level --DEBUG
samples.pretty_log forwarding.l2_learning

```
Mininet Single Topo: sudo mn --controller=remote,ip=192.16.20.34 --topo=single,2 --
switch=ovsk,protocols=OpenFlow10 --mac
```

```
STP & LLDP: cd /home/ubuntu/pox && ./pox.py log.level --DEBUG
samples.pretty_log forwarding.l2_multi openflow.discovery openflow.spanning_tree -
-no-flood --hold-down
```

```
Mininet Torus Topo: sudo mn --controller=remote,ip=192.16.20.34 --
ipbase=192.16.20.128/25 --topo=torus,4,4 --switch=ovsk,protocols=OpenFlow10 --
mac
```

Ryu INSTALLATION

Download Ubuntu v14.04.5 LTS (Trusty Tahr) Server X64 =>
<http://releases.ubuntu.com/14.04.5/ubuntu-14.04.5-server-amd64.iso>

Build VM with 2 GB of RAM, 1 vCPU and NAT or Bridged vNIC, use minimum of 20 GB for VHD

Install OS with credentials as ryu/ryu and install OpenSSH Server

Configure network interfaces => sudo nano /etc/network/interfaces

```
auto eth0
iface eth0 inet static
    address 192.16.20.35
    netmask 255.255.255.0
    network 192.16.20.0
    broadcast 192.16.20.255
    gateway 192.16.20.1
    dns-nameservers 8.8.8.8
```

Install Python Prerequisites =>

```
sudo apt-get update
sudo apt-get install python-pip
sudo apt-get install python-dev
sudo apt-get install python-setuptools
sudo apt-get install python-eventlet
sudo apt-get install python-webob
```

```
sudo apt-get install python-paramiko
```

Install latest Ryu v4.20 => <https://osrg.github.io/ryu/>

```
sudo apt-get install git
```

```
git clone git://github.com/osrg/ryu.git
```

```
cd ryu
```

```
git checkout v4.20
```

```
sudo pip install .
```

```
sudo pip install six --upgrade
```

Run Ryu v4.20 with Apps => <https://osrg.github.io/ryu-book/en/html/>

Switching Hub OF10 => cd /usr/local/lib/python2.7/dist-packages/ryu/app &&
sudo ryu run gui_topology/gui_topology.py simple_switch.py --observe-links

Mininet Linear Topo: sudo mn --controller=remote,ip=192.16.20.35 --
topo=linear,5 --switch=ovsk,protocols=OpenFlow10 --mac

Switching Hub OF13 => cd /usr/local/lib/python2.7/dist-packages/ryu/app &&
sudo ryu run gui_topology/gui_topology.py simple_switch_13.py --observe-links

Mininet Linear Topo: sudo mn --controller=remote,ip=192.16.20.35 --
topo=linear,5 --switch=ovsk,protocols=OpenFlow13 --mac

Switching Hub OF14 => cd /usr/local/lib/python2.7/dist-packages/ryu/app &&
sudo ryu run gui_topology/gui_topology.py simple_switch_14.py --observe-links

Mininet Linear Topo: sudo mn --controller=remote,ip=192.16.20.35 --
topo=linear,5 --switch=ovsk,protocols=OpenFlow13 --mac

Switching Hub OF15 => cd /usr/local/lib/python2.7/dist-packages/ryu/app &&
sudo ryu run simple_switch_15.py

Mininet Linear Topo: sudo mn --controller=remote,ip=192.16.20.35 --
topo=linear,5 --switch=ovsk,protocols=OpenFlow13 --mac

Ryu Topology Viewer => <http://192.16.20.35:8080>

Datacentre Topo - Ryu

Run Ryu with Spanning Tree OF13: cd /usr/local/lib/python2.7/dist-packages/ryu/app &&
sudo ryu run simple_switch_stp_13.py

Run Mininet wtih Datacentre Topo: sudo mn --custom ./Downloads/DCTopo.py --topo
DCTopo,4,4 --controller=remote,ip=192.16.20.35 --switch=ovsk,protocols=OpenFlow13 --

mac

```
from mininet.util import irange
from mininet.topo import Topo

class DCTopo( Topo ):
    "Datacentre Topology"

    def build( self, Racks=4, Hosts=4, Switches=2 ):
        if Switches >= 16:
            raise Exception( "Please use less than 16 switches" )

        self.racks = []
        coreSwitches = []
        lastSwitch = None

        #links all the switches
        for i in irange( 1, Switches ):
            coreSwitch = self.addSwitch( 's%s' % i )
            coreSwitches.append( coreSwitch )
            if lastSwitch:
                self.addLink( lastSwitch, coreSwitch )

            lastSwitch = coreSwitch

        #links last switch to the first switch
        if Switches > 1:
            self.addLink( lastSwitch, coreSwitches[0] )

        for i in irange( 1, Racks ):
```

```

rack = self.buildRack( i, Hosts )
self.racks.append( rack )
for switch in rack:
    for coreSwitch in coreSwitches:
        self.addLink( coreSwitch, switch )

def buildRack( self, loc, Hosts ):
    "Build a rack with a core switch"

    dpid = ( loc * 16 ) + 1
    switch = self.addSwitch( 's1r%s' % loc, dpid='%x' % dpid )

    for n in xrange( 1, Hosts ):
        host = self.addHost( 'h%sr%s' % ( n, loc ) )
        self.addLink( switch, host )

    return [switch]

#Import using `mn --custom <filename> --topo DCTopo`
topos = {
    'DCTopo': DCTopo
}

```

Mininet Bridge to Physical Interface

Edit Mininet VM Interfaces => sudo nano /etc/network/interfaces

```

auto eth1
iface eth1 inet static
    address 192.168.2.246
    netmask 255.255.255.0
    network 192.168.2.0
    broadcast 192.168.2.255

```

gateway 192.168.2.1

dns-nameservers 8.8.8.8

sudo reboot -h now

Flush Routing Table => sudo /sbin/ip route flush table main

Check Routing Table => route -n

Add Routes =>

sudo ip route add 192.16.20.0/24 via 0.0.0.0 dev eth0

sudo ip route add 192.168.2.0/24 via 0.0.0.0 dev eth1

Check Connectivity =>

ping 192.16.20.10 (VM1)

ping 192.168.2.13 (Hyper-V Host)

Run the Topology => sudo mn --controller=remote,ip=192.16.20.254 --topo=single,2 --switch=ovsk,protocols=OpenFlow10 --mac --ipbase=192.16.20.0/24

Add eth1 Port to OVS => sudo ovs-vsctl add-port s1 eth1

Remove Route to 192.16.20.0/24 => sudo route del -net 192.168.2.0 netmask 255.255.255.0
eth1

Check the OVS => sudo ovs-vsctl show

Start Web Server on Host2 => h2 python -m SimpleHTTPServer 80 &

Change IP Address for Host2 => h2 ifconfig h2-eth0 192.168.2.200/24 up

Check Routing Table on Host2 => h2 route -n

Check Connectivity to Hyper-V Host => h2 ping 192.168.2.13

Appendix 9

PER FLOW

Ryu (1) =>

sudo mn --mac --switch=ovsk --ipbase=192.16.20.0/24 --controller=remote

xterm s1

sudo su

ovs-vsctl set bridge s1 protocols=OpenFlow13

ovs-vsctl set-manager ptcp:6632

Ryu (2) =>

```

sudo su

sed '/OFPFlowMod(/,/s/)/,table_id=1)/* ryu/ryu/app/simple_switch_13.py >
ryu/ryu/app/qos_simple_switch_13.py

cd ryu/; python ./setup.py install

ryu-manager --observe-links ryu.app.rest_topology ryu.app.ws_topology
ryu.app.gui_topology.gui_topology ryu.app.ofctl_rest ryu.app.rest_qos
ryu.app.qos_simple_switch_13 ryu.app.rest_conf_switch

Ryu (3) =>

sudo su

curl -g -X PUT -d ""tcp:127.0.0.1:6632"""

http://127.0.0.1:8080/v1.0/conf/switches/0000000000000001/ovsdb_addr

ovs-vsctl list qos (shows qos table)

ovs-vsctl list queue (shows queue table)

ovs-vsctl --all destroy qos (destroys qos table)

ovs-vsctl --all destroy queue (destroys queue table)

tc qdisc show dev s1-eth2 (shows traffic control qdisc classless)

tc class show dev s1-eth2 (shows traffic control classes)

tc qdisc del dev s1-eth2 root (deletes traffic control queue disciplines)

curl -g -X POST -d '{"port_name": "s1-eth2", "type": "linux-htb", "max_rate": "10000000",
"queues": [{"max_rate": "1000000"}, {"min_rate": "500000"}]}'

http://127.0.0.1:8080/qos/queue/0000000000000001 [{"switch_id":
"0000000000000001", "command_result": {"result": "success", "details": {"0": {"config": {"max-rate": "1000000"}, "1": {"config": {"min-rate": "500000"}}}}}}

curl -g -X GET http://127.0.0.1:8080/qos/queue/0000000000000001 (gets queue)

ovs-ofctl -O OpenFlow13 dump-flows s1 (dumps flows)

curl -g -X POST -d '{"match": {"nw_dst": "192.16.20.2", "nw_proto": "UDP", "tp_dst": "5000"}, "actions": {"queue": "1"} }' http://127.0.0.1:8080/qos/rules/0000000000000001
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "QoS added. : qos_id=1"}]}]

curl -g -X POST -d '{"match": {"nw_dst": "192.16.20.2", "nw_proto": "TCP", "tp_dst": "21"}, "actions": {"queue": "0"} }' http://127.0.0.1:8080/qos/rules/0000000000000001

```

```
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "QoS added. : qos_id=0"}]}]
```

```
curl -g -X POST -d '{"match": {"nw_dst": "192.16.20.2", "nw_proto": "TCP", "tp_dst": "80"}, "actions": {"queue": "1"} }' http://127.0.0.1:8080/qos/rules/0000000000000001
```

```
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "QoS added. : qos_id=1"}]}]
```

curl -g -X GET http://127.0.0.1:8080/qos/rules/0000000000000001 (gets rules)

curl -g -X DELETE http://127.0.0.1:8080/qos/queue/0000000000000001 (deletes queue)

curl -g -X DELETE -d '{"qos_id": "1"}'

http://127.0.0.1:8080/qos/rules/0000000000000001 (deletes specific qos_id)

curl -g -X DELETE -d '{"qos_id": "all"}'

http://127.0.0.1:8080/qos/rules/0000000000000001 (deletes specific all qos_id)

ovs-ofctl -O OpenFlow13 dump-flows s1 (dumps flows)

Ryu (1) =>

xterm h2

```
iperf -s -u -i 1 -p 5000
```

xterm h1

```
iperf -c 192.16.20.2 -p 5000 -u -t -1
```

xterm h2

```
iperf -s -i 1 -p 21
```

xterm h1

```
iperf -c 192.16.20.2 -p 21 -t -1
```

xterm h2

```
iperf -s -u -i 1 -p 80
```

xterm h1

```
iperf -c 192.16.20.2 -p 80 -t -1
```

PER CLASS WITH DSCP

Ryu (1) =>

```
sudo mn --topo linear,2 --mac --switch=ovsk --controller=remote
```

xterm s1

```

sudo su

ovs-vsctl set bridge s1 protocols=OpenFlow13

xterm s2

sudo su

ovs-vsctl set bridge s2 protocols=OpenFlow13

ovs-vsctl set-manager ptcp:6632

xterm h1

ip addr del 10.0.0.1/8 dev h1-eth0

ip addr add 192.16.10.10/24 dev h1-eth0

ip route add default via 192.16.10.1

xterm h2

ip addr del 10.0.0.2/8 dev h2-eth0

ip addr add 192.16.20.20/24 dev h2-eth0

ip route add default via 192.16.20.1

Ryu (2) =>

sudo su

sed '/OFPFlowMod(.,)/s/0, cmd/1, cmd/' ryu/ryu/app/rest_router.py >
ryu/ryu/app/qos_rest_router.py

cd ryu/; python ./setup.py instal

ryu-manager ryu.app.rest_qos ryu.app.qos_rest_router ryu.app.rest_conf_switch

Ryu (3) =>

sudo su

curl -g -X PUT -d ""tcp:127.0.0.1:6632"""

http://127.0.0.1:8080/v1.0/conf/switches/0000000000000002/ovsdb_addr

ovs-vsctl list qos (shows qos table)

ovs-vsctl list queue (shows queue table)

ovs-vsctl --all destroy qos (destroys qos table)

ovs-vsctl --all destroy queue destroys queue table)

tc qdisc show dev s2-eth1 (shows traffic control qdisc classless)

tc class show dev s2-eth1 (shows traffic control classes)

```

tc qdisc del dev s2-eth1 root (deletes traffic control queue disciplines)

```
curl -g -X POST -d '{"port_name": "s2-eth1", "type": "linux-htb", "max_rate": "1000000", "queues": [{"max_rate": "1000000"}, {"min_rate": "300000"}, {"min_rate": "500000"}]}' http://127.0.0.1:8080/qos/queue/0000000000000002 [{"switch_id": "0000000000000001", "command_result": {"result": "success", "details": {"0": {"config": {"max-rate": "1000000"}}, "1": {"config": {"min-rate": "300000"}}, "2": {"config": {"min-rate": "500000"}}}}}]
```

curl -g -X GET http://127.0.0.1:8080/qos/queue/0000000000000002 (gets S2 queue)

```
curl -g -X POST -d '{"address": "192.16.10.1/24"}'
```

```
http://127.0.0.1:8080/router/0000000000000001
```

```
curl -g -X POST -d '{"address": "192.16.30.1/24"}'
```

```
http://127.0.0.1:8080/router/0000000000000001
```

```
curl -g -X POST -d '{"gateway": "192.16.30.10"}'
```

```
http://127.0.0.1:8080/router/0000000000000001
```

```
curl -g -X POST -d '{"address": "192.16.20.1/24"}'
```

```
http://127.0.0.1:8080/router/0000000000000002
```

```
curl -g -X POST -d '{"address": "192.16.30.10/24"}'
```

```
http://127.0.0.1:8080/router/0000000000000002
```

```
curl -g -X POST -d '{"gateway": "192.16.30.1"}'
```

```
http://127.0.0.1:8080/router/0000000000000002
```

curl -g -X GET http://127.0.0.1:8080/router/0000000000000001 (gets S1 routes)

curl -g -X GET http://127.0.0.1:8080/router/0000000000000002 (gets S2 routes)

ovs-ofctl -O OpenFlow13 dump-flows s1 (dumps S1 flows)

ovs-ofctl -O OpenFlow13 dump-flows s2 (dumps S2 flows)

```
curl -g -X POST -d '{"match": {"nw_dst": "192.16.10.10", "nw_proto": "UDP", "tp_dst": "21"}, "actions": {"mark": "18"} }' http://127.0.0.1:8080/qos/rules/0000000000000001 [{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": {"QoS added. : qos_id=1"}}}]}
```

```
curl -g -X POST -d '{"match": {"nw_dst": "192.16.10.10", "nw_proto": "UDP", "tp_dst": "80"}, "actions": {"mark": "36"} }' http://127.0.0.1:8080/qos/rules/0000000000000001 [{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": {"QoS added. : qos_id=2"}}}]}
```

```
curl -g -X POST -d '{"match": {"nw_dst": "192.16.10.10", "nw_proto": "UDP", "tp_dst": "5000"}, "actions": {"mark": "48"} }' http://127.0.0.1:8080/qos/rules/00000000000000000000000000000001  
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": ["QoS added. : qos_id=3"]}]}
```

curl -g -X GET http://127.0.0.1:8080/qos/rules/00000000000000000000000000000001 (gets S1 rules)

ovs-ofctl -O OpenFlow13 dump-flows s1 (dumps S1 flows)

ovs-vsctl list port (shows ports)

ovs-vsctl list qos (shows qos table)

ovs-vsctl list Queue (shows queue table)

```
curl -g -X POST -d '{"match": {"ip_dscp": "48"}, "actions": {"queue": "0"} }'  
http://127.0.0.1:8080/qos/rules/00000000000000000000000000000002
```

```
curl -g -X POST -d '{"match": {"ip_dscp": "18"}, "actions": {"queue": "1"} }'  
http://127.0.0.1:8080/qos/rules/00000000000000000000000000000002
```

```
curl -g -X POST -d '{"match": {"ip_dscp": "36"}, "actions": {"queue": "2"} }'  
http://127.0.0.1:8080/qos/rules/00000000000000000000000000000002
```

curl -g -X GET http://127.0.0.1:8080/qos/rules/00000000000000000000000000000002 (gets S2 rules)

ovs-ofctl -O OpenFlow13 dump-flows s2 (dumps S2 flows)

Ryu (1) =>

xterm h2

```
iperf -s -u -i 1 -p 5000
```

xterm h1

```
iperf -c 192.16.20.20 -p 5000 -u -t -1 (-b 1M: to limit UDP bandwidth)
```

xterm h2

```
iperf -s -u -i 1 -p 21
```

xterm h1

```
iperf -c 192.16.20.20 -p 21 -u -t -1 (-b 300k: to limit UDP bandwidth)
```

xterm h2

```
iperf -s -u -i 1 -p 80
```

xterm h1

```
iperf -c 192.16.20.20 -p 80 -u -t -1 (-b 500k: to limit UDP bandwidth)
```

```

curl -g -X DELETE http://127.0.0.1:8080/qos/queue/0000000000000002 (deletes S2 queue)
curl -g -X POST -d '{"port_name": "s2-eth1", "type": "linux-htb", "max_rate":"1000000",
"queues": [{"max_rate": "1000000"}, {"min_rate": "300000"}, {"min_rate":"500000"}]}'
http://localhost:8080/qos/queue/0000000000000002 (creates S2 queue)
curl -g -X DELETE -d '{"qos_id":"1"}' http://127.0.0.1:8080/qos/rules/0000000000000002
(deletes S2 rule id=1)
curl -g -X DELETE -d '{"qos_id":"all"}' http://127.0.0.1:8080/qos/rules/0000000000000002
(deletes S2 all rules)

```

UPGRADE OVS SCRIPT

```

#!/bin/sh -ev

# Reference: https://github.com/mininet/mininet/wiki/Installing-new-version-of-Open-
vSwitch

# How to test: ovs-vsctl -V

# Check permission
test $(id -u) -ne 0 && echo "This script must be run as root" && exit 0

#Remove old version ovs
aptitude remove openvswitch-common openvswitch-datapath-dkms openvswitch-controller
openvswitch-pki openvswitch-switch -y

#Install new version ovs
cd /tmp
wget http://openvswitch.org/releases/openvswitch-2.8.1.tar.gz
tar zxvf openvswitch-2.8.1.tar.gz
cd openvswitch-2.8.1
./configure --prefix=/usr --with-linux=/lib/modules/`uname -r`/build
make
make install

```

```
make modules_install  
rmmod openvswitch  
depmod -a  
  
# Say goodbye to openvswitch-controller  
/etc/init.d/openvswitch-controller stop  
update-rc.d openvswitch-controller disable
```

```
#Start new version ovs  
/usr/share/openvswitch/scripts/ovs-ctl start  
/etc/init.d/openvswitch-switch start  
  
#Clean ovs  
rm -rf /tmp/openvswitch-2.8.1*
```

METER WITH OVS

```
Ryu (1) =>  
sudo gedit ./ryu/ryu/app/qos_sample_topology2.py
```

```
from mininet.net import Mininet  
from mininet.cli import CLI  
from mininet.topo import Topo  
from mininet.node import OVSSwitch  
from mininet.node import RemoteController
```

```
class MyTopo(Topo):  
    def __init__( self ):  
        "Create custom topo."  
        # Initialize topology  
        Topo.__init__( self )
```

```

# Add hosts and switches

host01 = self.addHost('h1')
host02 = self.addHost('h2')
host03 = self.addHost('h3')

switch01 = self.addSwitch('s1')
switch02 = self.addSwitch('s2')
switch03 = self.addSwitch('s3')

# Add links

self.addLink(host01, switch01)
self.addLink(host02, switch02)
self.addLink(host03, switch03)

self.addLink(switch01, switch02)
self.addLink(switch01, switch03)

def genericTest(topo):

    net = Mininet(topo=topo, switch=OVSSwitch, controller=RemoteController)

    net.start()

    CLI(net)

    net.stop()

def main():

    topo = MyTopo()

    genericTest(topo)

if __name__ == '__main__':
    main()

sudo python ./ryu/ryu/app/qos_sample_topology2.py
Ryu (2) =>
sudo su

```

```

ovs-vsctl set bridge s1 protocols=OpenFlow13
ovs-vsctl set bridge s2 protocols=OpenFlow13
ovs-vsctl set bridge s3 protocols=OpenFlow13
ovs-vsctl set-manager ptcp:6632
sed '/OFPFlowMod(.,/s/),table_id=1)/' ryu/ryu/app/simple_switch_13.py >
ryu/ryu/app/qos_simple_switch_13.py
cd ryu/; python ./setup.py install
ryu-manager ryu.app.rest_qos ryu.app.rest_conf_switch ryu.app.qos_simple_switch_13
Ryu (3) =>
curl -g -X PUT -d ""tcp:127.0.0.1:6632"""
http://127.0.0.1:8080/v1.0/conf/switches/0000000000000001/ovsdb_addr
curl -g -X PUT -d ""tcp:127.0.0.1:6632"""
http://127.0.0.1:8080/v1.0/conf/switches/0000000000000002/ovsdb_addr
curl -g -X PUT -d ""tcp:127.0.0.1:6632"""
http://127.0.0.1:8080/v1.0/conf/switches/0000000000000003/ovsdb_addr
curl -g -X POST -d '{"port_name": "s1-eth1", "type": "linux-htb", "max_rate": "1000000",
"queues": [{"min_rate": "100000"}, {"min_rate": "300000"}, {"min_rate": "600000"}]}'
http://127.0.0.1:8080/qos/queue/0000000000000001
curl -g -X POST -d '{"match": {"ip_dscp": "0", "in_port": "2"}, "actions": {"queue": "1"}}'
http://127.0.0.1:8080/qos/rules/0000000000000001
curl -g -X POST -d '{"match": {"ip_dscp": "18", "in_port": "2"}, "actions": {"queue": "3"}}'
http://127.0.0.1:8080/qos/rules/0000000000000001
curl -g -X POST -d '{"match": {"ip_dscp": "36", "in_port": "2"}, "actions": {"queue": "2"}}'
http://127.0.0.1:8080/qos/rules/0000000000000001
curl -g -X POST -d '{"match": {"ip_dscp": "0", "in_port": "3"}, "actions": {"queue": "1"}}'
http://127.0.0.1:8080/qos/rules/0000000000000001
curl -g -X POST -d '{"match": {"ip_dscp": "18", "in_port": "3"}, "actions": {"queue": "3"}}'
http://127.0.0.1:8080/qos/rules/0000000000000001
curl -g -X POST -d '{"match": {"ip_dscp": "36", "in_port": "3"}, "actions": {"queue": "2"}}'
http://127.0.0.1:8080/qos/rules/0000000000000001
curl -g -X GET http://127.0.0.1:8080/qos/rules/0000000000000001 (gets S1 rules)

```

```

curl -g -X POST -d '{"match": {"ip_dscp": "18"}, "actions": {"meter": "1"} }'
http://127.0.0.1:8080/qos/rules/00000000000000000002

curl -g -X POST -d '{"meter_id": "1", "flags": "KBPS",
"bands": [{"type": "DSCP_REMARK", "rate": "400", "prec_level": "1"}]}'
http://127.0.0.1:8080/qos/meter/00000000000000000002

curl -g -X POST -d '{"match": {"ip_dscp": "18"}, "actions": {"meter": "1"} }'
http://127.0.0.1:8080/qos/rules/00000000000000000003

curl -g -X POST -d '{"meter_id": "1", "flags": "KBPS",
"bands": [{"type": "DSCP_REMARK", "rate": "400", "prec_level": "1"}]}'
http://127.0.0.1:8080/qos/meter/00000000000000000003

curl -g -X GET http://127.0.0.1:8080/qos/rules/00000000000000000002 (gets S2 rules)
curl -g -X GET http://127.0.0.1:8080/qos/rules/00000000000000000003 (gets S3 rules)
curl -g -X GET http://127.0.0.1:8080/qos/meter/00000000000000000002 (gets S2 meter)
curl -g -X GET http://127.0.0.1:8080/qos/meter/00000000000000000003 (gets S3 meter)

ovs-ofctl -O OpenFlow13 add-meter s2
meter=1,kbps,stats,bands=type=dscp_remark,rate=400,prec_level=1 (adds S2 meter)
ovs-ofctl -O OpenFlow13 add-meter s3
meter=1,kbps,stats,bands=type=dscp_remark,rate=400,prec_level=1 (adds S3 meter)
ovs-ofctl -O OpenFlow13 add-flow s2 in_port=1,actions=meter:1,output:2 (adds S2 flow)
ovs-ofctl -O OpenFlow13 add-flow s3 in_port=1,actions=meter:1,output:3 (adds S3 flow)

curl -g -X DELETE -d '{"qos_id": "all"}'
http://127.0.0.1:8080/qos/rules/00000000000000000002 (deletes S2 all rules)
curl -g -X DELETE -d '{"qos_id": "all"}'
http://127.0.0.1:8080/qos/rules/00000000000000000003 (deletes S3 all rules)
curl -g -X DELETE -d '{"meter_id": "1"}'
http://127.0.0.1:8080/qos/meter/00000000000000000002 (deletes S2 meter id=1)
curl -g -X DELETE -d '{"meter_id": "1"}'
http://127.0.0.1:8080/qos/meter/00000000000000000003 (deletes S2 meter id=1)

Ryu (1) =>
xterm h1
iperf -s -u -p 5001 -i 1

```

```

xterm h1
iperf -s -u -p 5002 -i 1

xterm h1
iperf -s -u -p 5003 -i 1

xterm h2
iperf -c 10.0.0.1 -p 5001 -u -b 600k -t -1

xterm h2
iperf -c 10.0.0.1 -p 5002 -u -b 600k --tos 0x90 -t -1

xterm h3
iperf -c 10.0.0.1 -p 5003 -u -b 800k -t -1

```

METER TABLE WITH OFSOFTSWITCH13

Ryu (1) =>

```

sudo apt-get install -y git-core autoconf automake autotools-dev pkg-config make gcc g++
libtool libc6-dev cmake libpcap-dev libxerces-c2-dev unzip libpcre3-dev flex bison libboost-
dev gedit curl

wget -nc http://de.archive.ubuntu.com/ubuntu/pool/main/b/bison/bison_2.5.dfsg-
2.1_amd64.deb http://de.archive.ubuntu.com/ubuntu/pool/main/b/bison/libbison-
dev_2.5.dfsg-2.1_amd64.deb

sudo dpkg -i bison_2.5.dfsg-2.1_amd64.deb libbison-dev_2.5.dfsg-2.1_amd64.deb

git clone https://github.com/netgroup-polito/netbee.git

cd netbee/src

cmake .

make

sudo cp ../*.so /usr/local/lib

sudo ldconfig

sudo cp -R ./include/* /usr/include/

cd ../../

git clone https://github.com/CPqD/ofsoftswitch13.git

cd ofsoftswitch13

git checkout d174464dcc414510990e38426e2e274a25330902

```

```
./boot.sh  
./configure  
make  
sudo make install  
cd ..  
sudo gedit ofsoftswitch13/lib/netdev.c  
netdev->speed = ecmd.speed;  
netdev->speed = 1; /* Fix to 1Mbps link */  
cd ofsoftswitch13  
make clean  
./boot.sh  
./configure  
make  
sudo make install
```

```
sudo gedit ./ryu/ryu/app/qos_sample_topology.py
```

```
from mininet.net import Mininet  
from mininet.cli import CLI  
from mininet.topo import Topo  
from mininet.node import UserSwitch  
from mininet.node import RemoteController
```

```
class SliceableSwitch(UserSwitch):  
    def __init__(self, name, **kwargs):  
        UserSwitch.__init__(self, name, ", **kwargs)
```

```
class MyTopo(Topo):  
    def __init__( self ):
```

```

"Create custom topo."

# Initialize topology
Topo.__init__( self )

# Add hosts and switches
host01 = self.addHost('h1')
host02 = self.addHost('h2')
host03 = self.addHost('h3')
switch01 = self.addSwitch('s1')
switch02 = self.addSwitch('s2')
switch03 = self.addSwitch('s3')

# Add links
self.addLink(host01, switch01)
self.addLink(host02, switch02)
self.addLink(host03, switch03)
self.addLink(switch01, switch02)
self.addLink(switch01, switch03)

def run(net):
    s1 = net.getNodeByName('s1')
    s1.cmdPrint('dpctl unix:/tmp/s1 queue-mod 1 1 100')
    s1.cmdPrint('dpctl unix:/tmp/s1 queue-mod 1 2 300')
    s1.cmdPrint('dpctl unix:/tmp/s1 queue-mod 1 3 600')

def genericTest(topo):
    net = Mininet(topo=topo, switch=SliceableSwitch,
                  controller=RemoteController)
    net.start()
    run(net)
    CLI(net)
    net.stop()

```

```

def main():
    topo = MyTopo()
    genericTest(topo)

if __name__ == '__main__':
    main()

sudo python ./ryu/ryu/app/qos_sample_topology.py
Ryu (2) =>
sudo su
sed '/OFPFlowMod(/,/s/)/,table_id=1)/* ryu/ryu/app/simple_switch_13.py >
ryu/ryu/app/qos_simple_switch_13.py
cd ryu/; python ./setup.py install
ryu-manager ryu.app.rest_qos ryu.app.rest_conf_switch ryu.app.qos_simple_switch_13
Ryu (3) =>
sudo su
curl -g -X POST -d '{"match": {"ip_dscp": "0", "in_port": "2"}, "actions": {"queue": "1"}}'
http://127.0.0.1:8080/qos/rules/0000000000000001 [{"switch_id":
"0000000000000001", "command_result": [{"result": "success", "details": "QoS added. :
qos_id=1"}]}]
curl -g -X POST -d '{"match": {"ip_dscp": "18", "in_port": "2"}, "actions": {"queue": "3"}}'
http://127.0.0.1:8080/qos/rules/0000000000000001 [{"switch_id":
"0000000000000001", "command_result": [{"result": "success", "details": "QoS added. :
qos_id=2"}]}]
curl -g -X POST -d '{"match": {"ip_dscp": "36", "in_port": "2"}, "actions": {"queue": "2"}}'
http://127.0.0.1:8080/qos/rules/0000000000000001 [{"switch_id":
"0000000000000001", "command_result": [{"result": "success", "details": "QoS added. :
qos_id=3"}]}]
curl -g -X POST -d '{"match": {"ip_dscp": "0", "in_port": "3"}, "actions": {"queue": "1"}}'
http://127.0.0.1:8080/qos/rules/0000000000000001 [{"switch_id":
```

```

"0000000000000001","command_result": [{"result": "success","details": "QoS added. :  
qos_id=4"}]}]

curl -g -X POST -d '{"match": { "ip_dscp": "18", "in_port": "3"}, "actions":{ "queue": "3"} }'  
http://127.0.0.1:8080/qos/rules/0000000000000001 [{"switch_id":  
"0000000000000001","command_result": [{"result": "success","details": "QoS added. :  
qos_id=5"}]}]

curl -g -X POST -d '{"match": { "ip_dscp": "36", "in_port": "3"}, "actions":{ "queue": "2"} }'  
http://127.0.0.1:8080/qos/rules/0000000000000001 [{"switch_id":  
"0000000000000001","command_result": [{"result": "success","details": "QoS added. :  
qos_id=6"}]}]

curl -g -X GET http://127.0.0.1:8080/qos/rules/0000000000000001 (gets S1 rules)

curl -g -X POST -d '{"match": { "ip_dscp": "18"}, "actions":{ "meter": "1"} }'  
http://127.0.0.1:8080/qos/rules/0000000000000002

curl -g -X POST -d '{"meter_id": "1", "flags": "KBPS",  
"bands": [{"type": "DSCP_REMARK", "rate": "400", "prec_level": "1"}]}'  
http://127.0.0.1:8080/qos/meter/0000000000000002

curl -g -X POST -d '{"match": { "ip_dscp": "18"}, "actions":{ "meter": "1"} }'  
http://127.0.0.1:8080/qos/rules/0000000000000003

curl -g -X POST -d '{"meter_id": "1", "flags": "KBPS",  
"bands": [{"type": "DSCP_REMARK", "rate": "400", "prec_level": "1"}]}'  
http://127.0.0.1:8080/qos/meter/0000000000000003

curl -g -X GET http://127.0.0.1:8080/qos/rules/0000000000000002 (gets S2 rules)  

curl -g -X GET http://127.0.0.1:8080/qos/rules/0000000000000003 (gets S3 rules)  

curl -g -X GET http://127.0.0.1:8080/qos/meter/0000000000000002 (gets S2 meter)  

curl -g -X GET http://127.0.0.1:8080/qos/meter/0000000000000003 (gets S3 meter)

```

Appendix 10

VM1 (OpenFlow1) - TE with LDP (TEinLinux.sh)

```

sudo ifconfig eth0 192.16.10.10 netmask 255.255.255.0 up  

sudo ip route add 192.16.10.0/24 via 192.16.10.2  

sudo ip route add default via 192.16.10.2 dev eth0

```

MPLS1 - TE with LDP (TEinLinux.sh)

```
sudo modprobe mpls_router
sudo modprobe mpls_gso
sudo modprobe mpls_iptunnel
sudo sysctl -w net.mpls.conf.eth1.input=1
sudo sysctl -w net.mpls.platform_labels=1048575
sudo ifconfig eth0 192.16.10.2 netmask 255.255.255.0 up
sudo ifconfig eth1 100.0.2.2 netmask 255.255.255.0 up
sudo bash -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
sudo bash -c "echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter"
sudo ip link add name lo0 type dummy
sudo ip link set dev lo0 up
sudo ip addr add 4.4.4.4/32 dev lo0
sudo iptables -t mangle -A FORWARD -p tcp --dport 20 -j DSCP --set-dscp-class EF
sudo iptables -t mangle -A FORWARD -p tcp --dport 21 -j DSCP --set-dscp-class EF
sudo iptables -t mangle -A FORWARD -p tcp --dport 80 -j DSCP --set-dscp-class EF
```

frr version 3.1-dev-MyOwnFRRVersion

frr defaults traditional

hostname mpls1

no ipv6 forwarding

username cumulus nopassword

!

service integrated-vtysh-config

!

log syslog informational

!

router ospf

ospf router-id 4.4.4.4

log-adjacency-changes

network 4.4.4.4/32 area 0

```
network 100.0.2.0/24 area 0
network 192.16.10.0/24 area 0
!
mpls ldp
router-id 4.4.4.4
!
address-family ipv4
discovery transport-address 4.4.4.4
!
interface eth1
!
exit-address-family
!
!
mpls label global-block 4000 4999
!
line vty
!
```

CSR1000V1 - TE with LDP

```
CSR1000V1#sh run
```

```
Building configuration...
```

```
Current configuration : 3210 bytes
```

```
!
```

```
! Last configuration change at 18:16:25 UTC Sun Feb 4 2018
```

```
!
```

```
version 15.4
```

```
service timestamps debug datetime msec
```

```
service timestamps log datetime msec
```

```
no platform punt-keepalive disable-kernel-core
```

```
platform console virtual
!
hostname CSR1000V1
!
boot-start-marker
boot-end-marker
!
enable secret 5 $1$IVGd$umXioKuLNXxL8lqW/iM8i/
!
no aaa new-model
!
no ip domain lookup
!
subscriber templating
!
mpls label range 1000 1999
mpls traffic-eng tunnels
mpls traffic-eng reoptimize timers frequency 10
multilink bundle-name authenticated
!
license udi pid CSR1000V sn 9HJT0U4IXIZ
license boot level premium
spanning-tree extend system-id
!
redundancy
mode none
!
interface Loopback100
ip address 100.100.100.1 255.255.255.255
!
```

```
interface Tunnel0
bandwidth 1024
ip unnumbered Loopback100
tunnel mode mpls traffic-eng
tunnel destination 100.100.100.2
tunnel mpls traffic-eng autoroute announce
tunnel mpls traffic-eng priority 2 2
tunnel mpls traffic-eng bandwidth 1024
tunnel mpls traffic-eng path-option 1 explicit name CSR1-CSR2
!
interface Tunnel1
bandwidth 6144
ip unnumbered Loopback100
tunnel mode mpls traffic-eng
tunnel destination 100.100.100.2
tunnel mpls traffic-eng autoroute announce
tunnel mpls traffic-eng priority 1 1
tunnel mpls traffic-eng bandwidth 1024
tunnel mpls traffic-eng path-option 1 explicit name CSR1-Dublin-CSR2
!
interface GigabitEthernet1
ip address 100.0.2.3 255.255.255.0
ip policy route-map pbr
negotiation auto
mpls ip
mpls traffic-eng tunnels
ip rsvp bandwidth 6144 6144
!
interface GigabitEthernet2
ip address 200.0.2.2 255.255.255.0
```

```
ip nbar protocol-discovery
load-interval 30
negotiation auto
mpls ip
mpls traffic-eng tunnels
ip rsvp bandwidth 6144 6144
!
interface GigabitEthernet3
description TELNET -> Management Access
ip address 192.168.2.241 255.255.255.0
negotiation auto
!
interface GigabitEthernet4
ip address 200.0.4.1 255.255.255.0
negotiation auto
mpls ip
mpls traffic-eng tunnels
ip rsvp bandwidth 1024
!
router ospf 1
router-id 100.100.100.1
passive-interface GigabitEthernet3
network 100.0.2.0 0.0.0.255 area 0
network 100.100.100.1 0.0.0.0 area 0
network 200.0.2.0 0.0.0.255 area 0
network 200.0.4.0 0.0.0.255 area 0
mpls traffic-eng router-id Loopback100
mpls traffic-eng area 0
!
virtual-service csr_mgmt
```

```
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
!
ip explicit-path name CSR1-Dublin-CSR2 enable
next-address 200.0.2.3
next-address 200.0.6.4
!
ip explicit-path name CSR1-CSR2 enable
next-address 200.0.4.2
!
ip access-list extended CS1
permit ip host 192.16.10.10 host 100.100.100.2 dscp cs1
permit ip any precedence priority
ip access-list extended EF
permit ip host 192.16.10.10 host 100.100.100.2 dscp ef
permit ip any precedence critical
ip access-list extended noDSCP
permit ip any host 100.100.100.2
!
route-map pbr permit 10
match ip address EF
set interface Tunnel0
!
route-map pbr permit 20
match ip address CS1
set interface Tunnel1
!
```

```
control-plane
```

```
!
```

```
line con 0
```

```
  stopbits 1
```

```
line vty 0 4
```

```
  exec-timeout 0 0
```

```
  password csr
```

```
  logging synchronous
```

```
  login
```

```
!
```

```
end
```

Dublin - TE with LDP

```
Dublin#sh run
```

```
Building configuration...
```

```
Current configuration: 1629 bytes
```

```
!
```

```
version 15.1
```

```
service timestamps debug datetime msec
```

```
service timestamps log datetime msec
```

```
no service password-encryption
```

```
!
```

```
hostname Dublin
```

```
!
```

```
boot-start-marker
```

```
boot-end-marker
```

```
!
```

```
no aaa new-model
```

```
!
```

```
dot11 syslog
ip source-route
!
ip cef
no ip domain lookup
no ipv6 cef
!
multilink bundle-name authenticated
!
mpls traffic-eng tunnels
mpls label range 2000 2999
!
voice-card0
!
crypto pki token default removal timeout 0
!
license udi pid CISCO2801 sn FCZ131592KK
!
redundancy
!
interface Loopback0
ip address 1.1.1.1 255.255.255.255
!
interface FastEthernet0/0
ip address 200.0.2.3 255.255.255.0
ip nbar protocol-discovery
duplex auto
speed auto
mpls traffic-eng tunnels
mpls ip
```

```
ip rsvp bandwidth 6144 6144
!
interface FastEthernet0/1
ip address 200.0.6.3 255.255.255.0
ip nbar protocol-discovery
duplex auto
speed auto
mpls traffic-eng tunnels
mpls ip
ip rsvp bandwidth 6144 6144
!
interface Serial0/2/0
no ip address
shutdown
clock rate 2000000
!
interface Serial0/2/1
no ip address
shutdown
clock rate 2000000
!
interface FastEthernet0/3/0
no ip address
shutdown
duplex auto
speed auto
!
router ospf 1
mpls traffic-eng router-id Loopback0
mpls traffic-eng area 0
```

```
router-id 1.1.1.1
network 1.1.1.1 0.0.0.0 area 0
network 200.0.2.0 0.0.0.255 area 0
network 200.0.6.0 0.0.0.255 area 0
!
ip forward-protocol nd
no ip http server
no ip http secure-server
!
control-plane
!
mgcp profile default
!
line con 0
exec-timeout 0 0
line aux 0
line vty 0 4
login
transport input all
!
scheduler allocate 20000 1000
end
```

CSR1000V2 - TE with LDP

```
CSR1000V2#sh run
```

```
Building configuration...
```

```
Current configuration: 3210 bytes
```

```
!
```

```
! Last configuration change at 18:16:26 UTC Sun Feb 4, 2018
```

```
!
```

version 15.4

service timestamps debug datetime msec

service timestamps log datetime msec

no platform punt-keepalive disable-kernel-core

platform console virtual

!

hostname CSR1000V2

!

boot-start-marker

boot-end-marker

!

!

enable secret 5 \$1\$h4u9\$m8v03bZWACUQ5TFNtueif.

!

no aaa new-model

!

no ip domain lookup

!

subscriber templating

!

mpls label range 3000 3999

mpls traffic-eng tunnels

mpls traffic-eng reoptimize timers frequency 10

multilink bundle-name authenticated

!

license udi pid CSR1000V sn 91TETUYP9DF

license boot level premium

spanning-tree extend system-id

!

redundancy

```
mode none
!
interface Loopback100
ip address 100.100.100.2 255.255.255.255
!
interface Tunnel0
bandwidth 1024
ip unnumbered Loopback100
tunnel mode mpls traffic-eng
tunnel destination 100.100.100.1
tunnel mpls traffic-eng autoroute announce
tunnel mpls traffic-eng priority 2 2
tunnel mpls traffic-eng bandwidth 1024
tunnel mpls traffic-eng path-option 1 explicit name CSR2-CSR1
!
interface Tunnel1
bandwidth 6144
ip unnumbered Loopback100
tunnel mode mpls traffic-eng
tunnel destination 100.100.100.1
tunnel mpls traffic-eng autoroute announce
tunnel mpls traffic-eng priority 1 1
tunnel mpls traffic-eng bandwidth 1024
tunnel mpls traffic-eng path-option 1 explicit name CSR2-Dublin-CSR1
!
interface GigabitEthernet1
ip address 100.0.6.3 255.255.255.0
ip policy route-map pbr
negotiation auto
mpls ip
```

```
mpls traffic-eng tunnels
ip rsvp bandwidth 6144 6144
!
interface GigabitEthernet2
ip address 200.0.6.4 255.255.255.0
ip nbar protocol-discovery
load-interval 30
negotiation auto
mpls ip
mpls traffic-eng tunnels
ip rsvp bandwidth 6144 6144
!
interface GigabitEthernet3
description TELNET -> Management Access
ip address 192.168.2.242 255.255.255.0
negotiation auto
!
interface GigabitEthernet4
ip address 200.0.4.2 255.255.255.0
negotiation auto
mpls ip
mpls traffic-eng tunnels
ip rsvp bandwidth 1024
!
router ospf 1
router-id 100.100.100.2
passive-interface GigabitEthernet3
network 100.0.6.0 0.0.0.255 area 0
network 100.100.100.2 0.0.0.0 area 0
network 200.0.4.0 0.0.0.255 area 0
```

```
network 200.0.6.0 0.0.0.255 area 0
mpls traffic-eng router-id Loopback100
mpls traffic-eng area 0
!
virtual-service csr_mgmt
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
!
ip explicit-path name CSR2-Dublin-CSR1 enable
next-address 200.0.6.3
next-address 200.0.2.2
!
ip explicit-path name CSR2-CSR1 enable
next-address 200.0.4.1
!
ip access-list extended CS1
permit ip host 192.16.20.20 host 100.100.100.1 dscp cs1
permit ip any any precedence priority
ip access-list extended EF
permit ip host 192.16.20.20 host 100.100.100.1 dscp ef
permit ip any any precedence critical
ip access-list extended noDSCP
permit ip any host 100.100.100.1
!
route-map pbr permit 10
match ip address EF
set interface Tunnel0
```

```

!
route-map pbr permit 20
match ip address CS1
set interface Tunnel1
!
control-plane
!
line con 0
stopbits 1
line vty 0 4
exec-timeout 0 0
password csr
logging synchronous
login
!
end

```

MPLS2 - TE with LDP (TEinLinux.sh)

```

sudo modprobe mpls_router
sudo modprobe mpls_gso
sudo modprobe mpls_iptunnel
sudo sysctl -w net.mpls.conf.eth1.input=1
sudo sysctl -w net.mpls.platform_labels=1048575
sudo ifconfig eth0 192.16.20.4 netmask 255.255.255.0 up
sudo ifconfig eth1 100.0.6.4 netmask 255.255.255.0 up
sudo bash -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
sudo bash -c "echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter"
sudo ip link add name lo0 type dummy
sudo ip link set dev lo0 up
sudo ip addr add 5.5.5.5/32 dev lo0
sudo iptables -t mangle -A FORWARD -p tcp --dport 20 -j DSCP --set-dscp-class EF

```

```
sudo iptables -t mangle -A FORWARD -p tcp --dport 21 -j DSCP --set-dscp-class EF  
sudo iptables -t mangle -A FORWARD -p tcp --dport 80 -j DSCP --set-dscp-class EF
```

```
frr version 3.1-dev-MyOwnFRRVersion
```

```
frr defaults traditional
```

```
hostname mpls2
```

```
no ipv6 forwarding
```

```
username cumulus nopassword
```

```
!
```

```
service integrated-vtysh-config
```

```
!
```

```
log syslog informational
```

```
!
```

```
router ospf
```

```
ospf router-id 5.5.5.5
```

```
log-adjacency-changes
```

```
network 5.5.5.5/32 area 0
```

```
network 100.0.6.0/24 area 0
```

```
network 192.16.20.0/24 area 0
```

```
!
```

```
mpls ldp
```

```
router-id 5.5.5.5
```

```
!
```

```
address-family ipv4
```

```
discovery transport-address 5.5.5.5
```

```
!
```

```
interface eth1
```

```
!
```

```
exit-address-family
```

```
!
```

```
!
mpls label global-block 5000 5999
!
line vty
!
```

VM2 (OpenFlow2) - TE with LDP (TEinLinux.sh)

```
sudo ifconfig eth0 192.16.20.20 netmask 255.255.255.0 up
sudo ip route add 192.16.10.0/24 via 192.16.20.4
sudo ip route add default via 192.16.20.4 dev eth0
```

SDN - TE with Floodlight

```
sudo nano /etc/network/interfaces
```

```
auto eth0
iface eth0 inet static
address 192.16.20.30/24
gateway 192.16.20.1
auto eth1
iface eth1 inet manual
pre-up ifconfig eth1 up
```

```
sudo nano /etc/resolvconf/resolv.conf.d/base
nameserver 8.8.8.8
nameserver 8.8.4.4
```

SDN - TE with Floodlight (OpenFlow3.py)

```
java -jar ./floodlight/target/floodlight.jar (runs Floodlight in terminal)
```

```
sudo gedit ./Downloads/OpenFlow3.py
```

```
#!/usr/bin/python
import re
```

```

import sys

from mininet.cli import CLI
from mininet.log import setLogLevel, info, error
from mininet.net import Mininet
from mininet.link import Intf
from mininet.topolib import TreeTopo
from mininet.util import quietRun
from mininet.node import Controller, RemoteController, OVSCController, OVSSwitch

def checkIntf( intf ):
    "Make sure intf exists and is not configured"
    config = quietRun( 'ifconfig %s 2>/dev/null' % intf, shell=True )
    if not config:
        error( 'Error:', intf, 'does not exist!\n' )
        exit( 1 )
    ips = re.findall( r'\d+\.\d+\.\d+\.\d+', config )
    if ips:
        error( 'Error:', intf, 'has an IP address,'
               'and is probably in use!\n' )
        exit( 1 )

if __name__ == '__main__':
    setLogLevel( 'info' )

    # Gets HW interface from the command line
    intfName = sys.argv[ 1 ] if len( sys.argv ) > 1 else 'eth1'
    info( '*** Connecting to hw intf: %s' % intfName )

    info( '*** Checking', intfName, '\n' )

```

```

checkIntf( intfName )

info( '*** Creating network\n' )

net = Mininet()

#Controller = net.addController(name='OVS', controller=RemoteController,
ip='192.16.20.30', port=6633)

Controller = net.addController(name='Floodlight', controller=RemoteController,
ip='192.16.20.31', port=6653)

#Controller = net.addController(name='OpenDaylight', controller=RemoteController,
ip='192.16.20.32', port=6633)

#Controller = net.addController(name='ONOS', controller=RemoteController,
ip='192.16.20.33', port=6633)

#Controller = net.addController(name='POX', controller=RemoteController,
ip='192.16.20.34', port=6633)

#Controller = net.addController(name='Ryu', controller=RemoteController,
ip='192.16.20.35', port=6633)

#Controller = net.addController(name='HPVANSNDN', controller=RemoteController,
ip='192.16.20.254', port=6633)

# Creates two hosts

VM1 = net.addHost('VM1', ip='192.16.20.2/24', defaultRoute='via 192.16.20.1')

VM2 = net.addHost('VM2', ip='192.16.20.4/24', defaultRoute='via 192.16.20.1')

# Creates three switches

S1 = net.addSwitch('S1')

S2 = net.addSwitch('S2')

S3 = net.addSwitch('S3')

S4 = net.addSwitch('S4')

S5 = net.addSwitch('S5')

```

```

# Adds links between the switches and each host

net.addLink(VM1,S4)

net.addLink(S1,S3)

net.addLink(S3,S2)

net.addLink(S2,S1)

net.addLink(VM2,S5)

net.addLink(S2,S5)

net.addLink(S1,S4)

```

```

switch = net.switches[ 0 ]

info( "*** Adding hardware interface", intfName, 'to switch',
      S3, '\n' )

_intf = Intf( intfName, node=S3 )

```

```

info( "*** Note: Reconfigure the interfaces for '
      'the Mininet hosts:\n', net.hosts, '\n' )

```

```

net.start()

CLI( net )

net.stop()

```

```

sudo chmod 777 ./Downloads/OpenFlow3.py (changes permission)

sudo ./Downloads/OpenFlow3.py (executes Custom Topo)

```

SDN - TE with Floodlight (FLPushFlowsTE.py)

```
sudo gedit ./Downloads/FLPushFlowsTE.py
```

```

#!/usr/bin/python

import httplib

import json

```

```
class StaticEntryPusher(object):

    def __init__(self, server):
        self.server = server

    def get(self, data):
        ret = self.rest_call({ }, 'GET')
        return json.loads(ret[2])

    def set(self, data):
        ret = self.rest_call(data, 'POST')
        return ret[0] == 200

    def remove(self, objtype, data):
        ret = self.rest_call(data, 'DELETE')
        return ret[0] == 200

    def rest_call(self, data, action):
        path = '/wm/staticentrypusher/json'
        headers = {
            'Content-type': 'application/json',
            'Accept': 'application/json',
        }
        body = json.dumps(data)
        conn = httplib.HTTPConnection(self.server, 8080)
        conn.request(action, path, body, headers)
        response = conn.getresponse()
        ret = (response.status, response.reason, response.read())
        print ret
```

```

conn.close()

return ret

pusher = StaticEntryPusher('192.16.20.31')

flow1 = {
    'switch':"00:00:00:00:00:00:00:04",
    "name":"flow_mod_1",
    "cookie":"0",
    "in_port":"1",
    "priority":"600",
    "eth_type":"0x800",
    "ipv4_src":"192.16.20.2",
    "ipv4_dst":"192.16.20.20",
    "active":"true",
    "actions":"output=2"
}

flow2 = {
    'switch':"00:00:00:00:00:00:00:01",
    "name":"flow_mod_2",
    "cookie":"0",
    "in_port":"3",
    "priority":"600",
    "eth_type":"0x800",
    "ipv4_src":"192.16.20.2",
    "ipv4_dst":"192.16.20.20",
    "active":"true",
    "actions":"output=2"
}

```

```
flow3 = {  
    'switch':"00:00:00:00:00:00:00:02",  
    "name":"flow_mod_3",  
    "cookie":"0",  
    "in_port":2,  
    "priority":600,  
    "eth_type":0x800,  
    "ipv4_src":192.16.20.2,  
    "ipv4_dst":192.16.20.20,  
    "active":true,  
    "actions": "output=1"  
}
```

```
flow4 = {  
    'switch':"00:00:00:00:00:00:00:03",  
    "name":"flow_mod_4",  
    "cookie":"0",  
    "in_port":2,  
    "priority":600,  
    "eth_type":0x800,  
    "ipv4_src":192.16.20.2,  
    "ipv4_dst":192.16.20.20,  
    "active":true,  
    "actions": "output=3"  
}
```

```
flow5 = {  
    'switch':"00:00:00:00:00:00:00:03",  
    "name":"flow_mod_5",
```

```
"cookie":"0",
    "in_port":"3",
"priority":"600",
    "eth_type":"0x800",
    "ipv4_src":"192.16.20.20",
    "ipv4_dst":"192.16.20.2",
"active":"true",
"actions":"output=2"
}
```

```
flow6 = {
'switch':"00:00:00:00:00:00:00:02",
"name":"flow_mod_6",
"cookie":"0",
    "in_port":"1",
"priority":"600",
    "eth_type":"0x800",
    "ipv4_src":"192.16.20.20",
    "ipv4_dst":"192.16.20.2",
"active":"true",
"actions":"output=2"
}
```

```
flow7 = {
'switch':"00:00:00:00:00:00:00:01",
"name":"flow_mod_7",
"cookie":"0",
    "in_port":"2",
"priority":"600",
    "eth_type":"0x800",
```

```
    "ipv4_src":"192.16.20.20",
    "ipv4_dst":"192.16.20.2",
    "active":"true",
    "actions":"output=3"
}
```

```
flow8 = {
    'switch':"00:00:00:00:00:00:00:04",
    "name":"flow_mod_8",
    "cookie":"0",
    "in_port":"2",
    "priority":"600",
    "eth_type":"0x800",
    "ipv4_src":"192.16.20.20",
    "ipv4_dst":"192.16.20.2",
    "active":"true",
    "actions":"output=1"
}
```

```
pusher.set(flow1)
pusher.set(flow2)
pusher.set(flow3)
pusher.set(flow4)
pusher.set(flow5)
pusher.set(flow6)
pusher.set(flow7)
pusher.set(flow8)
```

```
sudo chmod 777 ./Downloads/FLPushFlowsTE.py (changes permission)
```

```
sudo python ./Downloads/FLPushFlowsTE.py (executes Custom Topo)
```

VM1 (OpenFlow1) - TE with Floodlight

```
sudo nano /etc/network/interfaces
```

```
auto eth0
```

```
iface eth0 inet static
```

```
address 192.16.20.10/24
```

```
gateway 192.16.20.1
```

```
sudo nano /etc/resolvconf/resolv.conf.d/base
```

```
nameserver 8.8.8.8
```

```
nameserver 8.8.4.4
```

VM2 (OpenFlow2) - TE with Floodlight

```
sudo nano /etc/network/interfaces
```

```
auto eth0
```

```
iface eth0 inet static
```

```
address 192.16.20.20/24
```

```
gateway 192.16.20.1
```

```
sudo nano /etc/resolvconf/resolv.conf.d/base
```

```
nameserver 8.8.8.8
```

```
nameserver 8.8.4.4
```

SDN - TE with HPE Aruba VAN (HPEPushFlowsTE.sh)

```
sudo gedit ./Downloads/HPEPushFlowsTE.sh
```

```
#!/bin/bash
```

```
# Dump the specified message to stderr and exit.
```

```
function error {
```

```
echo "$@" >&2 && exit 1
```

```
}
```

```
tok="14d95ddd52d840a494331ff27331e541"
```

```
auth=$(curl -sk -H 'Content-Type:application/json' -d  
'{"login":{"user":"sdn","password":"skyline"} }' https://192.16.20.254:8443/sdn/v2.0/auth)  
echo "$auth" | sed "s/.*/\"token\":([^\"]*)\.\*/1;/^\$/d" | tr -d '"' > $tok
```

```
token="$([ -f $tok ] && cat $tok)"
```

```
payload1='{  
    "flow":{  
        "priority":60001,  
        "table_id":0,  
        "match": [  
            {"ipv4_src":"192.16.20.2"},  
            {"ipv4_dst":"192.16.20.20"},  
            {"eth_type":"ipv4"}],  
        "instructions": [  
            {"apply_actions": [{"output":2}]}]  
    }'  
}'
```

```
payload2='{  
    "flow":{  
        "priority":60001,  
        "table_id":0,  
        "match": [  
            {"ipv4_src":"192.16.20.2"},  
            {"ipv4_dst":"192.16.20.20"},  
            {"eth_type":"ipv4"}],  
        "instructions": [  
            {"apply_actions": [{"output":2}]}]  
    }'  
}'
```

```
payload3='{
"flow": {
    "priority":60001,
    "table_id":0,
    "match": [
        { "ipv4_src":"192.16.20.2"}, 
        { "ipv4_dst":"192.16.20.20"}, 
        { "eth_type":"ipv4"}],
    "instructions": [
        { "apply_actions": [ { "output":1 } ] } ]
}
}'
```

```
payload4='{
"flow": {
    "priority":60001,
    "table_id":0,
    "match": [
        { "ipv4_src":"192.16.20.2"}, 
        { "ipv4_dst":"192.16.20.20"}, 
        { "eth_type":"ipv4"}],
    "instructions": [
        { "apply_actions": [ { "output":3 } ] } ]
}
}'
```

```
payload5='{
"flow": {
    "priority":60001,
    "table_id":0,
    "match": [
        { "ipv4_src":"192.16.20.20"}, 
        { "ipv4_dst":"192.16.20.2"}, 
        { "eth_type":"ipv4"}]
}
}'
```

```
        {"eth_type":"ipv4"}],  
        "instructions": [{"apply_actions": [{"output":2}]}]}  
    }'
```

```
payload6='{  
"flow":{  
    "priority":60001,  
    "table_id":0,  
    "match": [  
        {"ipv4_src":"192.16.20.20"},  
        {"ipv4_dst":"192.16.20.2"},  
        {"eth_type":"ipv4"}],  
        "instructions": [{"apply_actions": [{"output":2}]}]}  
}'
```

```
payload7='{  
"flow":{  
    "priority":60001,  
    "table_id":0,  
    "match": [  
        {"ipv4_src":"192.16.20.20"},  
        {"ipv4_dst":"192.16.20.2"},  
        {"eth_type":"ipv4"}],  
        "instructions": [{"apply_actions": [{"output":3}]}]}  
}'
```

```
payload8='{  
"flow":{  
    "priority":60001,  
    "table_id":0,
```

```
"match": [
    {"ipv4_src":"192.16.20.20"},  

    {"ipv4_dst":"192.16.20.2"},  

    {"eth_type":"ipv4"}],  

    "instructions": [{"apply_actions": [{"output":1}]}]}  

}'
```

tok="14d95ddd52d840a494331ff27331e541"

```
echo $payload1 > d1  

echo $payload2 > d2  

echo $payload3 > d3  

echo $payload4 > d4  

echo $payload5 > d5  

echo $payload6 > d6  

echo $payload7 > d7  

echo $payload8 > d8
```

tok="14d95ddd52d840a494331ff27331e541"

```
token="$( [ -f $tok ] && cat $tok)"
```

```
#Set datapaths
```

```
echo  

=====
=====  

printf "\n"
```

```
#curl -sk -H "X-Auth-Token:$token" -X POST -H 'Content-Type:application/json' -d
```

```
echo "Setting flows..."
```

```
curl -sk -H "X-Auth-Token:$token" -X POST -H 'Content-Type:application/json' -d @d1
https://192.16.20.254:8443/sdn/v2.0/of/datapaths/00:00:00:00:00:00:04/flows

curl -sk -H "X-Auth-Token:$token" -X POST -H 'Content-Type:application/json' -d @d2
https://192.16.20.254:8443/sdn/v2.0/of/datapaths/00:00:00:00:00:00:01/flows

curl -sk -H "X-Auth-Token:$token" -X POST -H 'Content-Type:application/json' -d @d3
https://192.16.20.254:8443/sdn/v2.0/of/datapaths/00:00:00:00:00:00:02/flows

curl -sk -H "X-Auth-Token:$token" -X POST -H 'Content-Type:application/json' -d @d4
https://192.16.20.254:8443/sdn/v2.0/of/datapaths/00:00:00:00:00:00:03/flows

curl -sk -H "X-Auth-Token:$token" -X POST -H 'Content-Type:application/json' -d @d5
https://192.16.20.254:8443/sdn/v2.0/of/datapaths/00:00:00:00:00:00:03/flows

curl -sk -H "X-Auth-Token:$token" -X POST -H 'Content-Type:application/json' -d @d6
https://192.16.20.254:8443/sdn/v2.0/of/datapaths/00:00:00:00:00:00:02/flows

curl -sk -H "X-Auth-Token:$token" -X POST -H 'Content-Type:application/json' -d @d7
https://192.16.20.254:8443/sdn/v2.0/of/datapaths/00:00:00:00:00:00:01/flows

curl -sk -H "X-Auth-Token:$token" -X POST -H 'Content-Type:application/json' -d @d8
https://192.16.20.254:8443/sdn/v2.0/of/datapaths/00:00:00:00:00:00:04/flows
```

```
echo "Finished"
printf "\n\n"
echo
=====
=====
```

```
sudo chmod 777 ./Downloads/HPEPushFlowsTE.sh (changes permission)
sudo ./Downloads/HPEPushFlowsTE.sh (executes Custom Topo)
```

Appendix 11

Software Packages

Filename	Description
anyconnect-win-3.1.00495-pre-deploy-k9.msi	MSI of Cisco VPN Client v3.1.00495
CDM v2.12.26 WHQL Certified.zip	ZIP of USB to RJ14 Drivers v2.12.26
chr-6.40.4.ova	OVA of MikroTik RouterOS v6.40.4
chr-6.40.4.vmdk	VMDK of MikroTik RouterOS v6.40.4
csr1000v-universalk9.03.12.00.S.154-2.S-std.iso	ISO of Cisco CSR 1000V Series Advanced Enterprise Services v.9.03.12.0S.154-2
csr1000v-universalk9.03.12.00.S.154-2.S-std.ova	OVA of Cisco CSR 1000V Series Advanced Enterprise Services v.9.03.12.0S.154-2
Floodlight-v1.1+Mininet.vmdk	VMDK of Floodlight Controller VM v1.1
floodlight-vm.zip	ZIP of Floodlight Controller VM v1.1
flow-maker.zip	ZIP of Northbound Networks Flow Maker v1.1
hpe-net-optimizer-v1.5.4.109-x64.zip	ZIP of HPE Network Optimizer v1.5.4.109 X64
hpe-net-protector-1.5.4.30-x64.zip	ZIP of HPE Network Protector v1.5.4.30 X64
hpe-van-sdn-ctlr-2.8.8.0366-x64.zip	ZIP of HPE Aruba VAN SDN Controller v2.8.8.0366 X64
iperf-3.1.3-win64.zip	ZIP of iPerf v3.1.3 Traffic Generator X64
mikrotik-6.40.4.iso	ISO of MikroTik RouterOS v6.40.4
odl-test-desktop-4.ova	OVA of OpenDaylight Controller Test Desktop VM v4
onos-tutorial-1.2.1r2-ovf.zip	ZIP of ONOS Controller VM v1.2.1r2
putty-64bit-0.69-installer.msi	MSI of Putty Client v0.69 X64
python-2.7.14.msi	MSI of Python Libraries v.2.7.14
python-3.6.4.exe	EXE of Python Libraries v.3.6.4

SDN_tutorial_VM_64bit.ova	OVA of All-in-one SDN App Development Starter VM X64
sdn-toolkit_v1.00_openflow.tar	TAR of SDN-Toolkit v1.00
sdn-toolkit_v1.01_openflow.tat	TAR of SDN-Toolkit v1.01
sdn-toolkit_v1.21_openflow.tar	TAR of SDN-Toolkit v1.21
spear-1.21.3645.zip	ZIP of Narmox Spear v1.21.3645
Successful registration of SDN Evaluation licenses.pdf	PDF of SDN Evaluation licenses for HPE Aruba VAN SDN Controller
ubuntu-14.04.5-server-amd64.iso	ISO of Ubuntu v14.04.5 LTS (Trusty Tahr) Server X64
ubuntu-16.04.2-desktop-amd64.iso	ISO of Ubuntu v16.04.2 LTS (Xenial Xerus) Desktop X64
wetransfer-bb15ee.zip	ZIP of Cisco CSR 1000V Series Advanced Enterprise Services v.9.03.11.0S.154-1
win32-mgen-5.02b.zip	ZIP of MGEN Traffic Generator v5.02b X86
WinSCP-5.11.3-Setup.exe	EXE of WinSCP Client v5.11.3
Xming-6-9-0-31-setup.exe	EXE of Xming Server v6.9.0.31

Thesis Documents

Filename	Description
01_Proposal.docx	DOCX of Dissertation Proposal last edited in May 2017
02_Plan.mpp	MPP of Project Plan last edited in April 2018
03_Dissertation Commencement Form.doc	DOC of Dissertation Commencement Form last edited in May 2017
04_Literature Review.docx	DOCX of Literature Review Chapter last edited in June 2017
05_Test Environment.docx	DOCX of Test Environment Chapter last edited in July 2017
06_Test Methodology.docx	DOCX of Test Methodology Chapter last edited in September 2017
07_Dissertation Progress Presentation.pdf	PDF of Dissertation Progress Presentation last edited in November 2017
07_Dissertation Progress Presentation.pptx	PPTX of Dissertation Progress Presentation last edited in November 2017
07_Implementation.docx	DOCX of Implementation Chapter last edited in February 2018
08_Summary.docx	DOCX of Summary Chapter last edited in March 2018
Final Thesis.docx	DOCX of Final Thesis last edited in April 2018
Final Thesis.pdf	PDF of Final Thesis last edited in April 2018
Reseach Paper.docx	DOCX of Research Paper last edited in April 2018
Reseach Paper.pdf	PDF of Research Paper last edited in April 2018
Final Thesis Presentation.pptx	PPTX of Final Thesis Presentation last edited in April 2018
Final Thesis Presentation.pdf	PDF of Final Thesis Presentation last edited in April 2018

Dariusz_Terenko_X00088029_1.zip	ZIP of Progress Update Documents Revision 1 last edited May 2017
Dariusz_Terenko_X00088029_2.zip	ZIP of Progress Update Documents Revision 2 last edited September 2017
Dariusz_Terenko_X00088029_3.zip	ZIP of Progress Update Documents Revision 3 last edited November 2017
Dariusz_Terenko_X00088029_4.zip	ZIP of Progress Update Documents Revision 4 last edited January 2018
Dariusz_Terenko_X00088029_5.zip	ZIP of Progress Update Documents Revision 5 last edited February 2018
Dariusz_Terenko_X00088029_6.zip	ZIP of Progress Update Documents Revision 6 last edited March 2018
Dariusz_Terenko_X00088029_7.zip	ZIP of Progress Update Documents Revision 7 last edited April 2018

Virtual Machines

Filename	Description
CSR1000V1	Cisco CSR 1000V Series Advanced Enterprise Services v.9.03.12.0S.154-2 Username: csr Password: csr
CSR1000V2	Cisco CSR 1000V Series Advanced Enterprise Services v.9.03.12.0S.154-2 Username: csr Password: csr
CSR1000V3	Cisco CSR 1000V Series Advanced Enterprise Services v.9.03.12.0S.154-2 Username: csr Password: csr
CSR1000V4	Cisco CSR 1000V Series Advanced Enterprise Services v.9.03.12.0S.154-2 Username: csr Password: csr
Floodlight	Floodlight Controller v1.1 Username: floodlight Password: floodlight
HPVANSNDN	HPE Aruba VAN SDN Controller v2.8.8.0366 X64 Username: sdn Password: skyline
MPLS1	MPLS1 VM (Kildare) Linux kernel v4.12 Username: mpls Password: mpls
MPLS2	MPLS2 VM (Laois) Linux kernel v4.12 Username: mpls Password: mpls

ONOS	ONOS Magpie Controller v1.12.0 OS Username: onos OS Password: onos GUI Username: onos GUI Password: rocks
OpenDaylight	OpenDaylight Carbon Controller v0.6.2 SR2 Username: admin Password: admin
OpenFlow1	OpenFlow1 (VM1) Linux kernel v4.12 Username: openflow Password: openflow
OpenFlow2	OpenFlow1 (VM2) Linux kernel v4.12 Username: openflow Password: openflow
OpenFlow3	OpenFlow3 Linux kernel v4.12 Username: openflow Password: openflow
POX	POX Eel Controller v0.5.0 Username: ubuntu Password: ubuntu
Ryu	Ryu Controller v4.20 Username: ryu Password: ryu
SDN	SDN (Mininet VM) Linux kernel v4.12 Username: sdn Password: sdn

List of References

- 2Connect (2017) *The Increasing Popularity Of MPLS*. Available at:
<https://www.leasedlineandmpls.co.uk/the-increasing-popularity-of-mpls/> (Accessed: 25 September 2017).
- Abinaiya, N. and Jayageetha, J. (2015) ‘A Survey On Multi Protocol Label Switching’, *International Journal of Technology Enhancements and Emerging Engineering Research*, vol. 3, no. 2, pp. 25–28. Available at: <http://www.ijteee.org/final-print/feb2015/A-Survey-On-Multi-Protocol-Label-Switching.pdf> (Accessed: 4 June 2017).
- Almquist, P. (1992) ‘Type of Service in the Internet Protocol Suite’, *RFC 1349*, July 1992. Available at: <http://www.ietf.org/rfc/rfc1349.txt> (Accessed: 14 November 2017).
- Anderson, M. (2016) ‘How To Set Up vsftpd for a User's Directory on Ubuntu 16.04’, *DigitalOcean Tutorial*, September 2016. Available at:
<https://www.digitalocean.com/community/tutorials/how-to-set-up-vsftpd-for-a-user-s-directory-on-ubuntu-16-04> (Accessed: 19 November 2017).
- Antichi, G. (2017) *Open Source Network Tester*. Available at: <http://osnt.org> (Accessed: 25 February 2018).
- Arista (2017). *The World’s Most Advanced Network Operating System*. Available at:
<https://www.arista.com/en/products/eos> (Accessed: 23 December 2017).
- AskUbuntu (2013) *There are no interfaces on which a capture can be done*. Available at:
<https://askubuntu.com/questions/348712/there-are-no-interfaces-on-which-a-capture-can-be-done> (Accessed: 23 July 2017).
- Aun (2016) ‘Install Quagga Routing Suite on Ubuntu 15.10’, *LinuxPitStop*, March 2016. Available at: <http://linuxpitstop.com/install-quagga-on-ubuntu-15-10/> (Accessed: 10 November 2017).
- Benita, Y. (2005) ‘Kernel Korner - Analysis of the HTB Queuing Discipline’, *Linux Journal*, January 2005. Available at: <http://www.linuxjournal.com/article/7562> (Accessed: 12 January 2018).
- Bauer, E. (2015) ‘Lync / Skype for Business SDN API: Network Friend or Foe’, *Integrated Research*, April 2015. Available at: <http://www.ir.com/blog/lync-skype-for-business-sdn-api-network-friend-or-foe> (Accessed: 7 January 2018).
- Bernardi, G. (2014) ‘SDN For Real’, *RIPE 69 Conference in London*, November 2014. Available at: <https://ripe69.ripe.net/presentations/11-RIPE69.pdf> (Accessed: 22 December 2017).

- Bertnard, G. (2014) ‘Top 3 Differences Between LDP And RSVP’, *Gilles Bertrand Blog*, March 2014. Available at: <http://www.gilles-bertrand.com/2014/03/ldp-vs-rsvp-key-differences-signaling-mpls.html> (Accessed: 9 December 2017).
- Bierman, A., Bjorklund, M., Watsen, K., Fernando, R. (2014) ‘RESTCONF Protocol’, *draft-ietf-netconf-restconf-00*, March 2014. Available at: <https://tools.ietf.org/html/draft-ietf-netconf-restconf-00> (Accessed: 31 December 2017).
- Big Switch Networks (2017) *ONL Hardware Support*. Available at: <https://opennetlinux.org/hcl> (Accessed: 20 December 2017).
- Bombal, D. (2015) ‘SDN 101: Using Mininet and SDN Controllers’, *Pakiti Blog*, November 2015. Available at: <http://pakiti.com/sdn-101-using-mininet-and-sdn-controllers/> (Accessed: 30 December 2017).
- Bombal, D. (2017) *SDN Courses on Udemy*. Available at: <https://www.udemy.com/courses/search/?q=%20Bombal&src=ukw&p=1&courseLabel=7650> (Accessed: 1 December 2017).
- Brent, S. (2012) ‘Getting Started OpenFlow OpenvSwitch Tutorial Lab: Setup’, *Brent Salisbury's Blog*, June 2012. Available at: <http://networkstatic.net/openflow-openvswitch-lab/> (Accessed: 1 October 2017).
- Brocade Communications Systems (2015) *How packets are forwarded through an MPLS domain*. Available at: <http://www.brocade.com/content/html/en/configuration-guide/netiron-05900-mplsguide/GUID-9BE9AAA9-5CC5-4A7E-B125-23CF171C1DCC.html> (Accessed: 4 June 2017).
- Burgess, J. (2008) ‘ONOS (Open Network Operating System)’, *Ingram Micro Advisor Blog*, August 2008. Available at: <http://www.ingrammicroadvisor.com/data-center/7-advantages-of-software-defined-networking> (Accessed: 5 June 2017).
- BW-Switch (2016) *ICOS AND LINUX SHELL MANAGEMENT*. Available at: <https://bm-switch.com/index.php/blog/icos-linux-shell/> (Accessed: 21 December 2017).
- Byte Solutions (2015) *DSCP TOS CoS Presidence conversion chart*. Available at: http://bytesolutions.com/Support/Knowledgebase/KB_Viewer/ArticleId/34/DSCP-TOS-CoS-Presidence-conversion-chart (Accessed: 20 January 2018).
- Callway (2015) ‘IoT World: Snappy for Whitebox Switches’, *Ubuntu Insights Article*, May 2015. Available at: <https://insights.ubuntu.com/2015/05/13/iot-world-snappy-for-whitebox-switches/> (Accessed: 20 December 2017).
- Chato, O. and William, E. (2016) ‘An Exploration of Various Quality of Service Mechanisms

in an OpenFlow and Software Defined Networking Environment', Shanghai, China: IEEE. pp. 738–776.

CheckYouMath (2017) *Convert Mbps to KBps, KBps to Mbps - Data Rate Conversions (Decimal)*. Available at:

https://www.checkyourmath.com/convert/data_rates/per_second/megabits_kilobytes_per_second.php (Accessed: 30 September 2017).

CheckYourMath (2017) *Convert Megabits to Bytes, Bytes to Megabits - Digital Storage Conversions (Binary)*. Available at:

https://www.checkyourmath.com/convert/digital_storage/megabits_bytes.php (Accessed: 31 October 2017).

Chiosi, M., Clarke, D., Willis, P., Reid, A., Feger, J., Bugenhagen, M., Khan, W., Fargano, M., Dr. Cui, C., Dr. Deng, H., Benitez, J., Michel, U., Damker, H., Ogaki, K., Matsuzaki, T., Fukui, M., Shimano, K., Delisle, D., Loudier, Q., Kolias, C., Guardini, I., Demaria, E., Minerva, R., Manzalini, A., Lopez, D., Salguero, F., J., R., Ruhl, F., Sen, P. (2012) 'Network Functions Virtualisation - An Introduction, Benefits, Enablers, Challenges & Call for Action', *SDN and OpenFlow World Congres*, October 2012. Available at:

https://portal.etsi.org/NFV/NFV_White_Paper.pdf (Accessed: 26 December 2017).

Cisco (2002) *Multiprotocol label switching (MPLS) on Cisco Routers*. Available at:

http://www.cisco.com/c/en/us/td/docs/ios/12_0s/feature/guide/fs_rtr22.html (Accessed: 12 February 2017).

Cisco (2005) *MPLS Label Distribution Protocol (LDP)*. Available at:

https://www.cisco.com/c/en/us/td/docs/ios/12_4t/12_4t2/fldp41.pdf (Accessed: 19 August 2017).

Cisco (2007) 'Configuring a Basic MPLS VPN', *Configuration Examples and TechNotes*, November 2007. Available at: <http://www.cisco.com/c/en/us/support/docs/multiprotocol-label-switching-mpls/13733-mpls-vpn-basic.html> (Accessed: 4 June 2017).

Cisco (2007) *MPLS Static Labels*. Available at:

http://www.cisco.com/c/en/us/td/docs/ios/mpls/configuration/guide/15_0s/mp_15_0s_book/mp_static_labels.pdf?rwXrSUMV2tLMIZnKHQ&sig2=g0xxUdu4Je2R-4V98V5NbA (Accessed: 18 February 2017).

Cisco (2014) *Cisco IOS Quality of Service Solutions Configuration Guide, Release 12.2*. Available at:

https://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfabout.htm

[1](#) (Accessed: 14 November 2017).

Cisco (2014) *Cisco IOS XRv Router Installation and Configuration Guide*. Available at:

https://www.cisco.com/en/US/docs/ios_xr_sw/ios_xrv/install_config/b_xrvr_432.pdf

(Accessed: 28 January 2017).

Cisco (2014) *Cisco Networking Academy's Introduction to Routing Dynamically*. Available at:

<http://www.ciscopress.com/articles/article.asp?p=2180210&seqNum=5> (Accessed: 12

November 2017).

Cisco (2014) *One- and 2-Port serial and Asynchronous high-speed WAN Interface cards for Cisco 1800, 1900, 2800, 2900, 3800, and 3900 series integrated services Routers*. Available at: http://www.cisco.com/c/en/us/products/collateral/interfaces-modules/high-speed-wan-interface-cards/datasheet_c78-491363.html (Accessed: 12 February 2017).

Cisco (2016) *1- and 2-Port fast Ethernet high-speed WIC for Cisco integrated services*

Routers data sheet. Available at:

http://www.cisco.com/c/en/us/products/collateral/routers/2800-series-integrated-services-routers-isr/product_data_sheet0900aecd80581fe6.html (Accessed: 12 February 2017).

Cisco (2016) *Cisco 2800 series integrated services Routers*. Available at:

http://www.cisco.com/c/en/us/products/collateral/routers/2800-series-integrated-services-routers-isr/product_data_sheet0900aecd8016fa68.html (Accessed: 12 February 2017).

Cisco (2017) ‘Chapter: Enabling Protocol Discovery’, *QoS: NBAR Configuration Guide, Cisco IOS Release 15M&T*, August 2017. Available at:

https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/qos_nbar/configuration/15-mt/qos-nbar-15-mt-book/nbar-protocol-discvry.html (Accessed: 27 November 2017).

Cisco (2017) ‘Installing the Cisco CSR 1000v in Microsoft Hyper-V Environments’, *Cisco CSR 1000v Series Cloud Services Router Software Configuration Guide*, August 2017.

Available at:

https://www.cisco.com/c/en/us/td/docs/routers/csr1000/software/configuration/b_CSR1000v_Configuration_Guide/b_CSR1000v_Configuration_Guide_chapter_0110.html#d52294e267a1635 (Accessed: 6 November 2017).

Cisco (2017) ‘Products & Services / Routers’, *Cisco Cloud Services Router 1000V Series*,

October 2017. Available at: <https://www.cisco.com/c/en/us/products/routers/cloud-services-router-1000v-series/index.html#~stickynav=1> (Accessed: 6 November 2017).

Cisco (2017) *Cloud Services Router 1000V Release 3.12.2S*. Available at:

<https://software.cisco.com/download/release.html?mdfid=284364978&softwareid=28204647>

[7&release=3.11.2S&rellifecycle=ED](#) (Accessed: 6 November 2017).

Cisco (2017) *IOS Software-15.1.4M12a*. Available at:

<https://software.cisco.com/download/release.html?mdfid=279316777&flowid=7672&softwareid=280805680&release=15.1.4M12a&relind=AVAILABLE&rellifecycle=MD&reltype=latest> (Accessed: 12 February 2017).

Cisco dCloud (2017) *OpenDaylight Carbon SR1 with Apps with 8 Nodes v1*. Available at:

<https://dcloud2-lon.cisco.com/content/demo/229109> (Accessed: 28 January 2017).

Cisco DevNet (2017) *APIC Enterprise Module API Overview*. Available at:

<https://developer.cisco.com/docs/apic-em/#overview> (Accessed: 17 December 2017).

Cisco DevNet GitHub (2015) *OpenDaylight BGP and PCEP (Pathman) Apps*. Available at:

<https://github.com/CiscoDevNet/Opendaylight-BGP-Pathman-apps> (Accessed: 28 January 2017).

Cisco DevNet GitHub (2016) *OpenDaylight OpenFlow Manager (OFM) App*. Available at:

<https://github.com/CiscoDevNet/OpenDaylight-Openflow-App> (Accessed: 30 December 2017).

Cisco DevNet GitHub (2016) *OpenDaylight Pathman SR App*. Available at:

<https://github.com/CiscoDevNet/pathman-sr> (Accessed: 28 January 2017).

Cisco Support (2017) *Release Notes for the Catalyst 4500-X Series Switches, Cisco IOS XE 3.10.0E*. Available at:

<https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst4500/release/note/ol-310xe-4500x.html> (Accessed: 3 January 2018).

Comulus Networks (2017) *Cumulus Linux*. Available at:

<https://cumulusnetworks.com/products/cumulus-linux/> (Accessed: 16 December 2017).

Comulus Networks (2017) *Hardware Compatibility List*. Available at:

<https://cumulusnetworks.com/products/hardware-compatibility-list/> (Accessed: 20 December 2017).

Costiser Network Engineering (2014) *SDN Lesson #1 – Introduction to Mininet*. Available at:

<http://costiser.ro/2014/08/07/sdn-lesson-1-introduction-to-mininet/#.WetgOSzqBU->

(Accessed: 1 October 2017).

Costiser Network Engineering (2016) *My SDN Testbed*. Available at:

http://costiser.ro/2016/06/26/my-sdn-testbed/#.WetgcSzqBU_ (Accessed: 1 October 2017).

CPqD GitHub (2018) *OpenFlow 1.3 switch*. Available at:

<https://github.com/CPqD/ofsoftswitch13> (Accessed: 12 January 2018).

Darbha, R. (2017) ‘Throughput Testing and Troubleshooting’ on client’, *Cumulus Networks Knowledge Base*, July 2017. Available at: https://support.cumulusnetworks.com/hc/en-us/articles/216509388-Throughput-Testing-and-Troubleshooting#server_commands (Accessed: 28 October 2017).

Darbha. R. (2017) ‘Throughput Testing and Troubleshooting’, *Cumulus Networks Knowledge Base*, July 2017. Available at: https://support.cumulusnetworks.com/hc/en-us/articles/216509388-Throughput-Testing-and-Troubleshooting#server_commands (Accessed: 24 September 2017).

Dawson, T. (2000) ‘Building High Performance Linux Routers’, *O'Reilly Linux DevCenter Publication*, September 2000. Available at:
<http://www.linuxdevcenter.com/pub/a/linux/2000/09/28/LinuxAdmin.html> (Accessed: 21 October 2017).

Digital Hybrid (2012) *Quality of Service (QoS) -- DSCP TOS CoS Precedence Conversion Chart*. Available at: <https://my.digitalhybrid.com.au/knowledgebase/201204284/Quality-of-Service-QoS----DSCP-TOS-CoS-Precedence-Conversion-Chart.html> (Accessed: 10 December 2017).

Donahue, G., A. (2011) *Network Warrior: Everything You Need to Know That Wasn't on the CCNA Exam*, 2nd Edition. United States: O'Reilly Media Inc.

Duffy, J. (2015) ‘NSA uses OpenFlow for tracking... its network’, *Network World Article*, June 2015. Available at: <https://www.networkworld.com/article/2937787/sdn/nsa-uses-openflow-for-tracking-its-network.html> (Accessed: 31 December 2017).

eBay (2017) *New Cisco FTDI USB to RJ45 RS232 Console Cable (1.8M Fast Worldwide Delivery)*. Available at: http://www.ebay.ie/item/New-Cisco-FTDI-USB-to-RJ45-RS232-Console-Cable-1-8M-Fast-Worldwide-Delivery/201834707501?ssPageName=STRK%3AMEBIDX%3AIT&_trksid=p2057872.m2749.12649 (Accessed: 3 June 2017).

Egmond, P., V. (2007) ‘Network Operating System (NOS)’, *TechTarget*, February 2007. Available at: <http://searchnetworking.techtarget.com/definition/network-operating-system> (Accessed: 5 June 2017).

Ellingwood, J. (2017) ‘How To Install the Apache Web Server on Ubuntu 16.04’, *DigitalOcean Tutorial*, May 2017. Available at:
<https://www.digitalocean.com/community/tutorials/how-to-install-the-apache-web-server-on-ubuntu-16-04> (Accessed: 19 November 2017).

- English, J. (2017) ‘ONOS (Open Network Operating System)’, *TechTarget*, March 2017. Available at: <http://searchsdn.techtarget.com/definition/ONOS-Open-Network-Operating-System> (Accessed: 5 June 2017).
- Enns, R., Bjorklund, M., Schoenwaelder, J., Bierman, A. (2011) ‘Network Configuration Protocol (NETCONF)’, *Internet Engineering Task Force (IETF)*, Request for Comments: 6241, June 2011. Available at: <https://tools.ietf.org/html/rfc6241> (Accessed: 5 June 2017).
- Facebook (2016) *Introducing Backpack: Our second-generation modular open switch*. Available at: <https://code.facebook.com/posts/864213503715814/introducing-backpack-our-second-generation-modular-open-switch/> (Accessed: 17 December 2017).
- Finn, A. (2017) ‘Using a NAT Virtual Switch with Hyper-V’, *Petri IT Knowledge Blog*, March 2017. Available at: <https://www.petri.com/using-nat-virtual-switch-hyper-v> (Accessed: 14 October 2017).
- Fisher, B. (2016) ‘How to: Set up port mirroring in Hyper-V’, *Spiceworks Microsoft Hyper-V Article*, July 2016. Available at: https://community.spiceworks.com/how_to/130782-set-up-port-mirroring-in-hyper-v (Accessed: 19 November 2017).
- Floodlight (2017) *Projects*. Available at: <http://www.projectfloodlight.org/projects/> (Accessed: 13 December 2017).
- Fortier, R. (2017) ‘Software is Eating the Network: Going Native on Network Virtualization’, *VMware Blog*, March 2017 Available at: <https://blogs.vmware.com/networkvirtualization/2017/03/native-vswitch.html/> (Accessed: 7 January 2018).
- FRRouting (2017) *User Guide*. Available at: <https://frrouting.org/user-guide/> (Accessed: 2 December 2017).
- FRRouting (2017) *What’s in your router?* Available at: <https://frrouting.org> (Accessed: 11 November 2017).
- Fryguy (2013) ‘Cisco CSR1000v For Home Labs’, *Fryguy’s Blog ~ A Network Blog by a Network Engineer*, December 2013. Available at: <https://www.fryguy.net/2013/12/27/cisco-csr1000v-for-home-labs/> (Accessed: 7 November 2017).
- FTDI Chip (2017) *D2XX Drivers*. Available at: <http://www.ftdichip.com/Drivers/D2XX.htm> (Accessed: 2 July 2017).
- George, T. (2015) *Development*. Available at: <https://launchpad.net/~teejee2008/+archive/ubuntu/ppa> (Accessed: 22 July 2017).
- Git (2017) *Download for Linux and Unix*. Available at: <https://git-scm.com/download/linux>

(Accessed: 23 July 2017).

GitHub (2017) *Introduction to Mininet*. Available at:

[https://github.com/minenet/mininet/wiki/Introduction-to-Mininet](https://github.com/mininet/mininet/wiki/Introduction-to-Mininet) (Accessed: 2 July 2017).

GitHub (2017) *Mininet Examples*. Available at:

<https://github.com/minenet/mininet/tree/master/examples> (Accessed: 15 October 2017).

GitHub (2017) *Mininet VM Images*. Available at:

<https://github.com/minenet/mininet/wiki/Mininet-VM-Images> (Accessed: 23 July 2017).

Google (2017) *SDN to the public internet*. Available at:

<https://www.blog.google/topics/google-cloud/making-google-cloud-faster-more-available-and-cost-effective-extending-sdn-public-internet-espresso/> (Accessed: 17 December 2017).

Goransson, P. and Black, C. (2014) *Software defined networks: A comprehensive approach*.

United States: Morgan Kaufmann Publishers In.

Gulam, B. (2015) ‘Installing new version of Open vSwitch’, *Mininet Wiki*, February 2015.

Available at: [https://github.com/minenet/mininet/wiki/Installing-new-version-of-Open-vSwitch](https://github.com/mininet/mininet/wiki/Installing-new-version-of-Open-vSwitch) (Accessed: 20 January 2018).

Gupta, S.N. (2013) ‘*Next Generation Networks (NGN)-Future of Telecommunication*’,

International Journal of ICT and Management, 1(1), pp. 32–35.

Hardesty, L. (2017) ‘Cisco Prefers the Word ‘Automation’ to ‘SDN’’, *SDx Central Article*, February 2017. Available at: <https://www.sdxcentral.com/articles/news/cisco-prefers-word-automation-sdn/2017/02/> (Accessed: 11 December 2017).

Havrila, P. (2012) ‘MPLS VPN tutorial with configuration example for HP A-Series (H3C)’, *NetworkGeekStuff*, May 2012. Available at: <http://networkgeekstuff.com/networking/mpls-vpn-tutorial-with-configuration-example-for-hp-a-series-h3c/> (Accessed: 4 June 2017).

Havrila, P. (2015) ‘Tutorial for creating first external SDN application for HP SDN VAN controller – Part 1/3: LAB creation and REST API introduction’, *Network Geek Stuff Tutorial*, May 2015. Available at: <http://networkgeekstuff.com/networking/tutorial-for-creating-first-external-sdn-application-for-hp-sdn-van-controller-part-13-lab-creation-and-rest-api-introduction/> (Accessed: 1 January 2018).

Hebert, S. (2018) *Using Cisco NBAR to Monitor Traffic Protocols on Your Network*.

Available at: <https://slaptijack.com/networking/using-cisco-nbar-to-monitor-traffic-protocols-on-your-network/> (Accessed: 19 March 2017).

Hill, C. and Voit, E. (2015). ‘Embracing SDN in Next Generation Networks’, *Cisco Day at the Movies*, February 2015. Available at:

https://www.cisco.com/c/dam/en_us/solutions/industries/docs/gov/day-at-movies-sdn-v6-2-25-15.pdf (Accessed: 15 December 2017).

Hogg, S. (2014) ‘SDN Security Attack Vectors and SDN Hardening’, *Network World*, October 2014. Available at: <http://www.networkworld.com/article/2840273/sdn/sdn-security-attack-vectors-and-sdn-hardening.html> (Accessed: 11 June 2017).

Hong, L. (2016) ‘Northbound, Southbound, and East/Westbound. What do they mean?’, *Show IP Protocols Blog*, June 2014. Available at:

<http://showipprotocols.blogspot.ie/2014/06/northbound-southbound-and-eastwestbound.html> (Accessed: 5 June 2017).

HPE (2016) *HPE VAN SDN Controller 2.7 REST API Reference*. Available at:

<https://support.hpe.com/hpsc/doc/public/display?docId=c05040230> (Accessed: 28 January 2018).

HPE (2017) *Aruba VAN SDN Controller Software*. Available at:

<https://www.hpe.com/us/en/product-catalog/networking/networking-software/pip.hpe-van-sdn-controller-software.5443866.html> (Accessed: 14 December 2017).

HPE Support (2012) *HP Switch Software OpenFlow Support*. Available at:

https://support.hpe.com/hpsc/doc/public/display?sp4ts.oid=3437443&docLocale=en_US&docId=emr_na-c03170243 (Accessed: 3 January 2018)

Hu, F. (ed.) (2014) *Network innovation through Openflow and SDN: Principles and design*. Boca Raton, FL: Taylor & Francis.

Internet Live Stats (2017) *Number of Internet users*. Available at:

<http://www.internetlivestats.com/internet-users/> (Accessed: 12 February 2017).

iPerf (2017) *Change between iPerf 2.0, iPerf 3.0 and iPerf 3.1*. Available at:

<https://iperf.fr/iperf-doc.php> (Accessed: 18 February 2017).

iPerf (2017) *What is iPerf/iPerf3*. Available at: <https://iperf.fr> (Accessed: 18 February 2017).

Izard, R. (2017) ‘Static Entry Pusher API’, *Floodlight Controller REST API*, June 2017.

Available at:

<https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343518/Static+Entry+Pusher+API> (Accessed: 27 January 2018).

Jakma, P. (2011) ‘Quagga Documentation’, *Quagga Routing Suite*, April 2011. Available at: <http://www.nongnu.org/quagga/docs/docs-info.html> (Accessed: 7 November 2017).

Ji, M. (2016) ‘Install Wireshark 2.2.0 via PPA in Ubuntu 16.04, 14.04’, *Ubuntu Handbook*, September 2016. Available at: <http://ubuntuhandbook.org/index.php/2016/09/install->

wireshark-2-2-0-ppa-ubuntu-16-04/ (Accessed: 23 July 2017).

Ji, M. (2017) ‘Ukuu – Simple Tool to Install the Latest Kernels in Ubuntu / Linux Mint’, *Ubuntu Handbook*, February 2017. Available at:

<http://ubuntuhandbook.org/index.php/2017/02/ukuu-install-latest-kernels-ubuntu-linux-mint/> (Accessed: 22 July 2017).

Johnson, S. (2013) ‘Border Gateway Protocol as a hybrid SDN protocol’, *TechTarget*, May 2013. Available at: <http://searchsdn.techtarget.com/feature/Border-Gateway-Protocol-as-a-hybrid-SDN-protocol> (Accessed: 5 June 2017).

Johnson, S. (2013) ‘Border Gateway Protocol as a hybrid SDN protocol’, *TechTarget*, May 2013. Available at: <http://searchsdn.techtarget.com/feature/Border-Gateway-Protocol-as-a-hybrid-SDN-protocol> (Accessed: 5 June 2017).

Juniper (2016) ‘Configuring MPLS-Based Layer 3 VPNs’, *MPLS Feature Guide for EX4600 Switches*, September 2016. Available at:

https://www.juniper.net/documentation/en_US/junos/topics/example/mpls-qfx-series-vpn-layer3.html (Accessed: 4 June 2017).

Kahn, Z., A. (2016) ‘Project Falco: Decoupling Switching Hardware and Software’, *LinkedIn Engineering Blog*, February 2016. Available at:

<https://engineering.linkedin.com/blog/2016/02/falco-decoupling-switching-hardware-and-software-pigeon> (Accessed: 20 December 2017).

Kear, S. (2011) ‘Iperf Commands for Network Troubleshooting’, *Sam Kear Blog*, May 2011. Available at: <https://www.samkear.com/networking/iperf-commands-network-troubleshooting> (Accessed: 24 September 2017).

Kernel (2017) *IPRoute2*. Available at: <https://www.kernel.org/pub/linux/utils/net/iproute2/> (Accessed: 22 July 2017).

Kernel Newbies (2015) *Linux 4.1*. Available at: https://kernelnewbies.org/Linux_4.1 (Accessed: 22 July 2017).

Kernel Newbies (2017) *Linux 4.12*. Available at: https://kernelnewbies.org/Linux_4.12 (Accessed: 22 July 2017).

Kickstarter (2015) ‘Zodiac FX: The world's smallest OpenFlow SDN switch’, *Northbound Networks Projects*, July 2015. Available at:

<https://www.kickstarter.com/projects/northboundnetworks/zodiac-fx-the-worlds-smallest-openflow-sdn-switch> (Accessed: 7 January 2018).

KickstartSDN (2015) *Add custom flows with dpctl*. Available at: <http://kickstartsdn.com/dpctl/>

(Accessed: 24 September 2017).

Kumar, N. (2016) ‘Using meters to implement QoS in OpenDaylight’, *Talentica Blog*, November 2016. Available at: <https://blog.talentica.com/2016/11/25/using-meters-to-implement-qos-in-opendaylight/> (Accessed: 20 January 2018).

Lakshman, U. and Lobo, L. (2006) *MPLS Configuration on Cisco IOS Software*. Cisco Press. Available at: <http://flylib.com/books/en/2.686.1/> (Accessed: 14 November 2017).

Lantz, B., Heller, B., McKeown, N. (2010) ‘A network in a laptop: rapid prototyping for software-defined networks’, *Hotnets-IX Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Article No. 19, pp. 19. Available at: <http://dl.acm.org/citation.cfm?id=1868466> (Accessed: 2 July 2017).

Launchpad (2017) *Wireshark stable releases*. Available at: <https://launchpad.net/~wireshark-dev/+archive/ubuntu/stable> (Accessed: 23 July 2017).

Lawson, S. (2013) ‘Will software-defined networking kill network engineers beloved CLI?’, *Computerworld*, August 2013. Available at:

<http://www.computerworld.com/article/2484358/it-careers/will-software-defined-networking-kill-network-engineers--beloved-cli-.html> (Accessed: 5 June 2017).

Le Faucheur, F., Wu, L., Davie, B., Davari, S., Vaananen, P., Krishnan, R., Cheval, P., Heinanen, J. (2013) ‘Multi-Protocol Label Switching (MPLS) Support of Differentiated Services’, *RFC 3270*, May 2002. Available at: <https://tools.ietf.org/html/rfc3270> (Accessed: 14 November 2017).

LeClerc, M. (2016) ‘PSNC AND NOVIFLOW TEAM UP TO DEMO IOT OVER SDN FOR SMART CITIES AT TNC2016 IN PRAGUE’, *NoviFlow*, June 2016. Available at: <https://noviflow.com/tnc2016demo/> (Accessed: 25 February 2018).

Leu, J.R. (2013) *MPLS for Linux*. Available at: <https://sourceforge.net/projects/mpls-linux/> (Accessed: 12 February 2017).

Levy, S. (2012) ‘GOING WITH THE FLOW: GOOGLE'S SECRET SWITCH TO THE NEXT WAVE OF NETWORKING’, *Wired Article*, April 2012. Available at: <https://www.wired.com/2012/04/going-with-the-flow-google/> (Accessed: 19 December 2017).

Lewis, C. and Pickavance, S. (2006) ‘Implementing Quality of Service Over Cisco MPLS VPNs’, *Cisco Press Article*, May 2006. Available at: <http://www.ciscopress.com/articles/article.asp?p=471096&seqNum=6> (Accessed: 29 October 2017).

- Linkletter, B. (2016) ‘How to build a network of Linux routers using quagga’, *Open-Source Routing and Network Simulation*, June 2016. Available at:
<http://www.brianlinkletter.com/how-to-build-a-network-of-linux-routers-using-quagga/>
(Accessed: 10 November 2017).
- LinTut (2015) *How to install DHCP server on Ubuntu 14.04/14.10/15.04*. Available at:
<https://lintut.com/how-to-install-dhcp-server-on-ubuntuserver/> (Accessed: 15 October 2017).
- McCauley, M. (2015) ‘POX Wiki’, *Welcome to OpenFlow at Stanford*, March 2015.
Available at: <https://openflow.stanford.edu/display/ONL/POX+Wiki> (Accessed: 30 December 2017).
- McLendon, W. (2017) ‘Building FRR on Ubuntu 16.04LTS from Git Source’, *GitHub FRR Documentation*, October 2017. Available at:
https://github.com/FRRouting/frr/blob/master/doc/Building_FRR_on_Ubuntu1604.md
(Accessed: 11 November 2017).
- McNickle, M. (2014) ‘Five SDN protocols other than OpenFlow’, *TechTarget*, August 2014.
Available at: <http://searchsdn.techtarget.com/news/2240227714/Five-SDN-protocols-other-than-OpenFlow> (Accessed: 5 June 2017).
- Mert, A. (2016) ‘GNS3 LAB: CONFIGURING MPLS VPN BACKBONE TO INTERCONNECT TWO DATACENTERS’, *RouterCrash.net Blog*, May 2016. Available at:
<http://www.routercrash.net/gns3-lab-configuring-mpls-vpn-backbone-to-interconnect-two-datacenters/> (Accessed: 7 November 2017).
- MGEN (2017) *MGEN User's and Reference Guide Version 5.0*. Available at:
https://downloads.pf.itd.nrl.navy.mil/docs/mgen/mgen.html#_Transmission_Events
(Accessed: 30 October 2017).
- Microsoft (2017) *Microsoft volume licensing*. Available at: <https://www.microsoft.com/en-us/licensing/product-licensing/windows-server-2012-r2.aspx#tab=4> (Accessed: 18 February 2017).
- Milestone of System Engineer (2017) *Configuration of MPLS-VPN (Super Backbone and Sham-Link)*. Available at: <http://milestone-of-se.nesuke.com/en/nw-advanced/mpls-vpn/mpls-vpn-configuration/> (Accessed: 4 December 2017).
- Miller, R. (2017) ‘Cisco scoops up yet another cloud company, acquiring SD-WAN startup Viptela for \$610M’, *Tech Crunch Article*, May 2017. Available at:
<https://techcrunch.com/2017/05/01/cisco-scoops-up-yet-another-cloud-company-acquiring-sd-wan-startup-viptela-for-610-m/> (Accessed: 16 December 2017).

- Millman, R. (2015) ‘How to secure the SDN infrastructure’, *Computer Weekly*, March 2015. Available at: <http://www.computerweekly.com/feature/How-to-secure-the-SDN-infrastructure> (Accessed: 11 June 2017).
- Mininet (2017) *Download/Get Started With Mininet*. Available at: <http://mininet.org/download/> (Accessed: 23 July 2017).
- Mininet (2017) *Mininet Python API Reference Manual*. Available at: <http://mininet.org/api/index.html> (Accessed: 2 July 2017).
- Mininet GitHub (2017) *Mininet VM Images*. Available at: <https://github.com/mininet/mininet/wiki/Mininet-VM-Images> (Accessed: 2 July 2017).
- Mishra, A.K. and Sahoo, A. (2007) ‘*S-OSPF: A Traffic Engineering Solution for OSPF Based Best Effort Networks*’, Piscataway, NJ: IEEE, pp. 1845–1849.
- Mitchell, S. (2014) ‘The impacts of software defined networking (SDN) and network function virtualization (NFV) on the Stack, Part 1’, *Cohesive Blog*, May 2014. Available at: <https://cohesive.net/2014/05/the-impacts-of-software-defined.html> (Accessed: 5 June 2017).
- Molenaar, R. (2017) ‘Multiprotocol BGP (MP-BGP) Configuration’, *Network Lessons*, May 2017. Available at: <https://networklessons.com/bgp/multiprotocol-bgp-mp-bgp-configuration/> (Accessed: 14 November 2017).
- MPLS Info (2017) *MPLS Architecture*. Available at: <http://www.mplsinfo.org/architecture.html> (Accessed: 4 June 2017).
- Network Sorcery (2012) *RFC Sourcebook*. Available at: <http://www.networksorcery.com/enp/> (Accessed: 25 December 2017).
- Networking Forum (2009) *Configuring Basic MPLS TE Tunnels*. Available at: <http://www.networking-forum.com/blog/?p=145> (Accessed: 3 December 2017).
- Nichols, K., Blake, S., Baker, F., Black, D. (1998) ‘Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers’, *RFC 2474*, December 1998. Available at: <https://www.ietf.org/rfc/rfc2474.txt> (Accessed: 14 November 2017).
- Northbound Networks (2016) *FLOW MAKER DELUXE*. Available at: <https://www.hpe.com/us/en/product-catalog/networking/networking-software/pip.hpe-van-sdn-controller-software.5443866.html> (Accessed: 15 December 2017).
- NOX Repo GitHub (2017) *POX*. Available at: <https://github.com/noxrepo/pox> (Accessed: 27 December 2017).
- Nuage Networks (2015) *Evolution of Wide Area Networking*. Available at: http://www.nuagenetworks.net/wp-content/uploads/2015/08/PR1506012099EN_NN-

[VNS Enterprise CaseStudy.pdf](#) (Accessed: 24 December 2017).

OmniSecu (2017) *How to use PuTTY Terminal Emulator to configure, monitor or manage a Cisco Router or Switch*. Available at: <http://www.omnisecu.com/cisco-certified-network-associate-ccna/how-to-use-putty-to-configure-or-monitor-a-cisco-router-or-switch.php> (Accessed: 2 July 2017).

ONF (2016) *Special Report: OpenFlow and SDN – State of the Union*. Available at: <https://www.opennetworking.org/wp-content/uploads/2013/05/Special-Report-OpenFlow-and-SDN-State-of-the-Union-B.pdf> (Accessed: 26 December 2017).

ONF (2017) *Software-Defined Networking (SDN) Definition*. Available at: <https://www.opennetworking.org/sdn-definition/> (Accessed: 12 December 2017).

ONOS (2015) *CORD: The Central Office Re-architected as a Datacenter*. Available at: http://onosproject.org/wp-content/uploads/2015/06/PoC_CORD.pdf (Accessed: 18 December 2017).

ONOS (2017) *Features*. Available at: <https://onosproject.org/features/> (Accessed: 13 December 2017).

Open Networking Foundation (2013) *OpenFlow Switch Specification Version 1.3.2*. Available at: <https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-spec-v1.3.2.pdf> (Accessed: 12 February 2017).

Open Networking Foundation (2013) *OpenFlow Switch Specification Version 1.4.0*. Available at: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf> (Accessed: 31 December 2017).

Open Networking Foundation (2015) *OpenFlow Switch Specification Version 1.5.1*. Available at: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf> (Accessed: 12 February 2017).

Open Networking Foundation (2015) *The Benefits of Multiple Flow Tables and TTPs*. Available at: https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/TR_Multiple_Flow_Tables_and_TTPs.pdf (Accessed: 7 January 2018).

Open Networking Foundation (2017) *ONF Overview*. Available at: <https://www.opennetworking.org/about/onf-overview> (Accessed: 4 June 2017).

Open vSwitch (2016) *Open vSwitch Manual*. Available at: <http://openvswitch.org/support/dist-docs-2.5/ovs-ofctl.8.txt> (Accessed: 29 December 2017).

Open vSwitch (2018) *Open vSwitch Download*. Available at:

<http://openvswitch.org/download/> (Accessed: 20 January 2018).

OpenCORD (2017) *Specs*. Available at: <https://opencord.org/specs> (Accessed: 18 December 2017).

OpenDaylight (2017) *Current Release*. Available at: <https://www.opendaylight.org/what-we-do/current-release> (Accessed: 12 December 2017).

OpenFlow (2009) *OpenFlow Switch Specification Version 1.0.0*. Available at: <http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf> (Accessed: 12 February 2017).

OpenFlow (2011) *Create OpenFlow network with multiple PCs/NetFPGAs*. Available at: <http://archive.openflow.org/wp/Deploy-labsetup/> (Accessed: 23 July 2017).

OpenFlow (2011) *View source for Ubuntu Install*. Available at: http://archive.openflow.org/wk/index.php?title=Ubuntu_Install&action=edit (Accessed: 18 February 2017).

OpenManiak (2008) *Quagga Tutorial*. Available at: <https://openmaniak.com/quagga.php> (Accessed: 7 November 2017).

OpenManiak (2010) *Iperf Tutorial*. Available at: <https://openmaniak.com/iperf.php> (Accessed: 18 February 2017).

OpenSourceRouting (2017) ‘RFC Compliance Test Results for latest code’, *GitHub FRR Wiki*, April 2017. Available at:

https://raw.githubusercontent.com/wiki/FRRouting/frr/RFC_Compliance_Results/LDP_exended_results.pdf (Accessed: 11 November 2017).

OpenSwitch (2017) *Hardware*. Available at: <https://www.openswitch.net/hardware/> (Accessed: 23 December 2017).

O'Reilly, J. (2014) ‘SDN Limitations’, *Network Computing*, October 2014. Available at: <https://www.networkcomputing.com/networking/sdn-limitations/241820465> (Accessed: 5 June 2017).

Paessler (2017) *SOFTWARE DEFINED NETWORKING & PRTG*. Available at: <https://www.paessler.com/software-defined-networking> (Accessed: 5 June 2017).

Partsenidis, C. (2011) ‘MPLS VPN tutorial’, *TechTarget*, June 2011. Available at: <http://searchenterprisewan.techtarget.com/tutorial/MPLS-VPN-tutorial> (Accessed: 4 June 2017).

Penguin Computing (2017) *Arctica Network Switch Features*. Available at: <https://www.penguincomputing.com/products/network-switches/> (Accessed: 20 December 2017).

2017).

Pepelnjak, I. (2007) ‘10 MPLS traffic engineering myths and half truths’, *SearchTelecom TechTarget*, October 2007. Available at: <http://searchtelecom.techtarget.com/tip/10-MPLS-traffic-engineering-myths-and-half-truths> (Accessed: 9 December 2017).

Perkin, R. (2016) ‘MPLS Tutorial – MPLS Configuration Step by Step’, *Roger Perking Networking Tutorials from CCIE #50038*, June 2016. Available at:
http://www.rogerperkin.co.uk/ccie/mpls/cisco-mpls-tutorial/?doing_wp_cron=1510148437.3483409881591796875000 (Accessed: 8 November 2017).

Pica8 (2017) *PicOS*. Available at: <http://www.pica8.com/products/picos> (Accessed: 26 December 2017).

Picket, G. (2015) ‘Abusing Software Defined Networks’, *DefCon 22 Hacking Conference*, August 2015, Rio Hotel & Casino in Last Vegas. Available at:
<https://www.defcon.org/html/links/dc-archives/dc-22-archive.html> (Accessed: 10 January 2018).

PR Newswire (2015) *Extreme Networks Announces Real-World SDN Solutions for Enterprise-Scale Customers featuring OpenDaylight Integration*. Available at:
<https://www.prnewswire.com/news-releases/extreme-networks-announces-real-world-sdn-solutions-for-enterprise-scale-customers-featuring-opendaylight-integration-300073052.html> (Accessed: 7 January 2018).

Prabhu, R. (2016) ‘MPLS tutorial’, *Proceedings of NetDev 1.1: The Technical Conference on Linux Networking*, February 2016. Available at:
<http://www.netdevconf.org/1.1/proceedings/slides/prabhu-mpls-tutorial.pdf> (Accessed: 22 July 2017).

PuTTY (2017) *Download PuTTY: latest release (0.69)*. Available at:
<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html> (Accessed: 2 July 2017).

Quagga (2017) *Quagga Routing Suite*. Available at: <http://www.nongnu.org/quagga/> (Accessed: 21 December 2017).

Rao, S. (2016) ‘How to install & use iperf & jperf tool’, *Linux Thrill Tech Blog*, April 2016. Available at: <http://linuxthrill.blogspot.ie/2016/04/how-to-install-use-iperf-jperf-tool.html> (Accessed: 18 February 2017).

Reber, A. (2015) ‘On the Scalability of the Controller in Software-Defined Networking’, *MSc in Computer Science*, University of Liege, Belgium. Available at:

<http://www.student.montefiore.ulg.ac.be/~agirmanr/src/tfe-sdn.pdf> (Accessed: 5 June 2017).

REST API Tutorial by RESTfulAPI.net (2018) *HTTP Methods*. Available at:

<https://restfulapi.net/http-methods/> (Accessed: 9 January 2018).

Roggy Blog (2009) *My First MPLS blog*. Available at:

<http://roggyblog.blogspot.ie/search?q=mpls> (Accessed: 2 December 2017).

Rogier, B. (2016) ‘NETWORK PERFORMANCE: LINKS BETWEEN LATENCY, THROUGHPUT AND PACKET LOSS’, *Performance Vision Blog*, May 2016. Available at: <http://blog.performancevision.com/eng/earl/links-between-latency-throughput-and-packet-loss> (Accessed: 29 September 2017).

Rohilla, N. (2016) ‘[ovs-dev] Openflow 1.5 - Enable setting all pipeline fields in packet-out [EXT-427]’, *OVS Development Archives*, February 2016. Available at:

<https://mail.openvswitch.org/pipermail/ovs-dev/2016-February/310289.html> (Accessed: 31 December 2017).

Rosen, E., Viswanathan, A., Callon, R. (2001) ‘*Multiprotocol Label Switching Architecture*’, Internet Engineering Task Force, January 2001. Available at:

<https://tools.ietf.org/html/rfc3031.html> (Accessed: 12 February 2017).

Rouse, M. (2013) ‘OVSDB (Open vSwitch Database Management Protocol)’, *TechTarget*, September 2013. Available at: <http://searchsdn.techtarget.com/definition/OVSDB-Open-vSwitch-Database-Management-Protocol> (Accessed: 5 June 2017).

Russell, S. (2015) ‘MPLS testbed on Ubuntu Linux with kernel 4.3’, *Sam Russell Central Blog*, December 2015. Available at: <https://pieknywidok.blogspot.ie/2015/12/mpls-testbed-on-ubuntu-linux-with.html?showComment=1499536329246#c8233433417428817510> (Accessed: 22 July 2017).

Russello, J. (2016) ‘Southbound vs. Northbound SDN: What are the differences?’, *WEb Werks Blog*, June 2016. Available at: <http://blog.webwerks.in/data-centers-blog/southbound-vs-northbound-sdn-what-are-the-differences> (Accessed: 5 June 2017).

Ryu (2014) *RYU the Network Operating System (NOS) Documentation*. Available at:

<http://ryu.readthedocs.io/en/latest/> (Accessed: 13 December 2017).

Ryu SDN Framework Community (2017) *WHAT'S RYU?* Available at:

<http://osrg.github.io/ryu/index.html> (Accessed: 30 December 2017).

Salisbury, B. (2013) ‘OpenFlow: SDN Hybrid Deployment Strategies’, *Brent Salisbury's Blog*, January 2013. Available at: <http://networkstatic.net/openflow-sdn-hybrid-deployment-strategies/> (Accessed: 5 June 2017).

- Sam Russell Central (2015) *MPLS testbed on Ubuntu Linux with kernel 4.3*. Available at: <http://www.samrussell.nz/2015/12/mpls-testbed-on-ubuntu-linux-with.html> (Accessed: 19 August 2017).
- Sanchez-Monge, A. and Szarkowicz, K.G. (2015) *MPLS in the SDN era: Interoperable scenarios to make networks scale to new services*. United States: O'Reilly Media Inc.
- Sanfilippo, S. (2014) 'hping3 Package Description', *Kali Tools*, February 2014. Available at: <https://tools.kali.org/information-gathering/hping3> (Accessed: 27 November 2017).
- Schoenwaelder, J. (2003) 'Overview of the 2002 IAB Network Management Workshop', *RFC 3535*, May 2003. Available at: <https://tools.ietf.org/html/rfc3535> (Accessed: 7 January 2018).
- ScienceLogic (2017) *See How Your Software-Defined Network Is Performing*. Available at: <https://www.scienceologic.com/product/technologies/software-defined-networking> (Accessed: 5 June 2017).
- SDN Central (2016) *2016 SDN Controller Landscape Is there a Winner?* Available at: <https://events.static.linuxfound.org/sites/events/files/slides/3%20202016%20ONS%20ONF%20Mkt%20Opp%20Controller%20Landscape%20RChua%20Mar%2014%202016.pdf> (Accessed: 19 December 2017).
- SDN Hub (2014) *All-in-one SDN App Development Starter VM*. Available at: <http://sdnhub.org/tutorials/sdn-tutorial-vm/> (Accessed: 28 December 2017).
- SDxCentral (2017) *What is Software-Defined WAN (or SD-WAN or SDWAN)?* Available at: <https://www.sdxcentral.com/sd-wan/definitions/software-defined-sdn-wan/> (Accessed: 24 December 2017).
- SevOne (2017) *Software Defined Network Monitoring*. Available at: <https://www.sevone.com/solutions/software-defined-network-monitoring> (Accessed: 5 June 2017).
- Sharewood, R. (2014) *Tutorial: Whitebox/Bare Metal Switches*. Available at: <http://www.bigswitch.com/sites/default/files/presentations/onug-baremetal-2014-final.pdf> (Accessed: 11 December 2017).
- Sharp, D. (2017) 'ldpd-basic-test-setup.md', *GitHub FRR Documentation*, January 2017. Available at: <https://github.com/FRRouting/frr/blob/master/doc/ldpd-basic-test-setup.md> (Accessed: 12 November 2017).
- Shrestha, S. (2014) 'Connecting Mininet Hosts to Internet', *Sandesh Shrestha's Blog*, December 2014. Available at: <http://sandeshshrestha.blogspot.ie/2014/12/connecting-mininet.html>

[hosts-to-internet.html](#) (Accessed: 14 October 2017).

Simpkins, A. (2015) ‘Facebook Open Switching System ("FBOSS") and Wedge in the open’, *Facebook Article*, March 2015. Available at:

<https://code.facebook.com/posts/843620439027582/facebook-open-switching-system-fboss-and-wedge-in-the-open/> (Accessed: 21 December 2017).

SK (2017) ‘How To Create Files Of A Certain Size In Linux’, *OSTechNix Trouble Shooting*, July 2017. Available at: <https://www.ostechnix.com/create-files-certain-size-linux/> (Accessed: 19 November 2017).

Slattery, T. (2014) ‘QoS in an SDN’, *No Jitter*, May 2014. Available at:

<https://www.nojitter.com/post/240168323/qos-in-an-sdn> (Accessed: 7 January 2018).

Smith, B.R. and Aceves, C.L. (2008) *Best Effort Quality-of-Service*, St. Thomas, U.S. Virgin Islands: IEEE.

SnapRoute (2017) *Welcome to FlexSwitch from SnapRoute*. Available at:

<http://docs.snaproute.com/index.html> (Accessed: 21 December 2017).

Sneddon, J. (2016) ‘Ubuntu 14.04.4 LTS Released, This Is What’s New’, *OMG Ubuntu*, February 2016. Available at: <http://www.omgubuntu.co.uk/2016/02/14-04-4-lts-released-whats-new> (Accessed: 23 July 2017).

Sogay, S. (2013) ‘MPLS VPN QoS with GNS3 and Virtualbox’, *Baba AweSam Blog*, August 2013. Available at: <https://babaawesam.com/2013/08/07/mpls-vpn-qos-with-gns3-and-virtualbox/> (Accessed: 14 November 2017).

SONiC Azure GitHub (2017) *Features and Roadmap*. Available at:

<https://github.com/Azure/SONiC/wiki/Features-and-Roadmap> (Accessed: 20 December 2017).

SourceForge (2016) *SDN Toolkit*. Available at: <https://sourceforge.net/projects/sdn-toolkit/> (Accessed: 10 January 2018).

Stack Overflow (2015) *iproute2 commands for MPLS configuration*. Available at:

<https://stackoverflow.com/questions/31926342/iproute2-commands-for-mpls-configuration> (Accessed: 22 July 2017).

SysTutorials (2017) *avahi-daemon (8) - Linux Man Page*. Available at:

<https://www.systutorials.com/docs/linux/man/8-avahi-daemon/> (Accessed: 23 July 2017).

Taleqani, M. (2017) ‘BGP receives VPNv4 updates but not sending VPNv4 updates #651’, *FRRouting Issues*, June 2017. Available at: <https://github.com/FRRouting/frr/issues/651> (Accessed: 2 December 2017).

- Tucny, D. (2013) ‘Tucny Blog’, *DSCP & TOS*, April 2013. Available at: <https://www.tucny.com/Home/dscp-tos> (Accessed: 4 February 2018).
- U.S. Naval Research Laboratory (2017) *Multi-Generator (MGEN)*. Available at: <https://www.nrl.navy.mil/itd/ncs/products/mgen> (Accessed: 18 February 2017).
- Ubuntu (2017) *Package: mininet (2.1.0-0ubuntu1) [universe]*. Available at: <https://packages.ubuntu.com/trusty/net/mininet> (Accessed: 2 July 2017).
- Ubuntu (2017) *ReleaseNotes*. Available at: https://wiki.ubuntu.com/XenialXerus/ReleaseNotes#Official_flavour_release_notes (Accessed: 18 February 2017).
- Unix Stack Exchange (2014) *How do I set my DNS when resolv.conf is being overwritten?* Available at: <https://unix.stackexchange.com/questions/128220/how-do-i-set-my-dns-when-resolv-conf-is-being-overwritten> (Accessed: 15 October 2017).
- Unix Stack Exchange (2016) *How to set the Default gateway*. Available at: <https://unix.stackexchange.com/questions/259045/how-to-set-the-default-gateway> (Accessed: 19 August 2017).
- VirtualBox (2017) *About VirtualBox*. Available at: <https://www.virtualbox.org/wiki/VirtualBox> (Accessed: 2 July 2017).
- Vissicchio, S., Vanbever, L., Bonaventure, O. (2014) ‘Opportunities and Research Challenges of Hybrid Software Defined Networks’, *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 70-75. Available at: <http://dl.acm.org/citation.cfm?id=2602216> (Accessed: 5 June 2017).
- VMware (2017) *VMWARE NSX Datasheet*. Available at: <https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/products/nsx/vmware-nsx-datasheet.pdf> (Accessed: 15 December 2017).
- VyOS (2017) *Proposed Enhancements*. Available at: https://wiki.vyos.net/wiki/Proposed_enhancements (Accessed: 25 September 2017).
- Wang, M., Chen, L., Chi, P., Lei, C. (2017) ‘SDUDP: A Reliable UDP-based Transmission Protocol over SDN’, *IEEE Access*, vol. PP, no. 99, pp. 1-13. Available at: <http://ieeexplore.ieee.org/document/7898398/> (Accessed: 5 June 2017).
- WAWIT (2010) ‘Generate traffic using mgen’, *Quality of Service Blog*, May 2010. Available at: <http://qos.wawit.pl/2010/05/generate-traffic-using-mgen/> (Accessed: 30 October 2017).
- Westphal, R. (2016) ‘ldpd basic test setup’, *GitHub Wiki*, December 2016. Available at: <https://github.com/rwestphal/quagga-ldpd/wiki/Ldpd-basic-test-setup> (Accessed: 7 November 2016).

2017).

Westphal, R. (2016) ‘SimonCZW’s ldpd test setup’, *Github Wiki*, September 2016. Available at: <https://github.com/rwestphal/quagga-ldpd/wiki/SimonCZW's-ldpd-test-setup> (Accessed: 7 November 2017).

Wilson, O. (2016) ‘How To Install Flex and Bison Under Ubuntu’, *CCM*, September 2016. Available at: <http://ccm.net/faq/30635-how-to-install-flex-and-bison-under-ubuntu> (Accessed: 22 July 2017).

Wireshark (2014) *Platform-Specific information about capture privileges*. Available at: https://wiki.wireshark.org/CaptureSetup/CapturePrivileges#Other_Linux_based_systems_or_other_installation_methods (Accessed: 23 July 2017).

Wireshark (2016) *OpenFlow (openflow)*. Available at: <https://wiki.wireshark.org/OpenFlow> (Accessed: 23 July 2017).

Wireshark (2017) *What is a network protocol analyzer?*. Available at: <https://wireshark.com> (Accessed: 23 July 2017).

Yazici, V. (2013) *Discovery in Software-Defined Networks*. Available at: <http://vlkan.com/blog/post/2013/08/06/sdn-discovery/> (Accessed: 26 December 2017).

Yuri (2012) '#58 Spurious 'Connection refused' on client', *SourceForge Iperf Bugs*, November 2012. Available at: <https://sourceforge.net/p/iperf/bugs/58/> (Accessed: 28 October 2017).

Yusuke, I. (2016) ‘Problem when processing traffic based on OpenFlow 1.5’, *Ryu Development Mailing List*, July 2016. Available at: <http://ryu-devel.narkive.com/l8ocILWz/problem-when-processing-traffic-based-on-openflow-1-5> (Accessed: 31 December 2017).

Zhihao, S. and Wolter, K. (2016) ‘Delay Evaluation of OpenFlow Network Based on Queueing Model’, *Research Gate Publication*, August 2016. Available at: https://www.researchgate.net/publication/306397961_Delay_Evaluation_of_OpenFlow_Network_Based_on_Queueing_Model (Accessed: 19 March 2018).