



TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

KHATRI VIKRAMAJEET
**ANALYSIS OF OPENFLOW PROTOCOL IN LOCAL AREA NET-
WORKS**

Master of Science Thesis

Examiners: Professor Jarmo Harju
M.Sc. Matti Tiainen

Examiner and topic approved in the
Computing and Electrical Engineer-
ing Faculty Council meeting on
6.2.2013

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY

Degree Programme in Information Technology

KHATRI, VIKRAMAJEET: Analysis of OpenFlow Protocol in Local Area Networks

Master of Science Thesis, 62 pages, 4 Appendix pages

August 2013

Major: Communication Networks & Protocols

Examiners: Professor Jarmo Harju, M.Sc. Matti Tiainen

Keywords: Software Defined Networking, SDN, OpenFlow, LAN

The traditional networking infrastructure is still static in nature due to its complexity, vendor dependence and QoS requirements. Software Defined Networking (SDN) is aimed at surpassing the limits of traditional networking infrastructure and making it a dynamic network. In SDN, for a single change in network, the network configurations are changed only at central or some specific controller(s) rather than touching individual network devices.

One of the SDN protocols ‘OpenFlow’ is a normal application layer protocol, which is encapsulated inside TCP, IPv4 and Ethernet format. In this thesis, the integration and benefits of OpenFlow protocol in LAN environment have been analyzed. OpenFlow controller is the heart of the OpenFlow network, and in centralized hierarchy it poses a single point of failure and risk of DoS attacks. In an OpenFlow network, the switch follows its flow table to make forwarding decisions and rejects its traditional forwarding table. The flows must be carefully configured, since a mismatch leads to packets being forwarded to OpenFlow controller that may decide to broadcast packets, and lead to a drastic reduced throughput from 941 Mbps to approx. 340 Kbps in Gigabit network.

All the flows were manually configured and installed to switches via OpenFlow controller making the network again static in nature. In order to handle the dynamic network, an automation framework can be developed that adds or remove flows respectively. The flow concept can be interpreted as avoiding routers in a network, but in fact flows do not override the features of a router.

The benefits of OpenFlow in LANs include an independent and programmable control over the network. The conducted experiments have demonstrated its successful integration inside a single subnet in LANs. However, a full integration with LANs could not be achieved due to the lack of support for layer 3 protocols and OpenFlow’s slow integration into hardware. In addition, the deployment models are not well-suited to the service providers. OpenFlow protocol is more suited to the data centers or backbone networks to handle growing data, and smaller networks like campus area networks to isolate the research traffic from the network traffic.

PREFACE

This Master of Science thesis was compiled in the Department of Communication Engineering at Tampere University of Technology (TUT), Tampere. I am very thankful to HP Finland Oy for offering HP switches for conducting experiments.

I would like to express my deepest gratitude to my thesis supervisor, Professor Jarmo Harju from TUT for his guidance, patience and invaluable advice throughout the thesis. He gave me an opportunity to explore more about the OpenFlow protocol. I would also like to thank Matti Tiainen for his invaluable thoughts, support and guidance in the experiments throughout the thesis.

I would like to thank my parents for giving me high moral values throughout my life. Lastly, big thanks to all my friends and people whom I have met in my life and learnt something new and valuable from them.

Tampere, June 2013

Vikramajeet Khatri

TABLE OF CONTENTS

| | |
|---|------------|
| Abstract | II |
| Preface | III |
| Abbreviations and notations | VI |
| 1. Introduction | 1 |
| 2. Software Defined Networking | 2 |
| 2.1. Introduction | 2 |
| 2.2. Traditional networking technologies..... | 3 |
| 2.3. Limitations of traditional networking technologies | 4 |
| 2.4. Architecture for SDN | 5 |
| 2.5. Models of deployment for SDN | 6 |
| 2.6. Data center networking and SDN..... | 7 |
| 2.7. Scalability in SDN..... | 8 |
| 2.7.1. Controller scalability and load reduction | 9 |
| 2.7.2. Flow overhead..... | 9 |
| 2.7.3. Self-healing | 10 |
| 2.8. Network management in SDN | 11 |
| 3. OpenFlow | 13 |
| 3.1. Introduction | 13 |
| 3.2. Architecture..... | 14 |
| 3.2.1. OpenFlow enabled switch..... | 15 |
| 3.2.2. Controller | 17 |
| 3.3. Flow types | 17 |
| 3.4. Working methodology..... | 18 |
| 3.4.1. Message types exchanged between switch and controller | 19 |
| 3.4.2. Connection establishment between switch and controller | 19 |
| 3.4.3. Connection between hosts on OpenFlow network..... | 20 |
| 3.5. Packet Format..... | 21 |
| 3.6. OpenFlow projects | 23 |
| 3.6.1. Open vSwitch..... | 23 |
| 3.6.2. FlowVisor..... | 25 |
| 3.6.3. LegacyFlow..... | 26 |
| 3.6.4. RouteFlow | 27 |
| 3.6.5. OpenFlow MPLS | 28 |
| 3.7. Progress and current deployment of OpenFlow | 29 |
| 4. Experiments | 31 |
| 4.1. Laboratory setup..... | 31 |
| 4.1.1. HP Switches | 31 |
| 4.1.2. Linux tools and utilities..... | 34 |

| | | |
|-----------|--|-----------|
| 4.1.3. | Floodlight Controller..... | 34 |
| 4.2. | Experiment 1: Basic setup inside the same subnet..... | 36 |
| 4.2.1. | Flow entries..... | 37 |
| 4.2.2. | Results..... | 38 |
| 4.3. | Experiment 2: Verifying <i>modify actions</i> in a flow..... | 39 |
| 4.3.1. | Flow entries..... | 40 |
| 4.3.2. | Results..... | 41 |
| 4.4. | Experiment 3: VLANs in OpenFlow network..... | 41 |
| 4.4.1. | Flow entries..... | 42 |
| 4.4.2. | Results..... | 42 |
| 4.5. | Experiment 4: Fail-safe in case of controller failure..... | 43 |
| 4.5.1. | Flow entries..... | 44 |
| 4.5.2. | Results..... | 45 |
| 4.6. | Experiment 5: OpenFlow in hybrid mode..... | 46 |
| 4.6.1. | Flow entries..... | 46 |
| 4.6.2. | Results..... | 47 |
| 4.7. | Experiment 6: MAC based forwarding..... | 47 |
| 4.7.1. | Flow entries..... | 47 |
| 4.7.2. | Results..... | 48 |
| 4.8. | Experiment 7: VLANs with a router..... | 49 |
| 4.8.1. | Flow entries..... | 50 |
| 4.8.2. | Results..... | 50 |
| 4.9. | Experiment 8: VLANs with RouteFlow..... | 51 |
| 4.9.1. | Flow entries..... | 52 |
| 4.9.2. | Results..... | 52 |
| 4.10. | Discussion on results..... | 53 |
| 5. | Conclusions and future work..... | 55 |
| 5.1. | Conclusions..... | 55 |
| 5.2. | Future work..... | 57 |
| | References..... | 58 |
| | Appendix A: Comparison between switches..... | 63 |
| | Appendix B: Comparison between controllers..... | 64 |
| | Appendix C: OpenFlow protocol in Wireshark..... | 65 |
| | Appendix D: Features of OpenFlow in HP..... | 66 |
| D.1 | Features available for OpenFlow..... | 66 |
| D.2 | Features not available for OpenFlow..... | 66 |
| D.3 | Features not interoperable with OpenFlow..... | 66 |

ABBREVIATIONS AND NOTATIONS

| | |
|---------------|---|
| API | Application Programming Interface |
| ARP | Address Resolution Protocol |
| ASIC | Application Specific Integrated Circuit |
| BGP | Border Gateway Protocol |
| CAPEX | CAPital EXpenditure |
| CENTOS | Community ENTerprise Operating System |
| CLI | Command Line Interface |
| CPU | Central Processing Unit |
| DHCP | Dynamic Host Control Protocol |
| DNS | Domain Name System |
| DoS | Denial of Service |
| DPID | DataPath Identifier |
| DVD | Digital Versatile Disc |
| FEC | Forwarding Equivalence Class |
| FP7 | Framework Programme 7 |
| FPGA | Field Programmable Gate Array |
| FRP | Functional Reactive Programming |
| GENI | Global Environment for Network Innovations |
| GPL | General Public License |
| GRE | Generic Routing Encapsulation |
| GUI | Graphical User Interface |
| HP | Hewellett & Packard |
| HTTP | Hyper Text Transfer Protocol |
| IBM | International Business Machines |
| ICMP | Internet Control Message Protocol |
| ICT | Information and Communication Technology |
| IEEE | Institute of Electrical and Electronics Engineers |
| IP | Internet Protocol |

| | |
|---------------|---|
| JSON | JavaScript Object Notation |
| JunOS | Juniper Operating System |
| KVM | Kernel based Virtual Machine |
| LAN | Local Area Network |
| LLDP | Link Layer Discovery Protocol |
| LSP | Label Switched Path |
| LSR | Label Switched Router |
| MAC | Medium Access Control |
| MPLS | Multiple Protocol Label Switching |
| NAT | Network Address Translation |
| NDDI | Network Development and Deployment Initiative |
| NEC | Nippon Electric Company |
| NSF | National Science Foundation |
| OFELIA | OpenFlow in Europe: Linking Infrastructure and Applications |
| OFP | OpenFlow Protocol |
| ONF | Open Networking Foundation |
| OOBM | Out Of Band Management |
| OSI | Open Systems Interconnection |
| OPEX | Operating EXPenses |
| OSPF | Open Shortest Path First |
| PC | Personal Computer |
| PTR | PoinTeR |
| QoS | Quality of Service |
| REST | REpresentational State Transfer |
| RF | RouteFlow |
| RIP | Routing Information Protocol |
| SAN | Storage Area Network |
| SDK | Software Development Kit |
| SDN | Software Defined Networking |
| SNMP | Simple Network Management Protocol |

| | |
|-------------|-------------------------------|
| SSL | Secure Sockets Layer |
| STP | Spanning Tree Protocol |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |
| TOR | Top Of Rack |
| TR | Transparent Router |
| TTL | Time To Live |
| UDP | User Datagram Protocol |
| URL | Uniform Resource Locator |
| VLAN | Virtual Local Area Network |
| VM | Virtual Machine |
| VPN | Virtual Private Network |
| WAN | Wide Area Network |

1. INTRODUCTION

In a traditional local area network infrastructure, there are plenty of layer 2 and layer 3 devices, i.e., switches and routers, and a set of protocols that determine the optimal path from source to destination by looking into Ethernet and IP headers. With the growth of the network, the traditional method may lead to inefficiencies. In order to meet the growing traffic demands, network expansion takes place and much of the efforts go towards configuring switches and routers even for changes in a smaller segment of a local area network (LAN) that may contain hundreds of nodes. Therefore, smarter, faster and more flexible networks are desired that would control routing of flows in the network, where a flow refers to the unidirectional sequence of packets sharing a set of common packet header values.

Software Defined Networking (SDN) is a new approach in networking technology, designed to create high level abstractions on top of which hardware and software infrastructure can be built to support new cloud computing applications. SDN is also referred to as programmable network, since it isolates control plane from data plane and provides an independent and centralized unit to control the network. OpenFlow protocol follows SDN approach, and gives programmable control of flows to network administrators to define a path that a flow takes from source to destination regardless of the network topology, and utilizes flow based processing for forwarding packets. OpenFlow has gathered significant interest among developers and manufacturers of network switches, routers, and servers.

The main objective of the thesis has been aimed at analyzing the benefits of OpenFlow protocol in traditional LAN environment and its integration and compatibility with some popular protocols in LAN, e.g., virtual LANs (VLANs). It also investigates about the possibility to control the network from a central node by updating flows rather than touching individual devices in the network for a small change in network design.

The thesis starts with a literature review of SDN and OpenFlow protocol. Literature review reveals more about their architecture, working methodology, models of deployment and current deployment. A variety of available simulators, utilities and controllers are reviewed, and HP switches and Floodlight controller are chosen for conducting experiments in laboratory. Experiments helped to reveal more about OpenFlow protocol, which is discussed in Chapter 4. The conclusions of analyses and future work are mentioned in Chapter 5.

2. SOFTWARE DEFINED NETWORKING

2.1. Introduction

In networking devices, there exist three planes: data plane, control plane and management plane. Data plane refers to the hardware part where forwarding takes place, and control plane refers to the software part where all network logics and intelligence takes place. Typically in networking devices, control plane consist of firmware developed and maintained by vendors only [1]. Management plane is typically a part of control plane and is used for network monitoring and controlling purposes. In this thesis, the focus is made on the data and control planes, and they can be seen in Figure 2.1. SDN is an emerging network architecture which separates data and control plane functionalities in a networking device, and makes the control plane independent, centralized and programmable. The migration of control plane enables abstraction of network infrastructure and treats network as a virtual or logical entity.

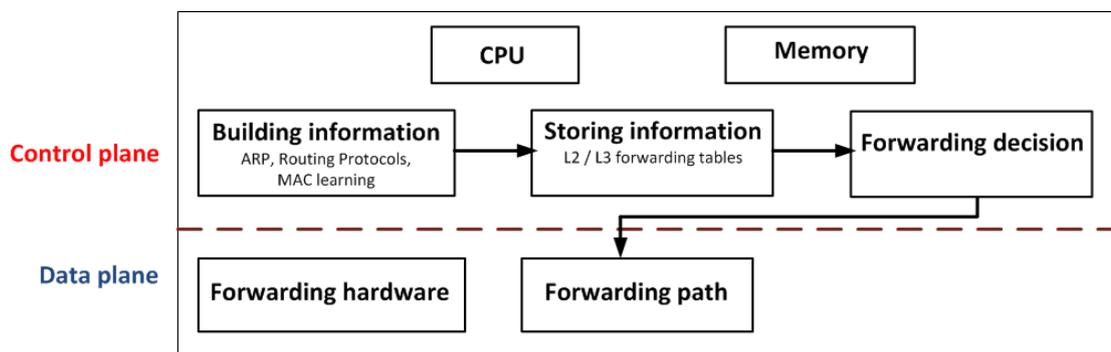


Figure 2.1. Data and control planes in networking hardware

SDN can also be called as a programmable network and seen as a new approach towards business agility by designing and operating innovative networks that are flexible, automated and adaptive to growing business demands of traffic. SDN has been designed to create high level abstractions on top of which hardware or software infrastructure can be built to support new cloud computing applications. SDN addresses a basic issue of maintaining network topology in growing network, and helps making necessary changes in an easy way. SDN allows service providers to expand their network and services with a common approach and tool set, i.e., lower equipment expenditure, while maintaining its control and performance. Apart from service providers and data centres, SDN can also be beneficial to campus and enterprise networks.

2.2. Traditional networking technologies

Traditional networking technologies refer to the LAN and Wide Area Network (WAN), which are composed of various networking devices including switches, routers and firewalls. A traditional LAN interconnects a group of nodes in a relatively small geographic area usually within same building such as university and home. Meanwhile WAN is not bound to any geographic area, but rather it can interconnect across significant areas such as nationwide network in a country and it connects many LANs together [2]. In this thesis, scope of traditional networking technologies is limited to LANs only.

In LAN, data is sent in the form of packets, and various transmission technologies can be utilized for packet transmission and reception. Ethernet is the one that is most widely used, it is specified in the IEEE 802.3 standard, and its recent version Gigabit Ethernet supports a data rate of 1 Gbit/sec and much higher. A packet originates from a source node and reaches its destination node by following a path, which is determined by the networking devices available in the network, i.e., switches and routers.

Switch

A switch operates at layer 2, i.e., data link layer of OSI reference model, and forwards data from one node to other node in a network, which may exist connected to the same switch or to another switch in the network. It registers the Medium Access Control (MAC) addresses of all nodes or devices connected to it into its database known as forwarding table. MAC refers to the hardware address of device, which is a unique address set by manufacturer of device. When a packet arrives at switch, it looks into its forwarding table, and forwards packet to the port of switch where the destination node is connected to [3].

Since, the switch operates at layer 2, any packet which is destined to another IP subnet cannot be processed by it and is sent to a router for further processing. The process of dividing a large network into smaller segments is known as subnetting and the formed network is known as a subnet. It is a common practice to utilize all the available IP addresses for a network so that each device has been assigned a unique IP address.

Router

A router operates at layer 3, i.e., network layer of OSI reference model, and connects multiple networks in a LAN and WAN together and performs computational tasks that include finding and directing the optimal path to the destination from the source, based on the protocol specifications or custom requirements such as the number of hops. It acts as a gateway for forwarding traffic from one IP subnet to other, and uses a routing table to make routing decisions.

A router can also be referred to as store-and-forward packet switch that makes its forwarding decisions based on the destination's IP address in contrast to MAC address used by a switch. When a packet arrives from a switch to a router, the router never forwards a packet to same or another switch, i.e., to a MAC address of a switch. Instead, a router forwards a packet to the destination node, if the destination node lies within the

same IP subnet as the router, or alternatively it forwards a packet to another router if the destination node lies further away [3].

2.3. Limitations of traditional networking technologies

The changing traffic patterns, rise of cloud services, and growing demand of bandwidth has lead service operators to look for innovative solutions, since traditional networking technologies are not able to meet those needs. Factors that limit achieving the growing demand while maintaining profits are enlisted here, which are discussed further [4]:

- Complexity
- Inconsistent policies
- Inability to scale
- Vendor dependence.

Networking protocols have evolved over the time to deliver improved reliability, security, connectivity and performance, and they have different specifications and compatibility levels. Therefore, when changes are planned for a network, all the networking devices must be configured to make changes into effect, resulting into relatively static nature and adding a level of complexity to network. To overcome static nature, server virtualization is being utilized nowadays making networks dynamic in nature. Virtual Machine (VM) migration brings new challenges for traditional networking such as addressing schemes, routing based design etc. Furthermore, All IP network is being operated to support voice, data and video traffic, and maintaining different QoS for different applications for each connection or session increases the complexity of the network. Considering all these issues, a traditional network is not able to dynamically adapt to changing applications and user demands.

Considering different QoS level service provision, a satisfactory QoS policy must be implemented over the network. Due to increasing mobile users, it is not feasible for a service operator to apply a consistent policy to the network, since it may make the network vulnerable to security breaches and other negative consequences.

A network must grow in line with the growing market demands to gain sustainable and competitive markets, users and profits. The network forecast analysis would be helpful, but due to current dynamic market nature it does not provide much help to plan scalability in advance. The complexity and inconsistent policies applied on traditional network limit the faster scalability of a network.

Some of the protocols, services and applications needed in a network environment are vendor dependent, and are not compatible with the equipment from other vendors. When the network is planned to be expanded or new services are to be introduced, the existing infrastructure consists of devices from multiple vendors. The underlying infrastructure needs to be modified, and vendor dependence problem may limit its planned progress and features as well.

2.4. Architecture for SDN

A logical view of SDN architecture can be seen in Figure 2.2. The infrastructure layer refers to the data plane where all hardware lies. The control layer refers to the control plane where SDN control intelligence lies, and the application layer includes all other applications handled by the network. The infrastructure and control layer are connected via control data plane interface such as OpenFlow protocol, whereas the application layer is connected to the control layer via application programming interfaces (APIs).

With the help of SDN, a vendor independent control from a single logical point can be obtained, and a network administrator can shape traffic from a centralized control console rather than going through individual switches. It also simplifies network devices, since devices do not need to understand and process numerous standard protocols but only handle instructions from the controller. In SDN architecture, APIs are used for implementing common network services which are customized to meet business demands such as routing, access control, bandwidth management, energy management, quality of service etc [4].

The limitations of traditional networking technologies make it harder to determine where security devices such as firewalls should be deployed in the network. SDN can overcome the classic problem by implementing a central firewall in the network, and thereby network administrators can route all traffic through a central firewall. This approach facilitates easier and centralized management for security firewall policies, real-time capture and analysis of traffic for intrusion detection and prevention. However, on the other side a central firewall poses a single point of failure in the network [5].

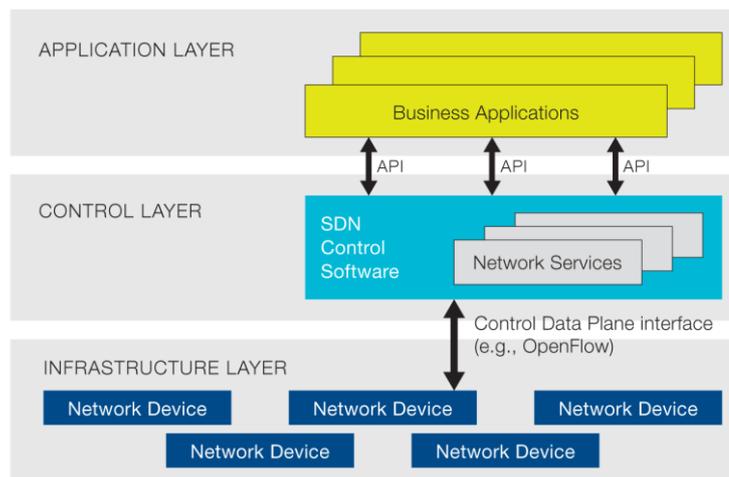


Figure 2.2. Software Defined Networking architecture [4]

The nodes at control layer are called as controllers, and they send information such as routing, switching, priority etc to the data plane nodes associated with them. After receiving the information from control node, the networking devices in the data plane update their forwarding table according to the information received from the control

plane. The control nodes can further be architecturally classified into centralized and distributed mode [6], which can be seen in Figure 2.3 and 2.4 respectively.

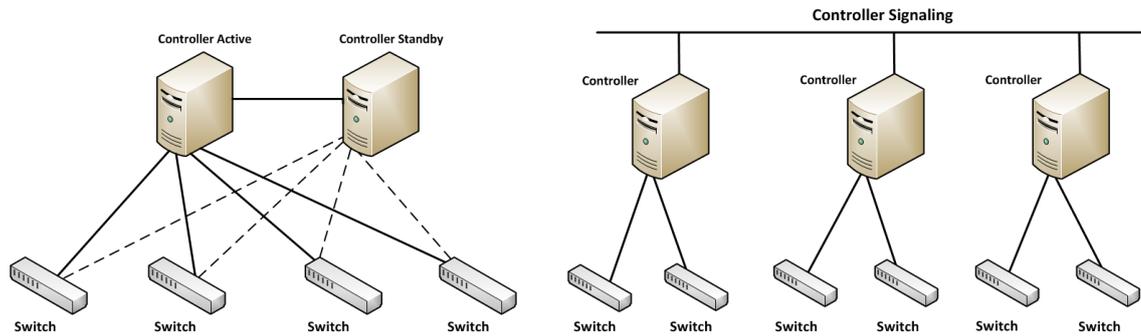


Figure 2.3. Centralized architecture for SDN **Figure 2.4.** Distributed architecture for SDN

In centralized mode, a single central control node sends switching, routing and other information to all networking hardware in the network. Meanwhile, in distributed mode, there are plenty of control nodes associated with certain networking hardware that send information to them [7]. The centralized mode possesses a risk of single point of failure, therefore load-balancing and redundancy mechanisms are often applied in centralized approach deployment.

2.5. Models of deployment for SDN

For the practical deployment of SDN, three different possible models can be approached: switched-based, overlay and hybrid [8]. According to [9], SDN can be classified into embedded and overlaid SDN that resembles to switched-based and overlay models in [8].

Switched-based model refers to replacing entire traditional network with SDN network, and having a centralized control system for each network element. It requires universal support from the network elements; however its limitation includes no leverage over existing layer 2 or layer 3 network equipments.

In overlay model, SDN end nodes are virtual devices that are part of hypervisor environment. This model controls virtual switches at the edge of a network, i.e., computing servers that set up path across the network as needed. It would be useful in cases when SDN network responsibility is handled by server virtualization team, and its limitations include debugging problems, bare metal nodes and overhead for managing the infrastructure.

Hybrid SDN model combines the first two models, and allows a smooth migration towards a switch-based design. The devices that do not support overlay tunnels such as bare metal servers are linked through gateways in this model.

2.6. Data center networking and SDN

A data center is a centralized repository, either physical or virtual, and employs many host servers and networking devices that processes requests and interconnects to another host in the network or to the public network Internet. The requests made to a data center range from web content serving, email, distributed computation to many cloud-based applications. The hierarchical topology of a data center network can be seen in Figure 2.5 below.

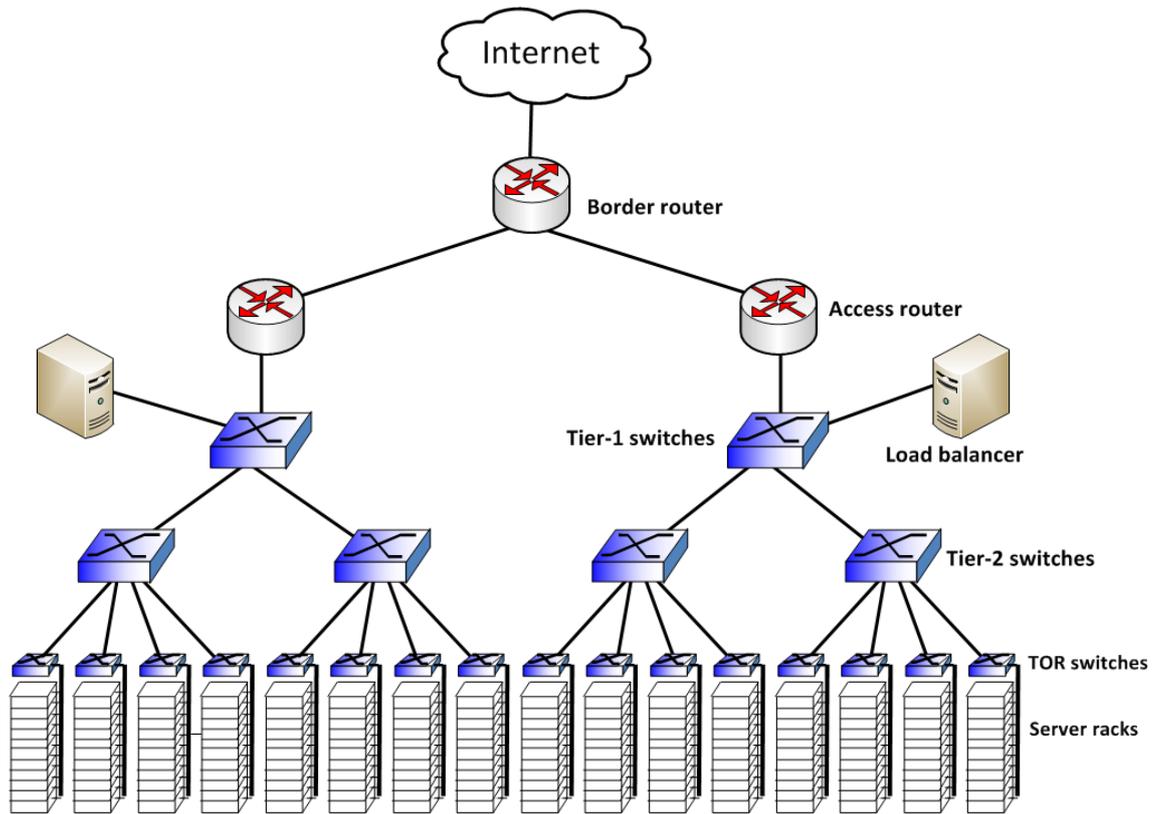


Figure 2.5. Hierarchical topology of a data center network [3]

Host server is also known as blade that has CPU, memory and disk storage. Each server rack resides about 20 to 40 blades inside it, and a switch named Top of Rack (TOR) switch lies on the top of each server rack that interconnects to other hosts and with other switches in the network. Tier-1 switches forward the traffic to and from an access router, and they control tier-2 switches that manage multiple TOR switches in the network. Border routers connect data center network to Internet, they handle external traffic and interconnect external clients and internal hosts to each other.

A data center provides many applications simultaneously that are associated with a publically visible IP address. A load balancer acts as a relay, and performs functions like NAT and firewall preventing direct interactions. So, all external requests are first directed to a load balancer that distributes responses to the associated host server(s), and balances the load across the host servers in the network. With the growing traffic demand, the conventional hierarchical architecture shown in Figure 2.5 can be scaled, but

it limits the host-to-host capacity. Since all the switches are interconnected with Ethernet 1 Gbps or 10 Gbps links, and with the growing traffic the overall throughput for each host is reduced. The solution to this limitation includes upgrading links, switches and routers to support higher rates, but it increases the overall cost of the network.

In order to reduce the overall cost and improve delay and throughput performance, the hierarchy of switches and routers can be replaced by fully connected topology as shown in Figure 2.6 below. In this architecture, each tier-1 switch is connected to all of the tier-2 switches in the network, thereby reducing the processing load as well as improving the host-to-host capacity and the overall performance [3].

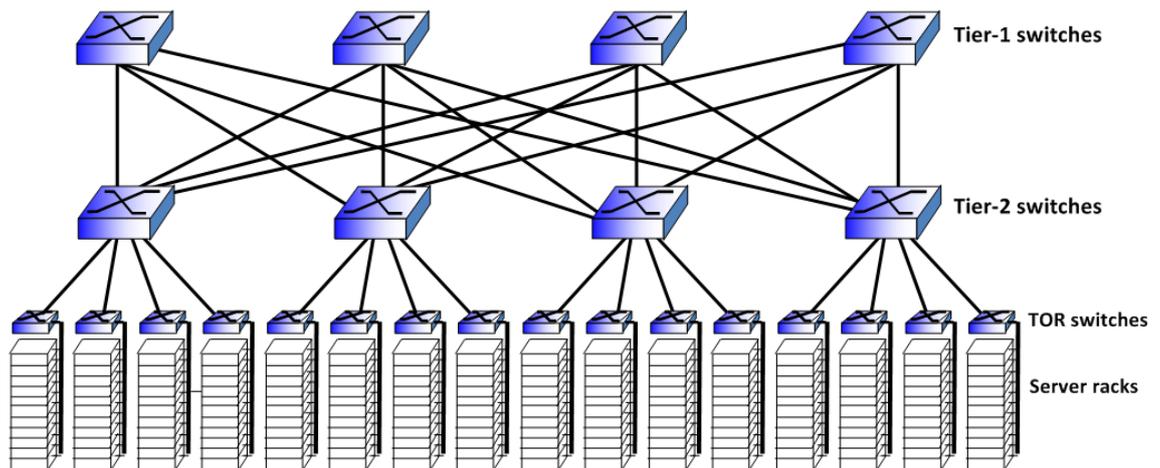


Figure 2.6. Highly interconnected topology of a data center network [3]

In such highly interconnected network design, design of suitable forwarding algorithms for the switches has been a major challenge. SDN approach can be utilized here to make an independent and programmable flow-based forwarding in the network, simplifying network management, and lowering OPEX costs as the network can be managed from a single point.

SDN has attracted many data center operators towards it, and Google has deployed SDN approach into one of its backbone WAN known as an internal (G-scale) network that carries traffic between data centers. The SDN deployed network has been in operation at Google, and has offered benefits including higher resources utilization, faster failure handling and faster upgradation. However, its challenges include fault tolerant controllers, flow programming and slicing of network elements for a distributed control [10]. Similarly NEC has also deployed successfully SDN approach in the data center and backbone network at its own Software Factory, Nippon Express Co., Ltd. and Kanazawa University Hospital in Japan [11][12][13].

2.7. Scalability in SDN

SDN brings numerous advantages including high flexibility, programmable network, vendor independence, innovation, independent control plane, and centralized network.

The centralized control in SDN does not scale well as the network grows, and it leads to concerns about the network performance. It may fail to handle the growing traffic and retain same QoS level as more events and requests are passed to single controller. NOX is one of the earliest OpenFlow controller developed in C++, and a benchmark on NOX controller has revealed that it can handle upto 30,000 flow initiations per second at a delay of 10 msec for each flow installation. The sustainable amount of flows may be sufficient for enterprise and campus area networks, but it does not fit into data center environment. The concerns in centralized approach can be overruled by deploying a distributed SDN. Major factors that have led to these scalability concerns are amount of load on controller, flow overhead and self-healing in failure cases, which are discussed further onwards [14].

2.7.1. Controller scalability and load reduction

In SDN control plane, shifting traditional control functionalities to a remote controller may add more signalling overhead resulting into network performance bottlenecks. The amount of load on a centralized controller can be reduced in various ways that are discussed briefly here. One approach include implementing a controller in parallel cores, this approach has boosted performance of NOX controller by order of magnitude compared with its implementation in a single core. Another approach include classifying the flows and events according to duration and priority, the short duration flows must be handled by data plane while longer duration flows must be sent to a controller and thereby reducing the amount of processing load for a controller.

Meanwhile, in distributed control hierarchy the load can be reduced significantly and this approach has been applied in many applications such as FlowVisor, Hyperflow, and Kandoo. Hyperflow synchronizes the state of a network with all available controllers, giving an illusory control over whole network thereby maintaining an overall topology of a network [15]. Kandoo sorts applications by its scope: local and network-wide; where locally scoped applications are deployed in vicinity of datapath that process requests and messages there thereby reducing controller load. Network-wide applications are handled by controller, and in a distributed hierarchy, a root controller takes care of them and updates about them to all other controllers in a SDN [16]. Flowvisor slices the network, and each slice is handled by a controller or a group of controllers reducing load and making an efficient decision handling mechanism [17].

2.7.2. Flow overhead

In earlier SDN design, controllers were proposed to operate in reactive flow handling manner, where packets for each new flow coming to switch will be sent to controller to decide what to do. Upon arriving at controller, controller looks into its flow table and if flow is found, it sends packet back to switch along with its flow to be installed in switch, otherwise it acknowledges switch to drop that packet. It takes a considerable

time for each flow to be installed in switch database, populates overheads due to flow modification and other messages, and may limit scalability as well.

The amount of delay can be approximated by the resources of a controller, its load, resources of a switch, and their performance. The distance between switch and controller also affects the delay parameter; if they are within proximity of one switch it approximates to 1 msec. Hardware switches are capable of supporting a few thousand flow installations per second with an approx. delay of 10 msec, and reasons for such poor performance include weaker management CPUs, poor high frequency communication support and non-optimal implementations of software that would be resolved in coming years [14].

Considering the reactive flow design, i.e., per flow basis design, it does not scale very well since memory of switch is limited and fixed, and flow overheads and flow setup delay make it less efficient. Therefore proactive manner is well suited, where all flows in a controller are installed instantly to switch database, and if arriving packets do not match any flow there, then they are sent to controller to decide what action should be carried out on them. In this thesis, Floodlight controller is used that follows proactive flow design approach.

2.7.3. Self-healing

In SDN, controller plays a key role and its failure leads to total or partial failure of network in centralized and distributed design respectively. Therefore it is vital to detect its failure via discovery mechanisms and adapt to recover it as soon as possible. Consider a failure situation where failed switch has not affected switch to controller communication as shown in Figure 2.7 below.

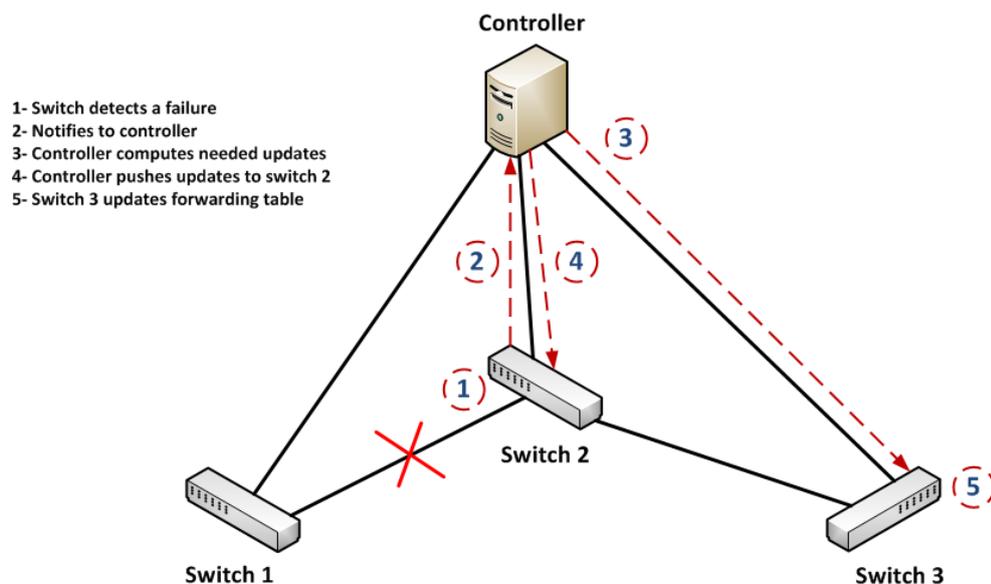


Figure 2.7. Link failure in SDN [14]

Switch 2 detects a failure and notifies controller about it. The controller decides repair actions and installs updates to affected data path element and switches in turn up-

date their forwarding table. Compared with traditional networks, where all link failure notifications are flooded; here in SDN they are directed only to controller. However, considering the worst case scenario where a controller fails then adapting mechanisms should be built in distributed approach using applications like FlowVisor to distribute its load and install flow table to nearby controllers. FlowVisor is described in more detail in Section 3.6.2.

Based on the above discussion and scalability concerns, in data center environment solutions like Kandoo can be implemented to reduce the processing load and make SDN a scalable network. Meanwhile in production network, flows can be aggregated to reduce delay and network slicing applications such as FlowVisor can be useful that help maintain the similar geographic topology from control point of view as well.

2.8. Network management in SDN

As discussed earlier, the network policies implemented via configuration in traditional hardware are low level and networks are static in nature, which are not capable of reacting and adapting to changing network state. In order to configure networking devices easier and faster, network operators usually use external automation tools or scripts that usually lead to some incorrect configurations and a handful troubleshooting at the end. Moreover vendor dependence has limited to proprietary tools and application development, where as network operator's demand for complex high level policies for traffic distribution is expanding rapidly. An event-driven control framework named Procera has been implemented in [18], and its architecture can be seen in Figure 2.8 below.

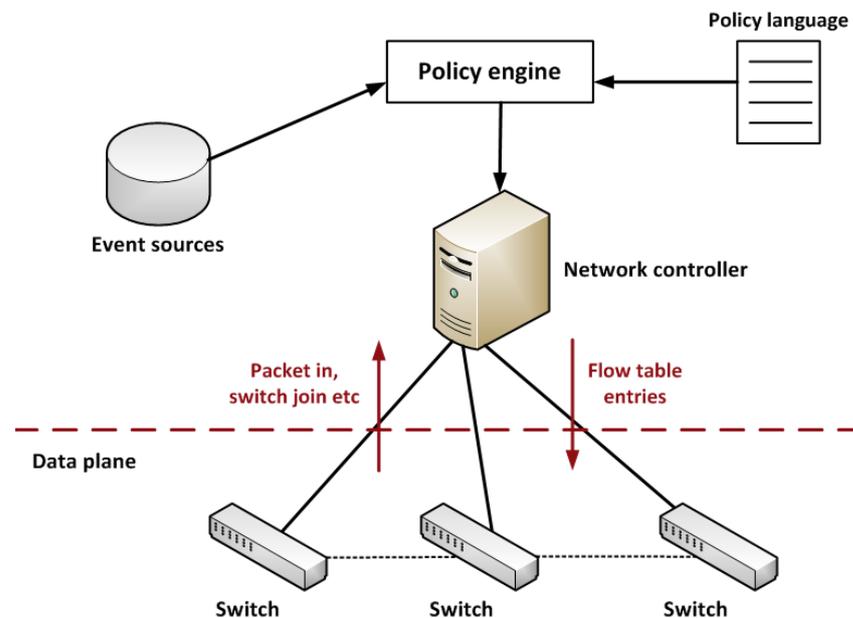


Figure 2.8. Procera architecture [18]

According to [18], three major problems of network management are updating frequent changes to network state, supporting high level policies and provision of better control for network diagnosis and troubleshooting. Procera has been designed using

functional reactive programming (FRP) that translates high level policies into forwarding rules to be installed to underlying switches. Procera supports four control domains namely time, data usage, status and flow which are the most commonly used parameters for implementing traffic policy in a network. In architecture, event sources refer to network components capable of sending dynamic events to controller such as authentication systems, bandwidth monitoring systems, SNMP parameters, and intrusion detection systems etc.

In implemented Procera, event sources were periodically sending files containing information such as bandwidth consumption by every end host device with timestamp. Meanwhile policy engine interprets policies from high level language, i.e., FRP into controller, and processes events arising from event sources. Policy engine is refreshed simultaneously to enforce new policies or make necessary amendments.

3. OPENFLOW

3.1. Introduction

OpenFlow is an open standard that offers controlling the networking equipment programmatically. It was originally designed for network researchers to test their experimental protocols on real networking hardware devices and campus networks that represent everyday networking environment like LAN and WAN. The networking research community has been facing extremely high barriers to experiment new ideas or protocols with the production or traditional networking environment. In order to use the available networking hardware and deployed network to test on experimental protocols and new ideas, OpenFlow emerged out in late 2008, and released its first specifications in December 2008 [19].

Looking at the progress of networking technology in last two decades, it has evolved through large scale and innovative transformations improving its speed, ubiquity, reliability and security. At physical layer, networking devices have improved in terms of computational power and a variety of applications have emerged that offer tools to inspect operations easily. But, the network infrastructure has not been in much change since its early days. In order to add new services, new components are added to support further value added services and operations on the higher layers, while it still remains same at the physical to network layer (layer 1 – 3).

In the existing infrastructure, the networking devices handle network level decisions such as routing or network access. There are a plenty of commercial vendors for networking devices, which run different firmware in their devices, and a network is usually set up in an open fashion rather than proprietary fashion to support devices from different vendors. Open fashion refers to vendor independent deployment, where as proprietary fashion refers to vendor dependent deployment and is deployed often. Depending upon the vendor and its networking device, it has been difficult to test new research ideas such as routing protocols and establish their compatibility in real networks. Furthermore, attempting any experimental ideas over the critical priority production network may result into the failure of the network at some point, which has led to the network infrastructure being static and inflexible, and has not attracted major innovations in this direction [20].

The lookup tables in Ethernet switches and routers have been a major key to implement firewalls, NAT, QoS or to collect performance statistics. OpenFlow takes into account the common set of functions supported by most vendors, which helps to achieve a standard way of utilizing flow tables for all network devices regardless of their vendor. It allows a flow based network partition organizing network traffic into different

flow classes that can be grouped together or isolated to be processed, routed or controlled in a desired manner. OpenFlow can be widely used in campus networks where isolation of research and production traffic is one of the crucial functions.

OpenFlow offers a programmatic control of flows to network administrators to define a path that a flow takes from source to destination, and utilizes flow based processing for forwarding packets. It offers a way to eliminate router's packet processing for defining path, saving power consumptions and network management costs while expanding a network. OpenFlow has gathered significant interest among developers and manufacturers of network switches, routers, and servers [21].

The term 'forwarding' does not refer to layer 2 switching in the OpenFlow protocol environment, since it covers layer 3 information as well, but on the other side it does not perform layer 3 routing. Therefore the term forwarding may be considered to take place between layer 2 switching and layer 3 routing.

3.2. Architecture

In networking devices, there exist three planes: data plane, control plane and management plane as discussed in Section 2.1; however in this thesis only data and control plane are focused. The concept regarding adoption of a centralized control over network, and the separation of control and data plane has been discussed by researchers earlier in [22], and [23]. In SoftRouter, a similar architecture highlighting the decoupling of data and control plane aimed at provisioning of more efficient packet forwarding has been proposed [23]. OpenFlow resembles to these architectures in the concept of separating data and control plane, but validating the concept of flow based processing with help of flow tables.

To gain programmable control over control plane, switches supporting OpenFlow and a controller containing network logic are needed. OpenFlow is based on a switching device with an internal flow table, and provides an open, programmable, virtualized switching platform to control switch hardware via software. It can implement the function of a switch, router or even both, and enables the control path of networking device to be controlled programmatically via OpenFlow protocol as shown in Figure 3.1 [24].

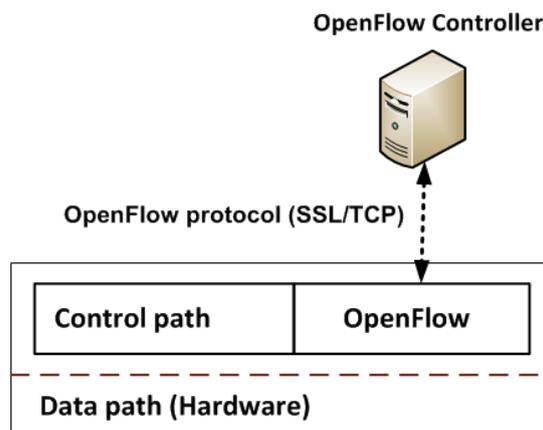


Figure 3.1. OpenFlow physical level architecture [24]

OpenFlow controller connection is secured using either SSL or TLS mostly, but it may be vulnerable to denial of service (DoS) attack; therefore a tight security measure must be implemented to prevent such attacks. In OpenFlow architecture, datapath flow forwarding still resides on the switch, but flow forwarding decisions are made in a separate OpenFlow controller or hierarchy of controllers, which is implemented in a server(s) that communicates with OpenFlow enabled switch(es) in the network through OpenFlow protocol.

Therefore, the main components of an OpenFlow network are:

- Switch(es) with OpenFlow support
- Server(s) running the controller(s)

The components are described further onwards.

3.2.1. OpenFlow enabled switch

A flow table database similar to traditional forwarding table resides on an OpenFlow enabled switch, which contains flow entries helping to perform packet lookup and packet forwarding decisions. An OpenFlow enabled switch is connected to the controller via a secure channel on which OpenFlow messages are exchanged between the switch and the controller to perform configuration and management tasks as shown in Figure 3.2 below.

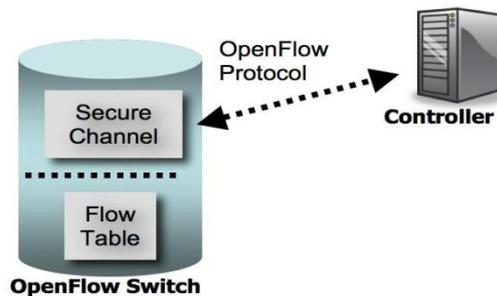


Figure 3.2. Connectivity between OpenFlow switch and controller [25]

An OpenFlow enabled switch contains one or more flow tables and a group table that perform packet lookups and forwarding. Each flow table in the switch contains a set of flow entries, where each flow entry consists of match fields, counters, and a set of instructions or actions to be applied on the matched packets. These fields of a flow entry are described in detail in Section 3.5.

An OpenFlow enabled switch makes forwarding decision by looking into its maintained flow table entries, and finding an exact match on the specific fields of the incoming packets such as port number, source or destination IPv4 address, source or destination MAC address etc. For each flow table entry, there resides an associated action that will be performed on incoming packets. For every incoming packet, the switch goes through its flow table, finds a matching entry and forwards the packets based on the

associated action. In case the incoming packets' flow entries do not match with the flow table of a switch, then, depending upon configuration of OpenFlow network, the switch sends them to the controller to make further decision or continue them to next flow table, as illustrated in Figure 3.3 [21][26].

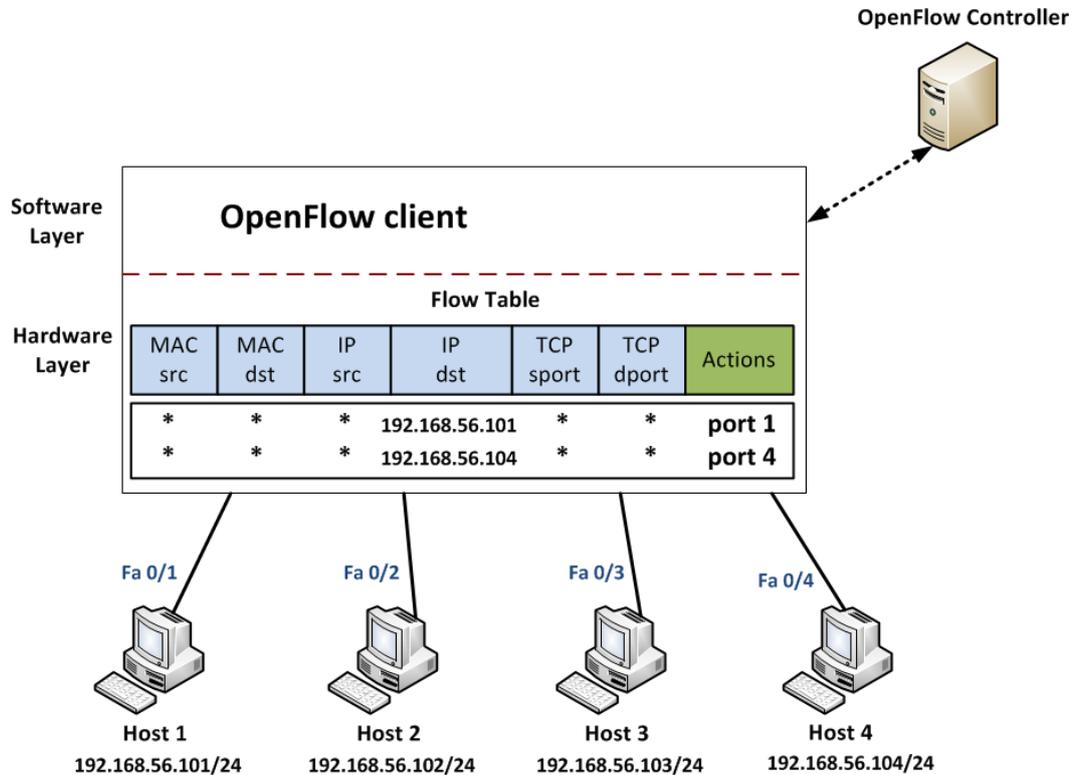


Figure 3.3. OpenFlow packet forwarding process [24]

The associated actions with each flow entry either contain actions or modify pipeline processing. Actions include packet forwarding, packet modification and group table processing whereas in pipeline processing, packets are sent to subsequent flow tables for further processing. The information is communicated from one table to other table in the form of metadata. The pipeline processing stops, when the instruction set associated with a flow entry does not mention a next table, and the packet is modified and forwarded further.

Flow entries may be forwarded to a physical port, logical port or a reserved port. The switch-defined logical port may specify link aggregation groups, tunnels or loop-back interfaces; whereas the specification-defined reserved port may execute generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods, i.e., traditional switch processing. The associated actions with each flow entry may direct packets to a group that does additional processing such as flooding, multipath forwarding, fast reroute and link aggregation [26].

Groups offer a way to forward multiple flow entries to a single identifier, e.g., common next hop. The group table contains group entries and depending upon group type each group entry contains a list of action buckets. Upon arrival of packets at the

group, the actions from associated buckets are executed onto them. The switches from most popular vendors: HP, NEC, Brocade Systems, Juniper Networks and Cisco are compared against features and support for OpenFlow protocol in Appendix A.

3.2.2. Controller

Controller is a centralized entity that gathers control plane functionality – creates updates and removes flow entries in flow tables on a switch, where a flow refers to the unidirectional sequence of packets sharing a set of common packet header values. Along with its primary function, it can further be extended to perform additional critical tasks such as routing and network access.

Currently there are several controller implementations available, which are open-source and are based on different programming languages such as Python, C++, and Java [27]. In this thesis, an open-source Java based controller named Floodlight controller has been chosen for conducting experiments. Typically, a controller runs on a network attached server, and can serve one or multiple switches depending on the network design. It can be designed with centralized hierarchy where one controller handles and controls all the switches in a network or distributed hierarchy where two or more controllers handle and control two or more groups of switches in a network. In centralized hierarchy, if controller fails then all network operations are interrupted, and it poses a single point of failure in an OpenFlow network.

In the distributed hierarchy, all the controllers should have the same copy of the network topology view in real time to avoid the packet losses. The network topology view includes the switch level topology; the locations of users, hosts, middle boxes, and other network elements and services. Moreover it includes all bindings between names and addresses. The most popular OpenFlow controllers: NOX, POX, Floodlight and Trema are compared against some features which can be seen in Appendix B.

3.3. Flow types

Flows can further be divided into microflows and aggregated flows according to number of hosts destined [24].

- **Microflows:** Every flow is individually set up by controller, and follows exact match flow entries criterion. Flow table contains one entry per flow, and is considered good for fine grain control, policy, and monitoring of, e.g., a campus network.
- **Aggregated:** In this case, one flow entry covers large groups of flows, and wildcard flow entries are allowed. Flow table contains one entry per category of flows, and is good for large number of flows, e.g., in a backbone network.

The population process of flow entries into switch can be further classified into reactive and proactive mode [24].

- **Reactive:** First packet of the flow triggers controller to insert flow entries, and switch makes efficient use of flow table where every flow needs small additional

flow setup time. In case of connection loss between switch and controller, switch has limited utility and fault recovery process is simple.

- **Proactive:** In this case, flow tables in switch are pre-populated by controller, and it doesn't account for any additional flow setup time. Typically it requires aggregated (i.e., wildcard) rules and in case of connection loss, traffic is not disrupted.

Aggregate flows reduce the flow overheads, and proactive flows reduce the processing load for a controller since the query is not sent to the controller each time.

3.4. Working methodology

The OpenFlow protocol does not define how the forwarding decisions for specific header fields (i.e., the actions) are made. But in fact, these decisions are made by controller and simply downloaded or installed into switches flow tables. When it comes to OpenFlow switches, the flow tables are looked up and incoming packets' header fields are matched with pre-calculated forwarding decisions where in case of a match the associated decision is followed. If no match is found, then the packet is forwarded to OpenFlow controller for further processing. Depending on the type of flow, i.e., reactive or proactive flow, the controller looks into its database, and upon finding a match it sends packet back to OpenFlow switch and installs the related flow into switch's flow table. The processing of packets via flows in OpenFlow protocol can be seen in a flowchart in Figure 3.4.

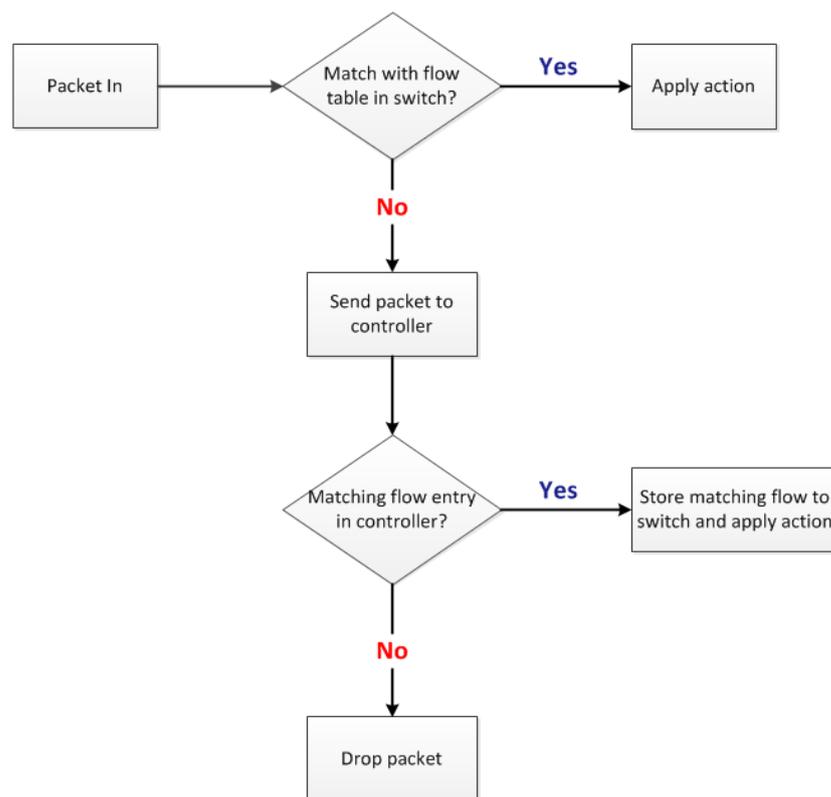


Figure 3.4. Flow processing in OpenFlow protocol [26]

The messages exchanged during connection establishment between a switch and a controller, and working methodology when two hosts connect to each other in an OpenFlow network are described next.

3.4.1. Message types exchanged between switch and controller

OpenFlow provides a protocol for communication between OpenFlow switches and controller, and supports three types of messages that are exchanged between them: controller-to-switch, asynchronous and symmetric messages that are discussed briefly here [26].

The controller-to-switch messages are initiated by the controller and may not always require a response from the switch. These messages are used to configure the switch, manage the switch's flow table and acquire information about the flow table state or the capabilities supported by the switch at any given time, e.g., Features, Config, Modify-State, Read-State, and Packet-Out which are described in connection establishment section next.

The asynchronous messages are sent without solicitation from the switch to the controller and denote a change in the switch or network state, which is also referred to as an event. One of the most significant events includes the packet-in event that occurs whenever a packet that does not have a matching flow entry reaches a switch. Upon occurrence of such an event, a packet-in message is sent to the controller that contains the packet or a fraction of the packet so that it can be examined and a decision about which kind of flow establishment can be made. Some other events include flow entry expiration, port status change or other error events.

Finally, the third category 'symmetric messages' are sent without solicitation in either direction, i.e., switch or controller. These messages are used to assist or diagnose problems in the switch-controller connection, and Hello and Echo messages fall into this category.

3.4.2. Connection establishment between switch and controller

When any switch is configured in OpenFlow mode, the switch starts looking for controller by sending TCP sync message to the controller IP address at the default TCP port 6633. Upon receiving the TCP sync acknowledgement message from the controller, the switch sends acknowledgment again to the controller, and TCP handshake takes place. Therefore, when any new switch will be added to an OpenFlow network, it would be automatically connected to controller. The connection establishment process between switch and controller can be seen in Figure 3.5, where arrows represent the direction of messages.

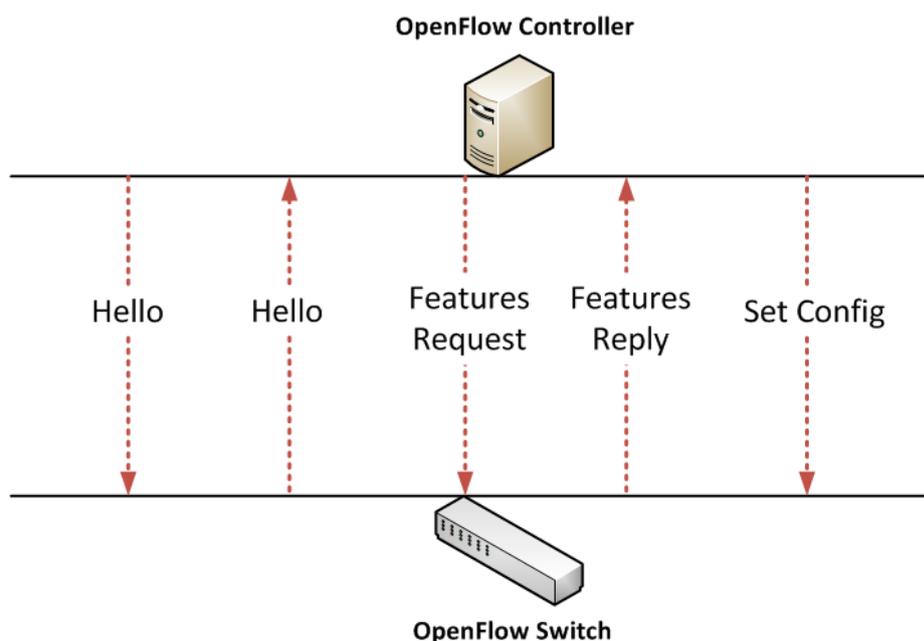


Figure 3.5. Connection establishment between switch and controller

Following the TCP handshake, the process starts from left side and ends at setting configuration to switch. The messages are briefly discussed here [26].

- **Hello** (controller → switch): Controller sends its version number to switch.
- **Hello** (switch → controller): Switch replies with its supported version number.
- **Features Request**: Controller asks to see which ports are available.
- **Features Reply**: Switch replies with a list of ports, port speeds, and supported tables and actions.
- **Set Config**: Controller requests switch to send flow expirations.

Other exchanged messages include Ping Request and Ping Reply.

3.4.3. Connection between hosts on OpenFlow network

After the establishment of connection between switch and controller, the communication process between two or more hosts over an OpenFlow network takes place as shown in Figure 3.6, where arrows represent the direction of messages. The messages are briefly discussed here [26].

- **Packet-In**: When any incoming packet didn't match any flow entry in the switch's flow table, then it is sent to controller.
- **Packet-Out**: Controller sends packets to one or more switch ports.
- **Flow-Mod**: Controller instructs switch to add a particular flow to its flow table.
- **Flow-Expired**: Switch informs controller about flows that have been timed out.
- **Port Status**: Switch notifies controller regarding addition, removal and modification of ports.

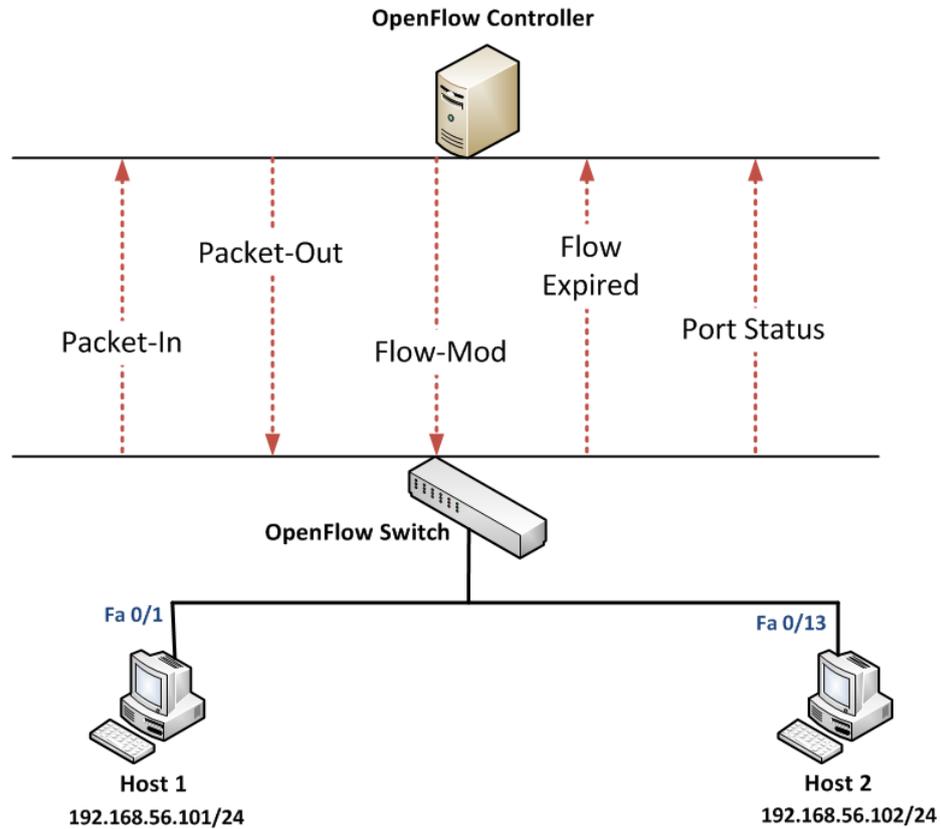


Figure 3.6. Connection between hosts on OpenFlow network

3.5. Packet Format

According to [26], OpenFlow protocol consists of three major fields: header fields, counters and action that reside in every flow entry in a flow table.

- **Header fields or match fields:** These fields identify the flow by matching packets with certain fields which can be seen in Figure 3.7.

| | Ethernet | | | | IP | | | | |
|--------------|----------|-----|------|---------|-----|-----|-------|----------|----------|
| Ingress port | Src | Dst | Type | VLAN ID | Src | Dst | Proto | Src port | Dst port |

Figure 3.7. Header fields to match against flow entries [26]

- **Counters:** They are used for statistics purposes, in order to keep track of the number of packets and bytes for each flow and the time that has elapsed since the flow initiation.
- **Actions:** The action specifies the way in which the packets of a flow will be processed. An action can be one of the following: 1) forward the packet to a given port or ports, after optionally rewriting some header fields, 2) drop the packet 3) forward the packet to the controller.

OpenFlow protocol lies between a switch and a controller enabling the transferring of control plane information. OpenFlow packet specification has been added to a Wireshark dissector [28], and OpenFlow protocol packets can be captured and analyzed in the standard packet capturing tool - Wireshark. OpenFlow protocol has been given acronym ‘OFP’, and the OpenFlow traffic can be filtered in Wireshark by activating ‘of’ filter for traffic. More information about the OpenFlow protocol type of messages and their purpose in Wireshark can be seen in Appendix C.

Consider an OpenFlow network that can be seen in Figure 3.8, where two hosts from a network 192.168.56.0/24 are functional nodes in an OpenFlow network. OpenFlow controller with IP address 192.168.58.110 controls the OpenFlow enabled switch. The OpenFlow enabled switch is connected with the OpenFlow controller via out of band management (OOBM) port with IP address 192.168.58.101.

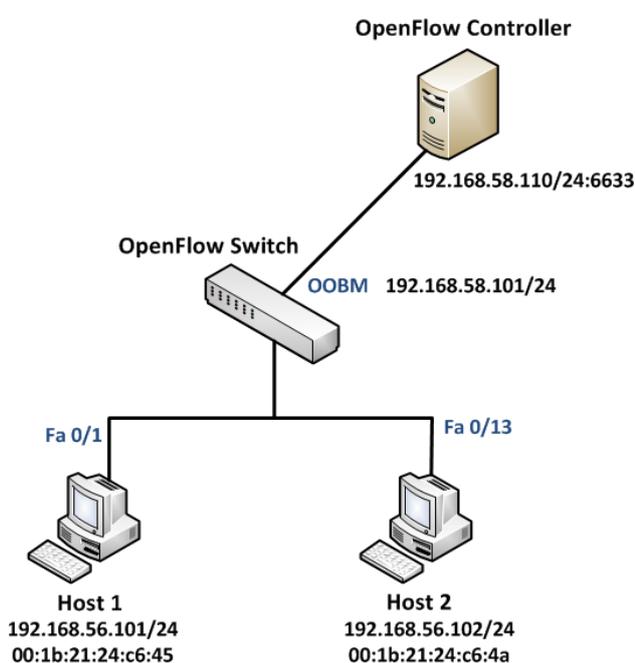


Figure 3.8. Network topology for a sample OpenFlow network

A Wireshark packet capture from this OpenFlow network can be seen in Figure 3.9. It shows header or match fields and actions fields, whereas the counter fields can be seen in the flow table of a switch only and are not visible in packet capture. The counter fields can be seen in Figure 4.5.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|------------|----------------|------------------------|----------|--------|--|
| 966 | 878.554459 | 192.168.58.101 | 192.168.58.110 | OFPP | 74 | Echo Request (SM) (8B) |
| 967 | 878.554901 | 192.168.58.110 | 192.168.58.101 | OFPP | 74 | Echo Reply (SM) (8B) |
| 968 | 878.748143 | 192.168.58.101 | 192.168.58.110 | TCP | 66 | 49714 > 6633 [ACK] Seq=2421 Ack=17107 Win=65872 Len=0 TSval=1384960 TSecr=459336 |
| 969 | 879.890669 | 192.168.58.110 | 192.168.58.101 | OFPP | 146 | Flow Mod (CSM) (88B) |
| 970 | 880.039225 | Cisco dd:85:09 | Spanning-tree-(for-br) | STP | 60 | Conf. Root = 32768/1/00:09:e8:dd:85:00 Cost = 0 Port = 0x0009 |
| 971 | 880.088128 | 192.168.58.101 | 192.168.58.110 | TCP | 66 | 49714 > 6633 [ACK] Seq=2421 Ack=17187 Win=65792 Len=0 TSval=1386300 TSecr=459669 |
| 972 | 882.039197 | Cisco dd:85:09 | Spanning-tree-(for-br) | STP | 60 | Conf. Root = 32768/1/00:09:e8:dd:85:00 Cost = 0 Port = 0x0009 |
| 973 | 882.647538 | Cisco dd:85:09 | Spanning-tree-(for-br) | STP | 60 | Conf. Root = 32768/1/00:09:e8:dd:85:00 Cost = 0 Port = 0x0009 |


```

Frame 969: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on 0
Ethernet II, Src: Hewlett- 63:4a:38 (00:18:fe:63:4a:38), Dst: 10:60:4b:b7:47:81 (10:60:4b:b7:47:81)
Internet Protocol Version 4, Src: 192.168.58.110 (192.168.58.110), Dst: 192.168.58.101 (192.168.58.101)
Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 49714 (49714), Seq: 17107, Ack: 2421, Len: 80
OpenFlow Protocol
  Header
  Flow Modification
    Match
      Match Types
        Input Port: 1
        Ethernet Type: IP (0x0800)
        IP Dst Addr: 192.168.56.102 (192.168.56.102)
        Cookie: 0x00a0000000000000
        Command: New Flow (0)
        Idle Time (sec) Before Discarding: 0
        Max Time (sec) Before Discarding: 0
        Priority: 32570
        Buffer ID: None
        Out Port (delete* only): None (not associated with a physical port)
      Flags
        Output Action(s)
          Action
            Type: Output to switch port (0)
            Len: 8
            Output port: 13
            Max Bytes to Send: 32767
            # of Actions: 1
  
```

— Header or match fields

— Actions

Figure 3.9. An OpenFlow packet capture in Wireshark

The selected packet in the above figure refers to flow modification (i.e., Flow-Mod) message, which is invoked when a new flow is added by the controller. The message originates from the controller (192.168.58.110) and is destined to the switch (192.168.58.101) where the flow will be installed. An example flow from Host1 towards Host2 can be seen in the packet capture, where the flow will be originated from port 1 where Host1 lies and will be destined to port 2 where Host2 lies.

The payload of the Flow-Mod message consists of match fields and associated action that will be inserted into the flow table. From the Wireshark packet capture, it is evident that an OpenFlow packet is just a normal application layer protocol, encapsulated inside TCP, IPv4 and Ethernet format.

3.6. OpenFlow projects

This section describes about some of the OpenFlow projects and frameworks that are relevant to the area of interest.

3.6.1. Open vSwitch

Open vSwitch is an open-source, multi-layer software switch that has been aimed at managing large scale virtualized environments. It is motivated by growing virtualized environment needs, and a superset of OpenFlow protocol is utilized for configuring switch forwarding path. It utilizes centralized controller approach for connecting to OpenFlow enabled switches; however additional management interfaces such as SNMP can be used for configurations.

It functions as a virtual switch, provides connectivity between virtual machines (VMs) and physical interfaces. It also emulates OpenFlow protocol in Linux based virtualized environment including XenServer, kernel based virtual machine (KVM), and VirtualBox. It has implemented OpenFlow protocol v1.0 and onwards, the latest version v1.9.0 includes support for IPv6 as well, which is specified in OpenFlow v1.2 and onwards. In addition to OpenFlow, it supports many other traditional switching techniques including 802.1Q VLAN, QoS configuration, 802.1ag connectivity fault management, and tunnelling techniques such as generic routing encapsulation (GRE) tunnelling [29]. It also provides useful tools and utilities for emulating OpenFlow protocol, which are:

- ovs-vsctl - a utility that queries and updates configuration of soft switch
- ovs-controller - a reference OpenFlow controller
- ovsdbmonitor - a GUI tool for viewing Open vSwitch databases and flow tables.
- ovs-ofctl - a utility that queries and controls OpenFlow switches and controllers.

The architecture of Open vSwitch can be seen in Figure 3.10 below, where ovsdb-server is the database holding the switch level configuration, and ovs-vswitchd is the core component of the Open vSwitch. Ovs-vswitchd supports multiple datapaths, and checks datapath flow counters for flow expiration and stats queries. Ovs-vswitchd is the communication hub that communicates with outside world, ovsdb-server, kernel module and the whole Open vSwitch system. VMs connect to the Open vSwitch kernel through virtual interfaces, and kernel provides connectivity to OpenFlow protocol and the underlying physical interfaces.

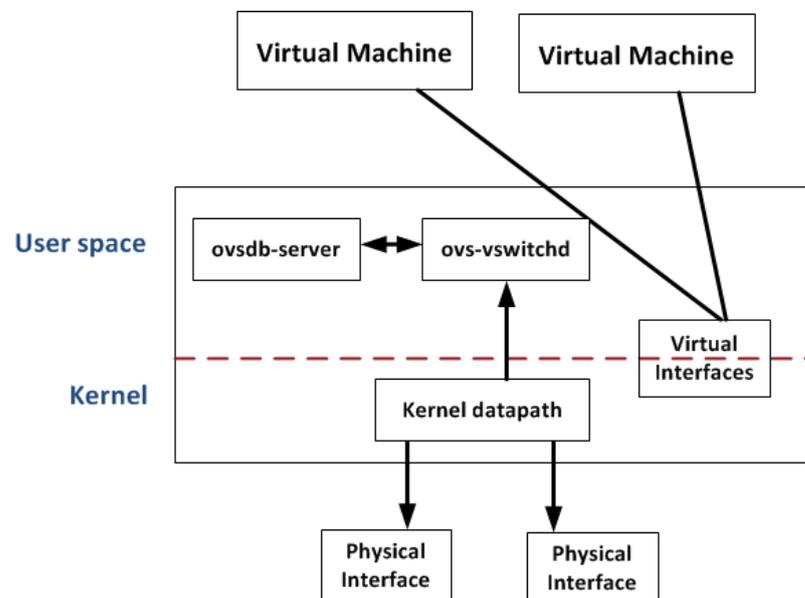


Figure 3.10. Architecture of the Open vSwitch

3.6.2. FlowVisor

The FlowVisor framework is aimed at helping deployment of OpenFlow network with distributed controller approach. It uses flowspaces to create network slices making it easier and independent for multiple OpenFlow controllers to manage flow processing. A slice is defined as a set of packet header bits which match a subset of OpenFlow network traffic, and flowspace refers to a region representing subset of traffic flows that match with packet header bits. Simply, flowspace can be defined as container for a specific region where flows are matched [17]. An OpenFlow network using FlowVisor can be seen in Figure 3.11 below.

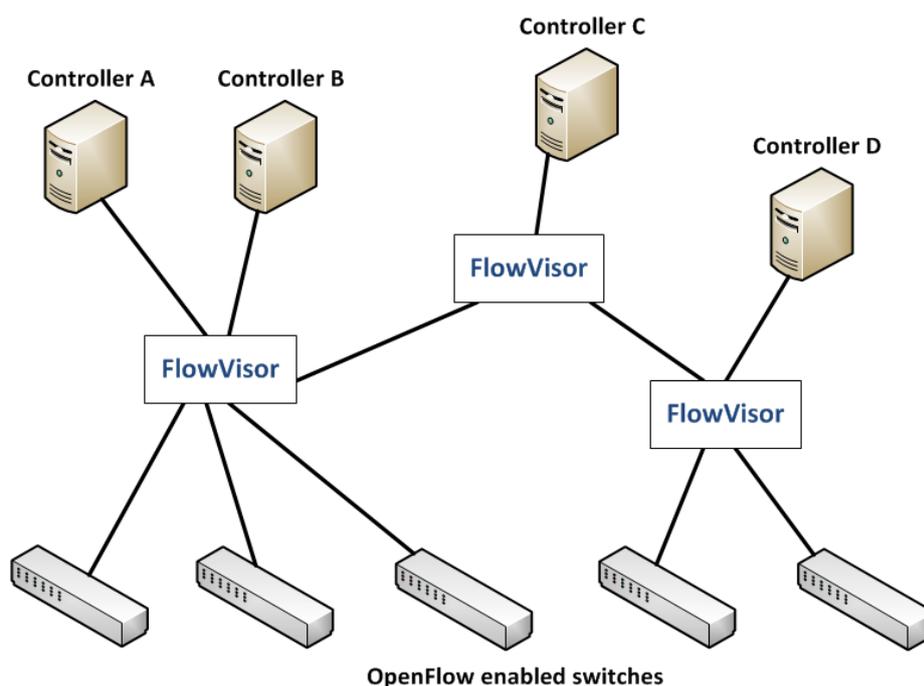


Figure 3.11. OpenFlow network with FlowVisor [30]

Each flowspace can be mapped to one or more OpenFlow controllers in FlowVisor. Furthermore, different levels of access control over a flowspace can be granted to each controller such as write or modify flows and read flows only. It also offers prioritized decision making and useful in cases where overlapping flowspaces occur.

OpenFlow messages arriving from a switch are passed to FlowVisor, where upon inspection they are forwarded to respective controller based on flowspace rules. Therefore each controller only receives packets and messages for which it is responsible, thereby reducing processing load from each controller. On the other side access control mechanism help in achieving it, i.e., packets arriving from a controller are forwarded to an intended switch only if controller has access control granted for that switch or region [30].

3.6.3. LegacyFlow

In order to utilize the benefits of OpenFlow network, all the networking devices (i.e., switches) must be supporting OpenFlow protocol that adds more costs. The proposed architecture aims at retaining traditional networking switches, while utilizing the OpenFlow network. It translates OpenFlow actions into vendor specific configurations for networking switches via SNMP or CLI interfaces, and bridges OpenFlow enabled switches over traditional networking switches by using circuit-based VLANs.

A hybrid model has been proposed in [31] that adds fewer OpenFlow enabled switches to already deployed traditional networking infrastructure. It follows SDN strategy, i.e., separate control and data plane as in OpenFlow protocol, and adds another virtual datapath that interacts with OpenFlow and traditional networks [31][32]. An OpenFlow network utilizing LegacyFlow architecture can be seen in Figure 3.12.

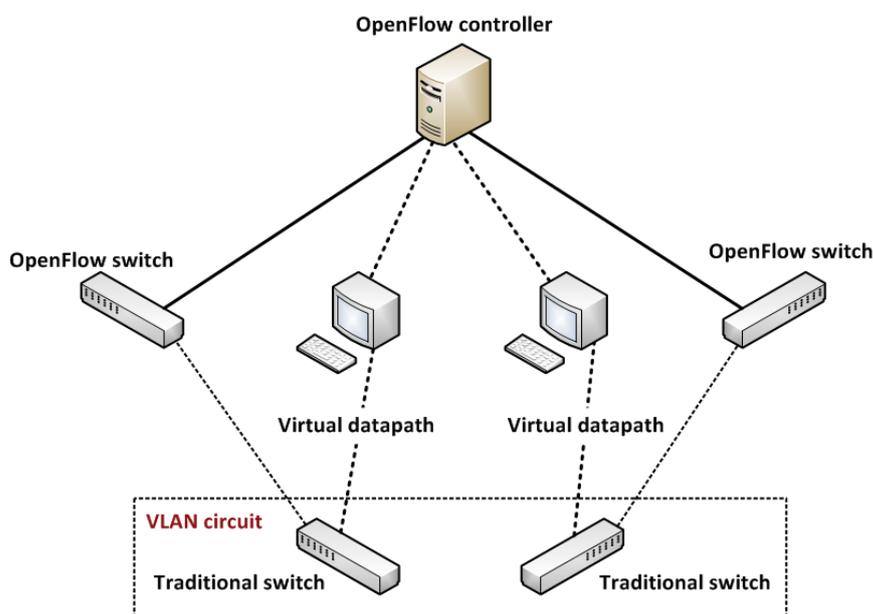


Figure 3.12. OpenFlow network using LegacyFlow approach [31]

Network interfaces are the primary required information to initiate an OpenFlow datapath that is passed as parameters. In accordance with number of ports available or in use in a traditional switch, corresponding number of virtual network interfaces are created by a Linux module mirroring the traditional switch. The Linux module runs in separate Linux machines as shown in the figure. The features of traditional switch are transferred to virtual interfaces via SNMP or a web service.

A virtual datapath is allocated to each traditional switch, which runs in either real or virtual guest Linux OS machine that creates two interfaces: input and output port. It is aimed at conveying an outside view of OpenFlow datapath, and yielding information about features of traditional switches such as sending and receiving packet rate, port numbers etc. OpenFlow messages from OpenFlow controller are received; interpreted and corresponding actions are applied to traditional switch via virtual datapath thereby

serving as a proxy. Actions applied to traditional switch using virtual datapath include creating circuits, acquiring statistics about packets transmitted and received, and removal of circuits. Furthermore, circuits are created with features: shorter duration, with QoS and without timeout.

LegacyFlow initiates virtual datapath and receives information about traditional switch model followed by virtual interface module initiation and creation of virtual interfaces. An outer and dedicated out of band channel, i.e., OOBM port communicates between traditional switch and virtual datapath that receives messages and applies corresponding actions to virtual interfaces. The sequence of messages is updated every 3 seconds to keep track with the changes in switch and network.

After the initiation phase, the corresponding interfaces are connected to the virtual datapath that receive OpenFlow actions from OpenFlow controller. Upon receiving an action, they are interpreted and checked for compatibility with virtual datapath. If it is compatible, a flow is installed into an OpenFlow switch. OpenFlow controller updates its flow table, and determines that destination can be reached via another OpenFlow switch, a circuit is established between these two elements using traditional switches.

3.6.4. RouteFlow

RouteFlow is an open source solution to provide legacy IP routing services such as OSPF, RIP etc over OpenFlow networks and provides a virtual gateway. The architecture of RouteFlow can be seen in Figure 3.13.

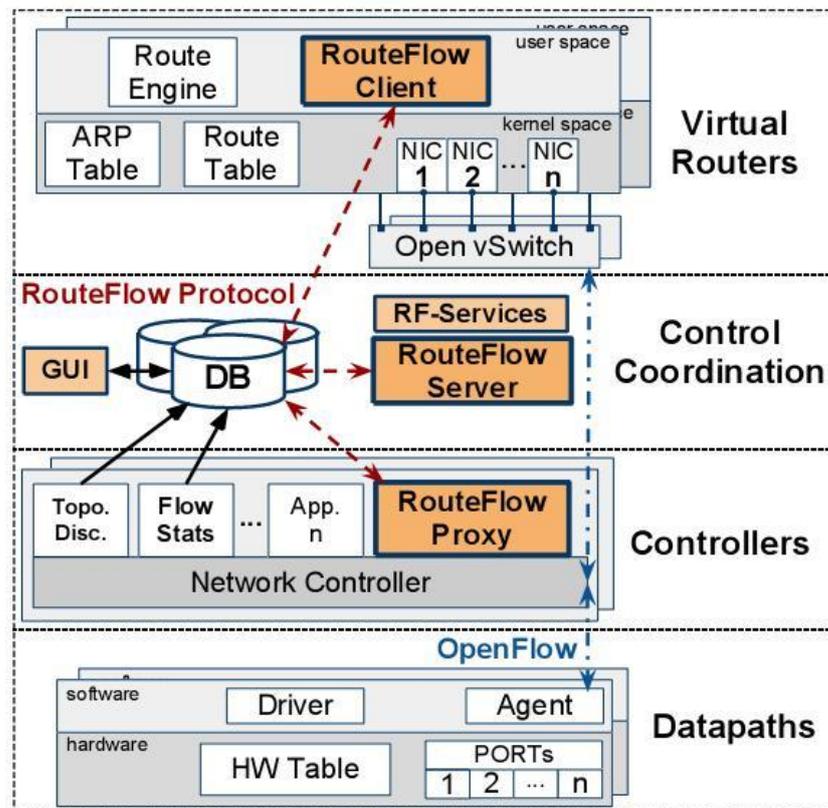


Figure 3.13. RouteFlow architecture [33]

An OpenFlow device is mapped to one virtual machine (VM) with Quagga that provides IP control plane, and RouteFlow monitors routing table changes and installs the corresponding flow entries in the OpenFlow device. Quagga is a GPL licensed routing software suite that provides implementations of OSPFv2, OSPFv3, RIP v1 and v2, and BGP for Unix platforms [34]. All packets that enter OpenFlow devices are sent to corresponding VM and vice versa, while routing protocol messages generated by Quagga are sent out through the RF-server. RouteFlow source code have been designed to work in virtual environment scenario by default, however a flexible mapping can be defined between physical OpenFlow devices and virtual control plane in VM with virtual interfaces mapped to physical ports in OpenFlow enabled device [33].

RF-client is a daemon running in VMs where Quagga is being executed, and is aimed at monitoring changes in Linux routing table. It sends probe packets acting as a location discovery technique that are helpful in mapping virtual interfaces to physical interfaces on an OpenFlow device. Upon detection of changes in Linux routing table, route information is forwarded to RF-server.

RF-server resides the core logic of RouteFlow, and upon receipt of messages about changes in Linux routing table from RF-client it triggers a flow install or flow modification event in OpenFlow device. It also receives registered events from RouteFlow controller module (NOX or POX), and decides what actions are to be taken for those events, e.g., packet-in, datapath-join etc. The last function of RF-server includes registration authority for VMs that maintains synchronization with datapaths.

Looking into architecture shown in [33], from the datapath module where hardware lies, RouteFlow connects to controller module via OpenFlow protocol similar to all other OpenFlow applications. In the controller module, RF-proxy takes care of controller either NOX or POX, and sends control information to control coordination module. In control coordination module, RF-server detects a change in Linux routing table and installs corresponding flows through information from RF-client. In virtual router module, RF-client operates and virtual mapping of ports are carried out.

3.6.5. OpenFlow MPLS

Multiprotocol label switching (MPLS) is a protocol that forwards packets by matching labels in the packet header to destination, and is widely used commercially by network operators. OpenFlow v1.0 specifications do not support MPLS protocol (OpenFlow Switch Specification v1.0.0, 2009), and in [35] an extension of OpenFlow v1.0 has been proposed and implemented to incorporate MPLS support.

The packet headers are modified with three actions namely pushing, popping and swapping MPLS label stack. This modification attaches or removes a label that identifies membership of a forwarding equivalence class (FEC) for packets, and this labelling is inserted in between layer 2 and layer 3, i.e., between IP and MAC. The length of MPLS label stack is 32 bits, out of which 20 bits constitute actual label while the rest indicate time to live (TTL), QoS parameters and indexing.

First of all, two MPLS label are added to the header fields of OpenFlow protocol shown earlier in Figure 3.7 used for identifying flow, which make the header fields 12 bits and can be seen in Figure 3.14 below. One label defines the type of service, e.g., VLAN and other defines transport tunnel as seen in most commercial deployments.

| | Ethernet | | | | IP | | | | | MPLS | |
|--------------|----------|-----|------|---------|-----|-----|-------|----------|----------|---------|---------|
| Ingress port | Src | Dst | Type | VLAN ID | Src | Dst | Proto | Src port | Dst port | Label 1 | Label 2 |

Figure 3.14. OpenFlow modified header fields for MPLS [35]

In order to add MPLS actions namely push, pop, swap, decrement TTL into OpenFlow actions set, virtual port abstraction mechanism was considered that can deal with complex actions requiring modifications to packet header. Thereby, OpenFlow data plane model is modified for virtual port that supports MPLS encapsulation and decapsulation. A virtual port table shows the grouping of virtual and physical ports together, where each table entry consist port number, parent port, actions to be performed and statistics that can be seen in Table 3.1 below.

Table 3.1. Virtual port table entry

| Port No. | Parent port | Actions | Statistics |
|----------|-------------|---------|------------|
|----------|-------------|---------|------------|

Furthermore, OpenFlow protocol was modified to add messages for OpenFlow controller to implement label switched paths (LSPs) into OpenFlow enabled switches. Those messages include vport_mod, vport_table_stats that add or remove a virtual port number, send its actions, and return statistics for them respectively. Moreover, port_stats (i.e., Port Status) message was modified to get the status for virtual ports whether they are available or not. The last field modified include switch_features_reply that informs whether a switch supports virtual ports or not.

After making the needed changes in OpenFlow protocol header and actions, they were implemented on NetFPGA hardware as well as with Open vSwitch. The NetFPGA acted as label switched router (LSR) and the MPLS flows performed well. The successful implementation of this project has lead to addition of MPLS to OpenFlow in OpenFlow v1.1 and onwards.

3.7. Progress and current deployment of OpenFlow

The commercial vendors of networking equipment have started to include the support of OpenFlow protocol to their products and are welcoming software defined networking (SDN) approach. Most of the vendors have joined Open Networking Foundation (ONF) - a body that standardizes and maintains OpenFlow development and has different working groups associated with it. Some of the vendors include HP, IBM, Netgear, NEC, Brocade Systems, and Juniper Networks. In addition to these vendors, Pica8, a

switch manufacturer has been offering software capable switches managed by Xorplus software that support various layer 2 and layer 3 protocols including OpenFlow [36]. Netgear has included OpenFlow support in one of its fully managed next generation switch GSM7328S-200 [37].

The practical deployments of OpenFlow protocol have been at many places worldwide. In USA, Global Environment for Network Innovations (GENI) is a test bed network of resources available for researchers, which are deployed using OpenFlow. GENI is sponsored by National Science Foundation (NSF), and is aimed at collaborating and exploring about innovative discoveries in global networks among academia, industry and public. Some universities in GENI OpenFlow deployment include Stanford University, Indiana University and Princeton University. Majority of the OpenFlow enabled switches used in GENI deployment are from HP, NEC and Pronto [38]. Furthermore, in USA, Network Development and Deployment Initiative (NDDI) programme has been going between Indiana University and Stanford University, which is aimed at delivering SDN solution using OpenFlow. NDDI will be used to create multiple virtual private networks and researchers will gain an opportunity to perform experimental tests on internet protocols and architectures at the same time [39].

In Europe, a European Union's FP7 ICT project OpenFlow in Europe: Linking Infrastructure and Applications (OFELIA) has been focusing upon OpenFlow deployment in Europe, which is aimed at facilitating the researchers to utilize OFELIA networks to perform experiments on research protocols [40].

Google has implemented, and is running its backbone network traffic on an SDN network utilizing OpenFlow protocol. Google decided to build software defined Wide Area Network (WAN) that has resulted into higher performance, more fault tolerant, reduced complexity and cost. Meanwhile the parameters of interest: network performance statistics, cost details and network design details are not mentioned in [41].

An OpenFlow network has also been setup at Kanazawa University Hospital in Japan utilizing NEC programmable flow series devices. The network at hospital had become vast and complex to meet the individual needs of each department and accommodate innovative medical technologies and equipments. It was hard for the hospital staff to utilize, manage and add new devices to the hospital network, and finally SDN approach was utilized. The deployed OpenFlow network at hospital has overcome the challenges, and offered a new stable, flexible, secure and easily manageable network [11].

Furthermore, NEC has implemented an OpenFlow network at its Software Factory in Japan that supports cloud-based software development environment. The deployment has resulted into an internal data center, offered SDN benefits, and an opportunity to switch between virtual servers at East Japan and West Japan premises. The whole infrastructure deployment was achieved within 10 days only where as the estimated deployment time with traditional equipments would have taken about two months [12]. Additionally, NEC has also deployed an OpenFlow network at Nippon Express Co., Ltd., Japan offering them a cloud-based network supporting their worldwide operations [13].

4. EXPERIMENTS

4.1. Laboratory setup

The experiments were conducted at Networks and Protocols laboratory at the department of Communications Engineering. The lab includes a variety of hubs, switches, routers and firewalls from three popular vendors, namely HP, Cisco and Juniper Networks. The lab has four racks installed with these devices and has multiple Linux workstations with CENTOS operating system.

For analyzing OpenFlow protocol, four HP 3800 series switches were used that support OpenFlow protocol v1.0. Two main features which OpenFlow protocol v1.0 specifications don't support are MPLS and IPv6, which are of higher importance for a commercial network deployment. Linux workstations available in the laboratory were used as clients, and one of the Linux workstation acted as controller running Floodlight controller on it. The Linux workstations were updated with the Wireshark dissector plug-in for OpenFlow protocol, and with the needed utilities for the analysis.

4.1.1. HP Switches

The networking switches from most popular vendors are compared against OpenFlow protocol support in Appendix A. HP switches were chosen, since HP has been involved with OpenFlow protocol development since 2007, and has implemented OpenFlow protocol v1.0 in larger number of products compared to other vendors. The other reason for using HP switches was that HP was offering their switches for experimental purposes for a short duration of two months and thereby HP switches were a natural choice for conducting OpenFlow experiments.

HP layer 3 switches HP 3800-24G-2XG (model information J9585A) were set up in the laboratory. The firmware of these switches was upgraded to the latest version KA.15.10.0003, since OpenFlow protocol is supported for firmware KA.15.10 and onwards. The updated firmware supports OpenFlow protocol v1.0 in the beta state, some of the features specified in v1.0 are included and some are not, further information in this regard can be seen in Appendix D. These switches allow standalone OpenFlow network deployment as well as hybrid mode deployment, i.e., OpenFlow and traditional network both can run side by side within the same device.

OpenFlow network can be divided into multiple instances (max. 128) and multiple VLANs (max. 2048) for isolation and identification purposes [42]. An OpenFlow instance is enabled by adding a listen port or a controller to it.

Architecture

Considering the deployment of OpenFlow network, HP switches have been classified architecturally into two modes, namely aggregation mode and virtual mode. Controller and management VLANs are not OpenFlow based, and follow similar structural position in the architecture. In both architectures, more than one controller can be connected for standby, load balancing and independent operations.

- **Aggregation Mode**

In aggregation mode, all the ports in the switch operate in OpenFlow mode and the switch does not support any traditional network traffic which is not OpenFlow tagged. It considers all of the ports and VLANs defined there into a single OpenFlow instance [42]. The aggregation mode architecture can be seen in Figure 4.1 below.

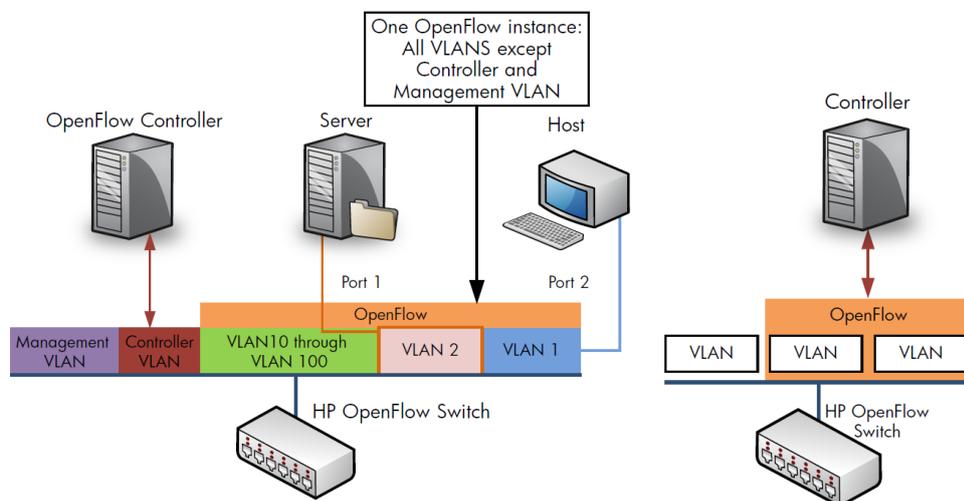


Figure 4.1. Aggregation mode in HP switches [42]

- **Virtualization Mode**

In virtualization mode, hybrid mode operation takes place, production as well as OpenFlow traffic run side by side. Virtualization mode supports multiple instances for OpenFlow network, and the membership of each instance is defined by membership of a VLAN group [42]. In the network configuration, those VLANs which are not members of an OpenFlow instance are not part of OpenFlow network, and they carry production network traffic. Each instance is independent, has its own OpenFlow configuration and a separate controller connection as well, thus enabling more distributed mode of operation. The virtualization mode architecture can be seen in Figure 4.2.

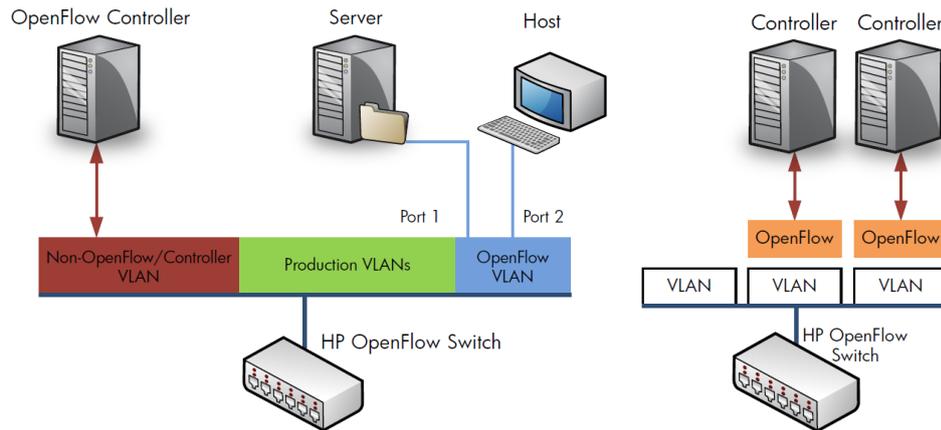


Figure 4.2. Virtualization mode in HP switches [42]

The limitation for virtualization mode include using only one VLAN for an OpenFlow instance, and the same VLAN cannot hold membership for any other instance. Simply,

$$\text{number of OpenFlow instances} = \text{number of VLANs in an OpenFlow network} \quad (4.1)$$

Modes of Operation

This section describes about different modes of operation for OpenFlow instances [42].

- **Active mode:** In active mode, packets that don't match with any flows in a switch's flow table are sent to OpenFlow controller for further actions.
- **Passive mode:** In contrast to active mode, here packets that don't match with any flows are handled by the switch itself rather than sending to OpenFlow controller and waiting for the decision from there.
- **Fail-secure mode:** In this mode, if switch has lost connection with OpenFlow controller, then all OpenFlow packets and messages destined to controller are dropped.
- **Fail-standalone mode:** In this mode, if switch has lost connection with OpenFlow controller, new flows behave similar to legacy switch or router and existing flows of an OpenFlow instance are removed. The switch does not look into OpenFlow flow table anymore and works in traditional way. This mode will be helpful to develop a traditional back up network, and will be activated when controller connection is interrupted and will mitigate network blackout in those cases.

In active mode, switch consults to controller via Packet-In message and controller responds with a decision via Packet-Out message. Usually the decision made is either installing flow in case of reactive flows if a flow match is found, or broadcasting the packets. The broadcasted packets ultimately reach the destination, but the overall throughput is reduced, which will be revealed more in the conducted experiments. However, in HP switches, Packet-In and Packet-Out messages can be stopped by activating the passive mode, but on the other side connectivity is lost. In contrast, with the active mode the network is at least in the operational state with the reduced throughput.

Flow location for OpenFlow

This section describes about hardware and software flows for OpenFlow, and their operational precedence and execution. They are named after the location where flow execution takes place.

- **Hardware flows:** These flows are programmed only in hardware, and those flows which have only action for forwarding to a port are hardware flows.
- **Software flows:** These flows are programmed in software, and those flows which have modify actions such as change source or destination IP address etc are classified as software flows.

A flow must be carefully programmed to enable hardware or software operations. If a flow has multiple actions including hardware actions – forwarding with it, and if any of actions has been supported in software then flow execution will take place in software only. A flow execution in hardware will only take place when flow matching parameters and actions are supported in hardware and it does not have any additional software actions there [42][42].

4.1.2. Linux tools and utilities

This section describes about different Linux tools and utilities that were utilized for conducting experiments at laboratory.

- **Curl:** It is a command line tool for transferring data with URL syntax that supports SSL certificates, HTTP and other protocols [43]. It is used to insert flows into the static flow entry API for Floodlight controller, and an example configuration can be seen in Section 4.2.
- **Avior:** It is a Java based application offering GUI for flow administration, logical patch panelling and real time statistics for controller, switch, device and ports [44]. Similar to curl, it is also used to insert flows into the static flow entry API for Floodlight controller.
- **Iperf:** It is a command line utility for measuring maximum TCP and UDP bandwidth performance, and reports delay jitter and datagram loss as well. It also allows tuning different parameters [45].
- **Netperf:** It is a benchmarking tool used to measure different networking performance aspects including unidirectional data transfer and request or response performance using either TCP or UDP [46].

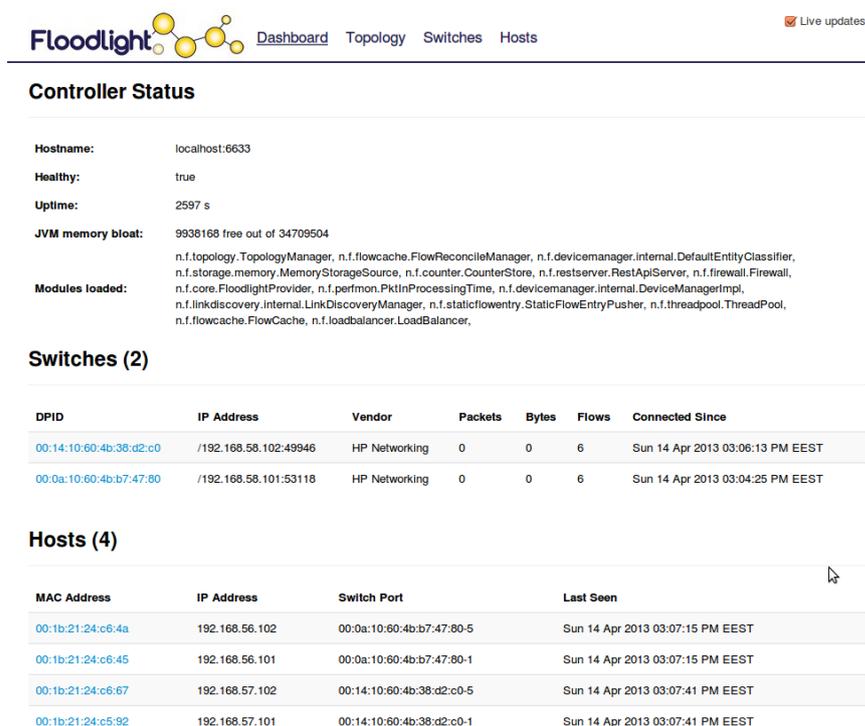
4.1.3. Floodlight Controller

Floodlight is an open source, enterprise class, Java based OpenFlow controller with an Apache license, which is developed and maintained currently by Big Switch Networks [47]. Flows are added to the controller using a REpresentational State Transfer (REST) Application Programming Interface (API) named Static Flow Pusher, which is a JavaScript Object Notation (JSON) interface to configure proactive flows into Floodlight controller [48]. Flows are inserted, deleted and overwritten to Floodlight controller using

curl command to reach Static Flow Entry Pusher object. Another popular utility to add flows to Floodlight controller is Avior, which is a Java based graphical user interface (GUI) application offering flow administration and real time statistics being updated frequently [44].

Floodlight offers adding proactive as well as reactive flows to OpenFlow enabled switch. In proactive flow, a flow entry is sent and stored into flow table of OpenFlow enabled switch before any traffic arrives. In contrast to proactive flows, in reactive flow, a flow entry is not instantly sent to flow table of a switch, but it stays in the OpenFlow controller. The flow will only be inserted into flow table of a switch, when a packet arrives at switch; it doesn't match with any flow and is sent to OpenFlow controller for further decision. OpenFlow controller checks its flow entries, makes the decision, sends packet back to switch and inserts the flow entry into flow table of switch.

Each OpenFlow switch attached to an OpenFlow controller is identified by its unique datapath identifier (DPID). DPID is 8 bytes long and is specified either as a decimal number or as 8 hexadecimal (HEX) octets, e.g., 00:00:00:23:10:35:ce:a5. The DPID ff:ff:ff:ff:ff:ff:ff is a "wildcard" DPID that matches all DPIDs inside an OpenFlow network. Floodlight controller offers a web interface shown in Figure 4.3 below that delivers information about controller (loaded modules & APIs), switches (DPID, ports & flows) and attached devices or hosts in an OpenFlow network.



The screenshot displays the Floodlight web interface. At the top, there is a navigation bar with the Floodlight logo and links for Dashboard, Topology, Switches, and Hosts. A 'Live updates' indicator is visible on the right. The main content area is divided into three sections:

- Controller Status:** Shows the controller's hostname as localhost:6633, its health as true, uptime as 2597 s, and JVM memory usage as 9938168 free out of 34709504. It also lists several loaded modules.
- Switches (2):** A table listing two switches with their DPIDs, IP addresses, vendors, and statistics.
- Hosts (4):** A table listing four hosts with their MAC addresses, IP addresses, switch ports, and last seen times.

| DPID | IP Address | Vendor | Packets | Bytes | Flows | Connected Since |
|-------------------------|-----------------------|---------------|---------|-------|-------|----------------------------------|
| 00:14:10:60:4b:38:d2:c0 | /192.168.58.102:49946 | HP Networking | 0 | 0 | 6 | Sun 14 Apr 2013 03:06:13 PM EEST |
| 00:0a:10:60:4b:b7:47:80 | /192.168.58.101:53118 | HP Networking | 0 | 0 | 6 | Sun 14 Apr 2013 03:04:25 PM EEST |

| MAC Address | IP Address | Switch Port | Last Seen |
|-------------------|----------------|---------------------------|----------------------------------|
| 00:1b:21:24:c6:4a | 192.168.56.102 | 00:0a:10:60:4b:b7:47:80-5 | Sun 14 Apr 2013 03:07:15 PM EEST |
| 00:1b:21:24:c6:45 | 192.168.56.101 | 00:0a:10:60:4b:b7:47:80-1 | Sun 14 Apr 2013 03:07:15 PM EEST |
| 00:1b:21:24:c6:67 | 192.168.57.102 | 00:14:10:60:4b:38:d2:c0-5 | Sun 14 Apr 2013 03:07:41 PM EEST |
| 00:1b:21:24:c5:92 | 192.168.57.101 | 00:14:10:60:4b:38:d2:c0-1 | Sun 14 Apr 2013 03:07:41 PM EEST |

Figure 4.3. Floodlight web interface

Based on the data available about devices and hosts connected to it, it draws a visual diagram showing which host is connected to which switch, and how switches are con-

nected to controller. The web interface is useful for analysis purposes only; therefore flow configuration or additional configuration cannot be invoked from the web interface.

The most popular OpenFlow controllers: NOX, POX, Floodlight and Trema are compared against some features in Appendix B. Floodlight controller provides proactive flows thereby reducing the load on controller and flow setup time. It responds to maximum number of queries per second made to it, but on the other side it has more latency than Trema controller. Flows can be installed easily using static flow entry pusher API or third partner utilities Curl and Avior discussed above. It is Java based, therefore can be deployed and controlled from multiple platforms including Android. It has been released in early 2012, but it has evolved from Beacon controller - the earliest OpenFlow controller developed at Stanford University in early 2010 [49]. Considering the proactive flows, its development history, an active mailing list and easier flow installation method, Floodlight controller was chosen.

4.2. Experiment 1: Basic setup inside the same subnet

This experiment has been conducted to verify the basic operation of OpenFlow protocol. The topology of the network can be seen in Figure 4.4, where two hosts from a network 192.168.56.0/24 are functional nodes in an OpenFlow network. The OpenFlow switch is configured in the aggregation mode.

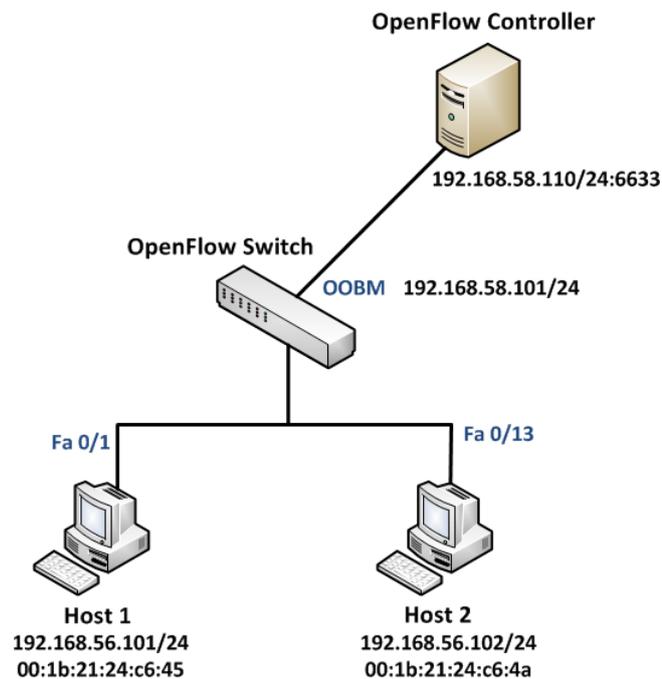


Figure 4.4. Network topology for Experiment 1

4.2.1. Flow entries

For this experiment, OpenFlow controller Floodlight was configured to create and install proactive flows to OpenFlow switch for hosts in network. A total of two flows were configured: one from Host1 to Host2 and second vice-versa, which can be seen in Table 4.1 below.

Table 4.1. Flows for Experiment 1

| Flow # | Matching criterion | Actions |
|--------|-----------------------------------|-------------|
| 1 | in_port 1, dst_ip 192.168.56.102 | out_port 13 |
| 2 | in_port 13, dst_ip 192.168.56.101 | out_port 1 |

In the above table, in_port refers to input port, dst_ip refers to destination IP address and out_port refers to output port. The flows are installed into HP switch, but initially configured to OpenFlow controller using Curl utility, and an example configuration can be seen here. While configuring the flow, the switch is identified by its unique eight HEX octet ID known as DPID here.

```
curl -d '{"switch":"10:00:10:60:4b:b7:47:80", "name":"flow1", "active":"true",
"priority":"32570","actions":"output=13","ether-type":"0x0800","ingress-port":
"1","dst-ip":"192.168.56.102"}'
http://192.168.58.110:8080/wm/staticflowentrypusher/json
```

The Floodlight controller supports proactive flows, and installs flows instantly into HP switch. One of the flows named ‘flow1’ appears in the flow table of HP switch as seen in Figure 4.5.

```
switch2(config)# show openflow instance aggregate flows
```

| OpenFlow Flow Table | |
|----------------------------|---------------------------------|
| Flow 1 | |
| Incoming Port : 1 | Ethernet Type : IP |
| Source MAC : 000000-000000 | Destination MAC : 000000-000000 |
| VLAN ID : 0 | VLAN priority : |
| Source IP : 0.0.0.0 | Destination IP : 192.168.56.102 |
| IP Protocol : 0x00 | IP ToS Bits : 0 |
| Source Port : 0 | Destination Port : 0 |
| Priority : 32570 | |
| Duration : 142 seconds | |
| Idle Timeout : 0 seconds | Hard Timeout : 0 seconds |
| Packet Count : 407920 | Byte Count : 0 |
| Flow Location : Hardware | |
| Actions | |
| Output | : 13 |

Figure 4.5. A flow in flowtable in HP switch

4.2.2. Results

After the flows were configured, ICMP ping messages were sent across the network to verify connectivity, and the tests revealed a functional OpenFlow network. The Wireshark capture on OpenFlow controller machine helped further analyzing the OpenFlow protocol steps: switch to controller connection, flows installed to switch (switch modification) and OpenFlow traffic monitoring.

Since the hosts are inside same subnet, they can reach other without any flows configured in OpenFlow network as well. However, in such cases the throughput between the hosts or total throughput of network will be reduced. When there are neither any flows available in switch nor any matching flows for a connection in switch, then switch contacts controller to determine what actions should be taken with the traffic using Packet-In message. Considering the ICMP ping request, Packet-In message goes through controller and controller sends Packet-Out message with the packet itself and its associated decision (in this case packet flooding), and finally the packet reaches the destination node via flooding.

Packet-In and Packet-Out messages are encapsulated inside an OpenFlow protocol packet header, and are captured as OFP+ICMP protocol in Wireshark. These messages and the process as a whole can further be seen in Figure 4.6 and Figure 4.7 respectively, and usually all encapsulated messages appear as OFP+ messages in Wireshark. In such cases, Wireshark dissector specifies the actual source and destination IP address, but in fact the packet goes from switch to controller for consultation. Considering Figure 4.6, an ICMP ping request from Host1 to Host2 is made; it goes from the OOBM port of an OpenFlow enabled switch (192.168.58.101) to the controller and can be figured out from IPv4 fields of a packet. It encapsulates the original ICMP packet from Host1 to Host2 inside the normal OpenFlow packet and sends a Packet-In message that originates from the OpenFlow enabled switch.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|------------|-------------------|---------------------------|-----------|--------|--|
| 204 | 252.008991 | Cisco dd:85:09 | Spanning-tree-(for-br:STP | STP | 60 | Conf. Root = 32768/1/00:09:e8:dd:85:00 Cost = 0 Port = 0x8009 |
| 205 | 252.553351 | Cisco dd:85:09 | Cisco dd:85:09 | LOOP | 60 | Reply |
| 206 | 253.462341 | 192.168.56.101 | 192.168.56.102 | OFP+ICMP | 186 | Packet In (AM) (120B) => Echo (ping) request id=0xfff05, seq=1/256, ttl=64 |
| 207 | 253.480144 | 192.168.56.101 | 192.168.56.102 | OFP+ICMP | 192 | Packet Out (CSM) (126B) => Echo (ping) request id=0xfff05, seq=1/256, ttl=64 |
| 208 | 253.480963 | 192.168.56.102 | 192.168.56.101 | OFP+ICMP | 186 | Packet In (AM) (120B) => Echo (ping) reply id=0xfff05, seq=1/256, ttl=64 |
| 209 | 253.486200 | 192.168.56.102 | 192.168.56.101 | OFP+ICMP | 272 | Packet Out (CSM) (126B) => Echo (ping) reply id=0xfff05, seq=1/256, ttl=64 |
| 210 | 253.686049 | 192.168.58.101 | 192.168.58.110 | TCP | 66 | 49714 > 6633 [ACK] Seq=1533 Ack=837 Win=65772 Len=0 TSval=759900 TSecr=303068 |
| 211 | 254.003947 | 10:60:4b:b7:47:b3 | LLDP_Multicast | OFP+LLDP | 151 | Packet Out (CSM) (85B) => Chassis Id = 10:60:4b:b7:47:80 Port Id = TTL = 120 |
| 212 | 254.004010 | 10:60:4b:b7:47:bf | LLDP_Multicast | OFP+LLDP | 151 | Packet Out (CSM) (85B) => Chassis Id = 10:60:4b:b7:47:80 Port Id = TTL = 120 |
| 213 | 254.008904 | Cisco dd:85:09 | Spanning-tree-(for-br:STP | STP | 60 | Conf. Root = 32768/1/00:09:e8:dd:85:00 Cost = 0 Port = 0x8009 |
| 214 | 254.048744 | 10:60:4b:b7:47:b3 | Broadcast | OFP+0x89: | 159 | Packet Out (CSM) (93B) => Ethernet II |
| 215 | 254.049273 | 10:60:4b:b7:47:bf | Broadcast | OFP+0x89: | 159 | Packet Out (CSM) (93B) => Ethernet II |
| 216 | 254.246038 | 192.168.58.101 | 192.168.58.110 | TCP | 66 | 49714 > 6633 [ACK] Seq=1533 Ack=1193 Win=65416 Len=0 TSval=760460 TSecr=303198 |

▶ Frame 206: 186 bytes on wire (1488 bits), 186 bytes captured (1488 bits)
 ▶ Ethernet II, Src: 10:60:4b:b7:47:81 (10:60:4b:b7:47:81), Dst: Hewlett- 63:4a:38 (00:18:fe:63:4a:38)
 ▶ Internet Protocol Version 4, Src: 192.168.58.101 (192.168.58.101), Dst: 192.168.58.110 (192.168.58.110)
 ▶ Transmission Control Protocol, Src Port: 49714 (49714), Dst Port: 6633 (6633), Seq: 1293, Ack: 505, Len: 120
 ▼ OpenFlow Protocol
 ▶ Header
 ▼ Packet In
 Buffer ID: 4294967295
 Frame Total Length: 102
 Frame Recv Port: 1
 Reason Sent: No matching flow (0)
 Frame Data: 001b2124c64a001b2124c645810040010800450000540000...
 ▶ Ethernet II, Src: IntelCor_24:c6:45 (00:1b:21:24:c6:45), Dst: IntelCor_24:c6:4a (00:1b:21:24:c6:4a)
 ▶ 802.1Q Virtual LAN, PRI: 2, CFI: 0, ID: 1
 ▶ Internet Protocol Version 4, Src: 192.168.56.101 (192.168.56.101), Dst: 192.168.56.102 (192.168.56.102)
 ▶ Internet Control Message Protocol

Figure 4.6. Packet-In message when flows are not defined

| No. | Time | Source | Destination | Protocol | Length | Info |
|--|------------|-------------------|-----------------------|----------|--------|---|
| 204 | 252.008901 | Cisco dd:85:09 | Spanning-tree-(for-br | STP | 60 | Conf. Root = 32768/1/00:09:e8:dd:85:00 Cost = 0 Port = 0x8009 |
| 205 | 252.553351 | Cisco dd:85:09 | Cisco dd:85:09 | LOOP | 60 | Reply |
| 206 | 253.462341 | 192.168.56.101 | 192.168.56.102 | 0FP+ICMP | 186 | Packet In (AM) (120B) => Echo (ping) request id=0xff05, seq=1/256, ttl=64 |
| 207 | 253.480144 | 192.168.56.101 | 192.168.56.102 | 0FP+ICMP | 192 | Packet Out (CSM) (126B) => Echo (ping) request id=0xff05, seq=1/256, ttl=64 |
| 208 | 253.480963 | 192.168.56.102 | 192.168.56.101 | 0FP+ICMP | 186 | Packet In (AM) (120B) => Echo (ping) reply id=0xff05, seq=1/256, ttl=64 |
| 209 | 253.486208 | 192.168.56.102 | 192.168.56.101 | 0FP+ICMP | 272 | Packet Out (CSM) (126B) => Echo (ping) reply id=0xff05, seq=1/256, ttl=64 |
| 210 | 253.686049 | 192.168.58.101 | 192.168.58.110 | TCP | 66 | 49714 > 6633 [ACK] Seq=1533 Ack=837 Win=65772 Len=0 TSval=759900 TSecr=303068 |
| 211 | 254.003947 | 10:60:4b:b7:47:b3 | LLDP_Multicast | 0FP+LLDP | 151 | Packet Out (CSM) (85B) => Chassis Id = 10:60:4b:b7:47:80 Port Id = TTL = 120 |
| ▶ Frame 207: 192 bytes on wire (1536 bits), 192 bytes captured (1536 bits) | | | | | | |
| ▶ Ethernet II, Src: Hewlett- 63:4a:38 (00:18:fe:63:4a:38), Dst: 10:60:4b:b7:47:81 (10:60:4b:b7:47:81) | | | | | | |
| ▶ Internet Protocol Version 4, Src: 192.168.58.110 (192.168.58.110), Dst: 192.168.58.101 (192.168.58.101) | | | | | | |
| ▶ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 49714 (49714), Seq: 505, Ack: 1413, Len: 126 | | | | | | |
| ▼ OpenFlow Protocol | | | | | | |
| ▶ Header | | | | | | |
| ▼ Packet Out | | | | | | |
| Buffer ID: None | | | | | | |
| Frame Recv Port: 1 | | | | | | |
| Size of action array in bytes: 8 | | | | | | |
| ▶ Output Action(s) | | | | | | |
| ▼ Frame Data: 001b2124c64a001b2124c645810040010800450000540000... | | | | | | |
| ▶ Ethernet II, Src: IntelCor_24:c6:45 (00:1b:21:24:c6:45), Dst: IntelCor_24:c6:4a (00:1b:21:24:c6:4a) | | | | | | |
| ▶ 802.1Q Virtual LAN, PRI: 2, CFI: 0, ID: 1 | | | | | | |
| ▶ Internet Protocol Version 4, Src: 192.168.56.101 (192.168.56.101), Dst: 192.168.56.102 (192.168.56.102) | | | | | | |
| ▶ Internet Control Message Protocol | | | | | | |

Figure 4.7. Packet-Out message when flows are not defined

From the figures, it is clear that in the absence of flows, the controller performs flooding that reduces the overall throughput of OpenFlow network. Further, it is evident that even though the flows are defined, and network connectivity retains in OpenFlow network, the flow performance should be determined by looking at flow table counters and throughput. Flow table counters such as packets count, bytes count can be determined by looking into flow table of HP switch as shown in Figure 4.5. The ICMP ping request may yield connectivity via broadcasted packets by controller, but flow counters and throughput results will reveal the reality. Both these mechanisms of flow counters and throughput analysis have been applied in all the experiments conducted in laboratory to obtain optimal results.

Throughput results were computed using iperf and netperf utilities for TCP and UDP streams, however TCP and UDP had similar throughput with both utilities. The result can be seen in Table 4.2 below that reveal a drastic drop in network throughput if there are not any flows or they are not configured properly.

Table 4.2. Throughput for Experiment 1

| Situation | Network throughput |
|-----------------------------------|--------------------|
| Before deploying OpenFlow network | 944 Mbits/sec |
| No flows in OpenFlow network | 367 Kbits/sec |
| Flows in OpenFlow network | 944 Mbits/sec |

4.3. Experiment 2: Verifying *modify actions* in a flow

This experiment has been conducted to explore and verify more about the software flows. Software flows include actions other than forwarding to port, and are carried by modify actions message (Flow-Mod). The modify actions include modifying layer 2, layer 3 and layer 4 information such as modifying source and destination MAC or IP

address, modifying TCP source and destination port. These flows are useful in situations like NAT, MAC based forwarding and carrying flow from one switch to another switch etc.

The topology of the network can be seen in Figure 4.8, where three hosts from a network 192.168.56.0/24 are functional nodes in an OpenFlow network. The OpenFlow switch is configured in the aggregation mode, and scenario includes diverting all the traffic arriving at Host 2 to Host3.

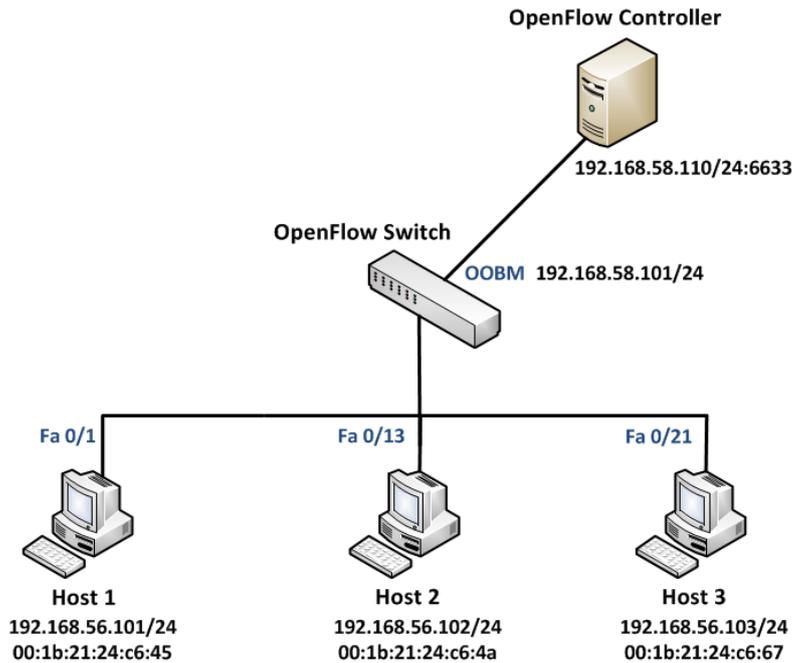


Figure 4.8. Network topology for Experiment 2

4.3.1. Flow entries

For this experiment, a total of three flows were configured: one from Host1 to Host2, second vice-versa and third from Host3 to Host1, which can be seen in Table 4.3 below.

Table 4.3. Flows for Experiment 2

| Flow # | Matching criterion | Actions |
|--------|-----------------------------------|--|
| 1 | in_port 1, dst_ip 192.168.56.102 | out_port 21, set_dst_ip 192.168.56.103 |
| 2 | in_port 13, dst_ip 192.168.56.101 | out_port 1 |
| 3 | in_port 21, dst_ip 192.168.56.101 | out_port 1 |

4.3.2. Results

ICMP ping query, iperf and netperf results revealed a functional OpenFlow network that diverted all incoming traffic to Host3, which was aimed at reaching Host2. Throughput results are similar to those of Experiment 1, and can be seen in Table 4.4 below.

Table 4.4. Throughput for Experiment 2

| Situation | Network throughput |
|-----------------------------------|--------------------|
| Before deploying OpenFlow network | 944 Mbits/sec |
| No flows in OpenFlow network | 336 Kbits/sec |
| Flows in OpenFlow network | 944 Mbits/sec |

4.4. Experiment 3: VLANs in OpenFlow network

This experiment has been conducted to explore about integration and behaviour of VLAN into OpenFlow network. Two VLANs with network 192.168.56.0/24 and 192.168.57.0/24 are considered, and each VLAN has two hosts within it. Each VLAN is configured into a separate switch, and switches are configured in the virtualization mode first and later the aggregation mode has also been tested. Both the aggregation and virtualization modes yielded similar performance and throughput statistics. The topology of the network in the aggregation mode can be seen in Figure 4.9, whereas with the virtualization mode addition of one more OpenFlow controller takes place as every instance in the virtualization mode demands a separate controller that has been performed later in Section 4.5.

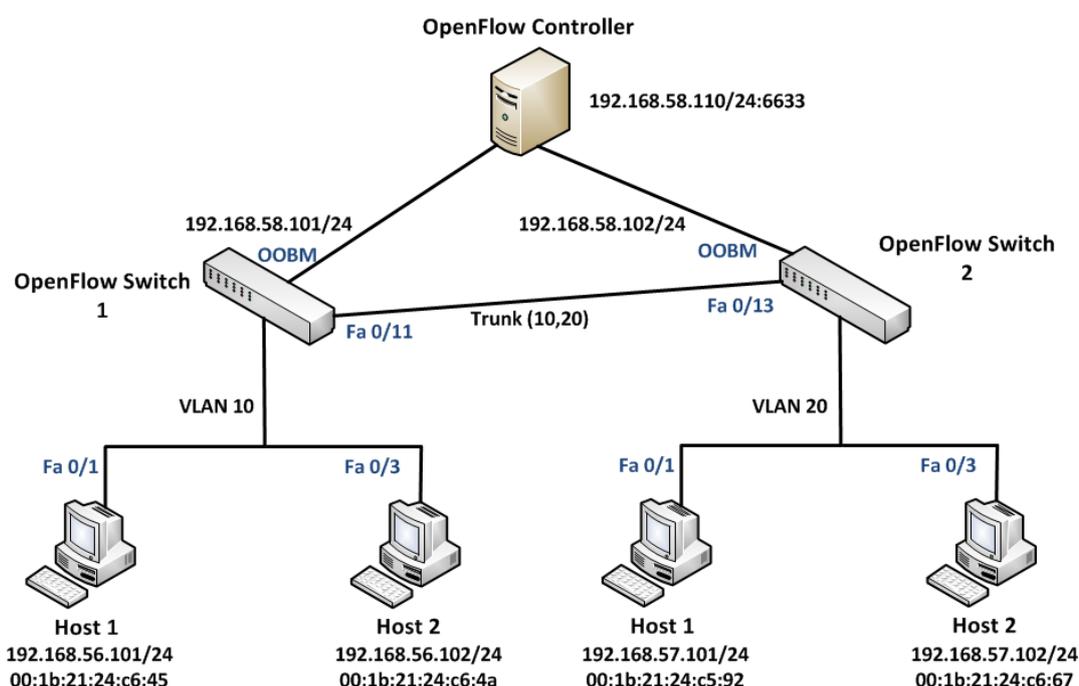


Figure 4.9. Network topology for Experiment 3

4.4.1. Flow entries

For this experiment, a total of six flows were configured for each switch: one from Host1 to Host2 in the same VLAN, second vice-versa, one flow for each host to reach hosts in other VLAN, and last two flows vice-versa, i.e., for reaching VLAN 10 hosts from VLAN 20 in switch 1. The flows for switch 1 and switch 2 can be seen in Table 4.5 and Table 4.6 respectively.

Table 4.5. Flows for switch 1 in Experiment 3

| Flow # | Matching criterion | Actions |
|--------|-----------------------------------|-------------|
| 1 | in_port 1, dst_ip 192.168.56.102 | out_port 3 |
| 2 | in_port 3, dst_ip 192.168.56.101 | out_port 1 |
| 3 | in_port 1, dst_ip 192.168.57.0 | out_port 11 |
| 4 | in_port 3, dst_ip 192.168.57.0 | out_port 11 |
| 5 | in_port 11, dst_ip 192.168.56.101 | out_port 1 |
| 6 | in_port 11, dst_ip 192.168.56.102 | out_port 3 |

Table 4.6. Flows for switch 2 in Experiment 3

| Flow # | Matching criterion | Actions |
|--------|-----------------------------------|-------------|
| 1 | in_port 1, dst_ip 192.168.57.102 | out_port 3 |
| 2 | in_port 3, dst_ip 192.168.57.101 | out_port 1 |
| 3 | in_port 1, dst_ip 192.168.56.0 | out_port 13 |
| 4 | in_port 3, dst_ip 192.168.56.0 | out_port 13 |
| 5 | in_port 13, dst_ip 192.168.57.101 | out_port 1 |
| 6 | in_port 13, dst_ip 192.168.57.102 | out_port 3 |

4.4.2. Results

ICMP ping query, iperf and netperf results revealed a functional VLAN for each switch in an OpenFlow network. However, inter-VLAN communication could not be established, when a host from one VLAN tried to reach each other host in other VLAN. The main reason included locating default gateway which can be seen in Figure 4.10.

There was not any router in the network topology that would solve this problem, although the HP switch was a layer 3 switch and VLAN was configured before enabling OpenFlow. Since OpenFlow looks into flow tables rather than looking into switch configuration, so VLAN configuration in HP switch could not resolve the issue. The solutions include attaching a router in the network or using RouteFlow, which have been tested in Section 4.8 and Section 4.9 respectively.

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|-----------|-------------------|----------------|----------|--------|--|
| 1 | 0.000000 | 10:60:4b:b7:47:bf | LLDP Multicast | LLDP | 61 | Chassis Id = 10:60:4b:b7:47:80 Port Id = TTL = 120 |
| 2 | 0.196709 | 10:60:4b:b7:47:bf | Broadcast | 0x8942 | 69 | Ethernet II |
| 3 | 0.689682 | 10:60:4b:b7:47:bf | LLDP Multicast | LLDP | 247 | Chassis Id = 10:60:4b:b7:47:80 Port Id = 1 TTL = 120 System Name = switch2 |
| 4 | 3.789638 | IntelCor_24:c6:45 | Broadcast | ARP | 42 | Who has 192.168.56.1? Tell 192.168.56.101 |
| 5 | 4.789448 | IntelCor_24:c6:45 | Broadcast | ARP | 42 | Who has 192.168.56.1? Tell 192.168.56.101 |
| 6 | 5.789323 | IntelCor_24:c6:45 | Broadcast | ARP | 42 | Who has 192.168.56.1? Tell 192.168.56.101 |
| 7 | 7.789002 | IntelCor_24:c6:45 | Broadcast | ARP | 42 | Who has 192.168.56.1? Tell 192.168.56.101 |
| 8 | 8.789907 | IntelCor_24:c6:45 | Broadcast | ARP | 42 | Who has 192.168.56.1? Tell 192.168.56.101 |
| 9 | 9.789832 | IntelCor_24:c6:45 | Broadcast | ARP | 42 | Who has 192.168.56.1? Tell 192.168.56.101 |
| 10 | 11.789717 | IntelCor_24:c6:45 | Broadcast | ARP | 42 | Who has 192.168.56.1? Tell 192.168.56.101 |
| 11 | 12.789560 | IntelCor_24:c6:45 | Broadcast | ARP | 42 | Who has 192.168.56.1? Tell 192.168.56.101 |
| 12 | 13.789435 | IntelCor_24:c6:45 | Broadcast | ARP | 42 | Who has 192.168.56.1? Tell 192.168.56.101 |
| 13 | 15.002883 | 10:60:4b:b7:47:bf | LLDP Multicast | LLDP | 61 | Chassis Id = 10:60:4b:b7:47:80 Port Id = TTL = 120 |
| Frame 4: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) | | | | | | |
| ▶ Ethernet II, Src: IntelCor_24:c6:45 (00:1b:21:24:c6:45), Dst: Broadcast (ff:ff:ff:ff:ff:ff) | | | | | | |
| ▼ Address Resolution Protocol (request) | | | | | | |
| Hardware type: Ethernet (1) | | | | | | |
| Protocol type: IP (0x0800) | | | | | | |
| Hardware size: 6 | | | | | | |
| Protocol size: 4 | | | | | | |
| Opcode: request (1) | | | | | | |
| [Is gratuitous: False] | | | | | | |
| Sender MAC address: IntelCor_24:c6:45 (00:1b:21:24:c6:45) | | | | | | |
| Sender IP address: 192.168.56.101 (192.168.56.101) | | | | | | |
| Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00) | | | | | | |
| Target IP address: 192.168.56.1 (192.168.56.1) | | | | | | |

Figure 4.10. Default gateway issue in Experiment 3

Within a VLAN, the flows were working properly and throughput results confirm their steady operation. The throughput results can be seen in Table 4.7 below.

Table 4.7. Throughput for Experiment 3

| Situation | Network throughput |
|-----------------------------------|--------------------|
| Before deploying OpenFlow network | 944 Mbits/sec |
| No flows in OpenFlow network | 383 Kbits/sec |
| Flows in OpenFlow network | 944 Mbits/sec |

4.5. Experiment 4: Fail-safe in case of controller failure

This experiment has been conducted to explore about fail-standalone mode for controller connection in HP OpenFlow switches. The fail-standalone mode will be useful in cases where controller connection breaks down, and network faces a single point of failure. More than one controller can be deployed and act as load-balance or distribute functionalities, and such slicing and distribution can be achieved by using FlowVisor [17][50], however other solution include configuring HP switch in fail-standalone mode and avoid the network breakdown in case of controller failure.

Considering the network topology for Experiment 4 shown in Figure 4.11, another controller is added here and each switch is configured in the virtualization mode. An instance is created in each switch having VLAN 10 and VLAN 20 members respectively with fail-standalone mode activated. HP switches are layer 3 switches, they can handle IP routing as well and the traditional VLAN configuration has been made in both of them.

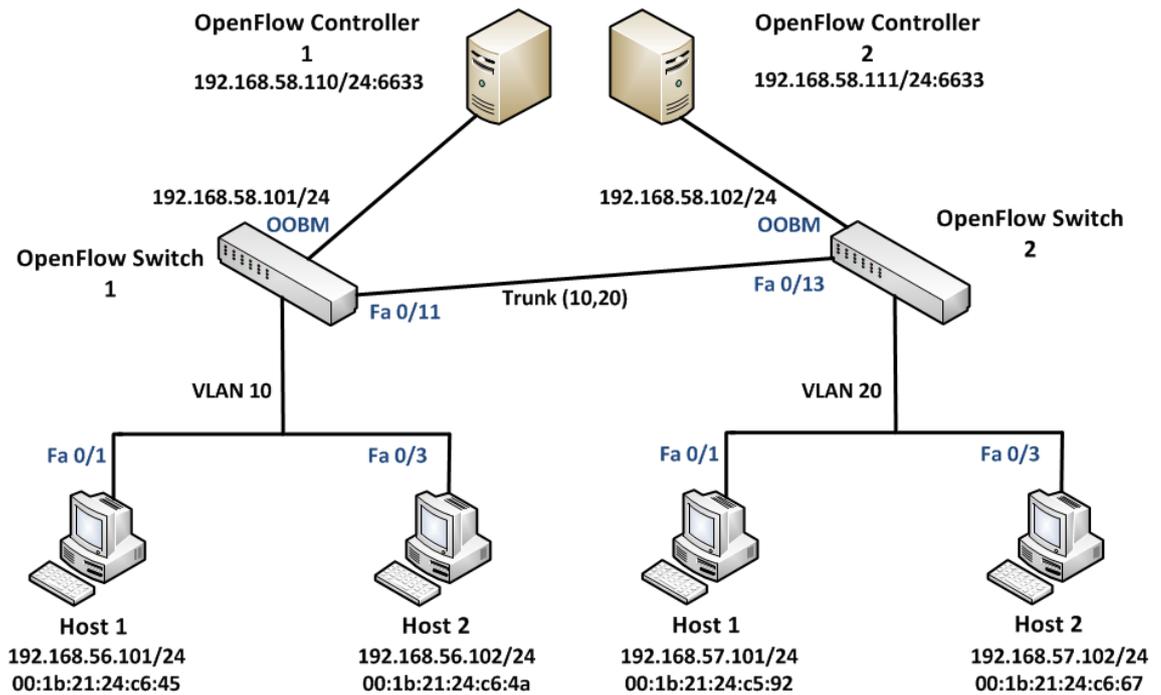


Figure 4.11. Network topology for Experiment 4

4.5.1. Flow entries

For this experiment, the flow entries are similar to those defined in Experiment 3, and can be seen in Table 4.8 and Table 4.9 for switch 1 and switch 2 respectively.

Table 4.8. Flows for switch 1 in Experiment 4

| Flow # | Matching criterion | Actions |
|--------|-----------------------------------|-------------|
| 1 | in_port 1, dst_ip 192.168.56.102 | out_port 3 |
| 2 | in_port 3, dst_ip 192.168.56.101 | out_port 1 |
| 3 | in_port 1, dst_ip 192.168.57.0 | out_port 11 |
| 4 | in_port 3, dst_ip 192.168.57.0 | out_port 11 |
| 5 | in_port 11, dst_ip 192.168.56.101 | out_port 1 |
| 6 | in_port 11, dst_ip 192.168.56.102 | out_port 3 |

Table 4.9. Flows for switch 2 in Experiment 4

| Flow # | Matching criterion | Actions |
|--------|-----------------------------------|-------------|
| 1 | in_port 1, dst_ip 192.168.57.102 | out_port 3 |
| 2 | in_port 3, dst_ip 192.168.57.101 | out_port 1 |
| 3 | in_port 1, dst_ip 192.168.56.0 | out_port 13 |
| 4 | in_port 3, dst_ip 192.168.56.0 | out_port 13 |
| 5 | in_port 13, dst_ip 192.168.57.101 | out_port 1 |
| 6 | in_port 13, dst_ip 192.168.57.102 | out_port 3 |

4.5.2. Results

With the virtualization mode and VLAN configuration, OpenFlow network has been first tested with an active OpenFlow controller connection using ping, iperf and netperf for each VLAN. However, the inter-VLAN communication has not been established due to the default gateway issue observed in Section 4.4. Later, the OpenFlow controller has been stopped, and fail-standalone mode is activated that stops looking into OpenFlow flow tables. It starts looking into the configuration of each switch, i.e., traditional forwarding table in each switch and network connectivity resumes after a delay of approximately 4 seconds. The switches are still in OpenFlow mode and are sending TCP sync messages from their OOBM port to the controller, but they fail to receive the TCP sync acknowledgment messages that can be seen in Figure 4.12. The same messages are transmitted when controller connection breaks, however in this mode, the traffic is forwarded using each switch's own configuration and routing table rather than waiting for OpenFlow controller.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|------------|-------------------|------------------------|----------|--------|---|
| 340 | 266.588526 | Cisco.dd:85:09 | Spanning-tree-(for-br) | STP | 60 | Conf. Root = 32768/1/00:09:e8:dd:85:00 Cost = 0 Port = 0x8009 |
| 341 | 261.897189 | 192.168.58.102 | 192.168.58.110 | TCP | 78 | 62983 > 6633 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=2 SACK_PERM=1 TSval=1994693810 TSecr=0 |
| 342 | 261.897212 | 192.168.58.110 | 192.168.58.102 | TCP | 54 | 6633 > 62983 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 343 | 262.371207 | 192.168.58.101 | 192.168.58.110 | TCP | 78 | 49573 > 6633 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=2 SACK_PERM=1 TSval=12522760 TSecr=0 |
| 344 | 262.371218 | 192.168.58.110 | 192.168.58.101 | TCP | 54 | 6633 > 49573 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 345 | 262.588531 | Cisco.dd:85:09 | Spanning-tree-(for-br) | STP | 60 | Conf. Root = 32768/1/00:09:e8:dd:85:00 Cost = 0 Port = 0x8009 |
| 346 | 264.588566 | Cisco.dd:85:09 | Spanning-tree-(for-br) | STP | 60 | Conf. Root = 32768/1/00:09:e8:dd:85:00 Cost = 0 Port = 0x8009 |
| 347 | 266.588721 | Cisco.dd:85:09 | Spanning-tree-(for-br) | STP | 60 | Conf. Root = 32768/1/00:09:e8:dd:85:00 Cost = 0 Port = 0x8009 |
| 348 | 266.909965 | Hewlett-63:4a:38 | 10:60:4b:38:d2:c1 | ARP | 42 | Who has 192.168.58.102? Tell 192.168.58.110 |
| 349 | 266.910190 | 10:60:4b:38:d2:c1 | Hewlett-63:4a:38 | ARP | 64 | 192.168.58.102 is at 10:60:4b:38:d2:c1 |
| 350 | 267.373967 | Hewlett-63:4a:38 | 10:60:4b:b7:47:81 | ARP | 42 | Who has 192.168.58.101? Tell 192.168.58.110 |
| 351 | 267.374179 | 10:60:4b:b7:47:81 | Hewlett-63:4a:38 | ARP | 64 | 192.168.58.101 is at 10:60:4b:b7:47:81 |

Frame 348: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on Ethernet II, Src: Hewlett-63:4a:38 (00:18:fe:63:4a:38), Dst: 10:60:4b:38:d2:c1 (10:60:4b:38:d2:c1)

Address Resolution Protocol (request)

- Hardware type: Ethernet (1)
- Protocol type: IP (0x0800)
- Hardware size: 6
- Protocol size: 4
- Opcode: request (1)
- [Is gratuitous: False]
- Sender MAC address: Hewlett-63:4a:38 (00:18:fe:63:4a:38)
- Sender IP address: 192.168.58.110 (192.168.58.110)
- Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00)
- Target IP address: 192.168.58.102 (192.168.58.102)

Figure 4.12. Switches searching for controller in fail-standalone mode

After the network connectivity resumes, inter-VLAN communication has been established using traditional VLAN configuration of the switch and throughput remains at 944 Mbits/sec. The fail-standalone mode can also be effective in the aggregation mode configuration provided the needed configuration resides in the switch. The throughput achieved in Experiment 4 can be seen in Table 4.10.

Table 4.10. Throughput for Experiment 4

| Situation | Network throughput |
|--|--------------------|
| Before deploying OpenFlow network | 944 Mbits/sec |
| No flows in OpenFlow network | 343 Kbits/sec |
| Flows in OpenFlow network | 944 Mbits/sec |
| Controller connection breaks in OpenFlow network | 944 Mbits/sec |

The amount of delay (timeout at 4 seconds) is significantly high in the modern networks, but it is at least better option than the total tear-down of the network.

4.6. Experiment 5: OpenFlow in hybrid mode

This experiment has been conducted to explore about OpenFlow operation in hybrid mode, i.e., OpenFlow network will be running in parallel with the traditional network. HP switch has been configured in virtualization mode to support hybrid mode, and network includes three different VLANs: VLAN 10, VLAN 20 and VLAN 30. VLAN 10 contains OpenFlow network, while the rest two VLANs are for non-OpenFlow traditional network. The network topology for Experiment 5 can be seen in Figure 4.13.

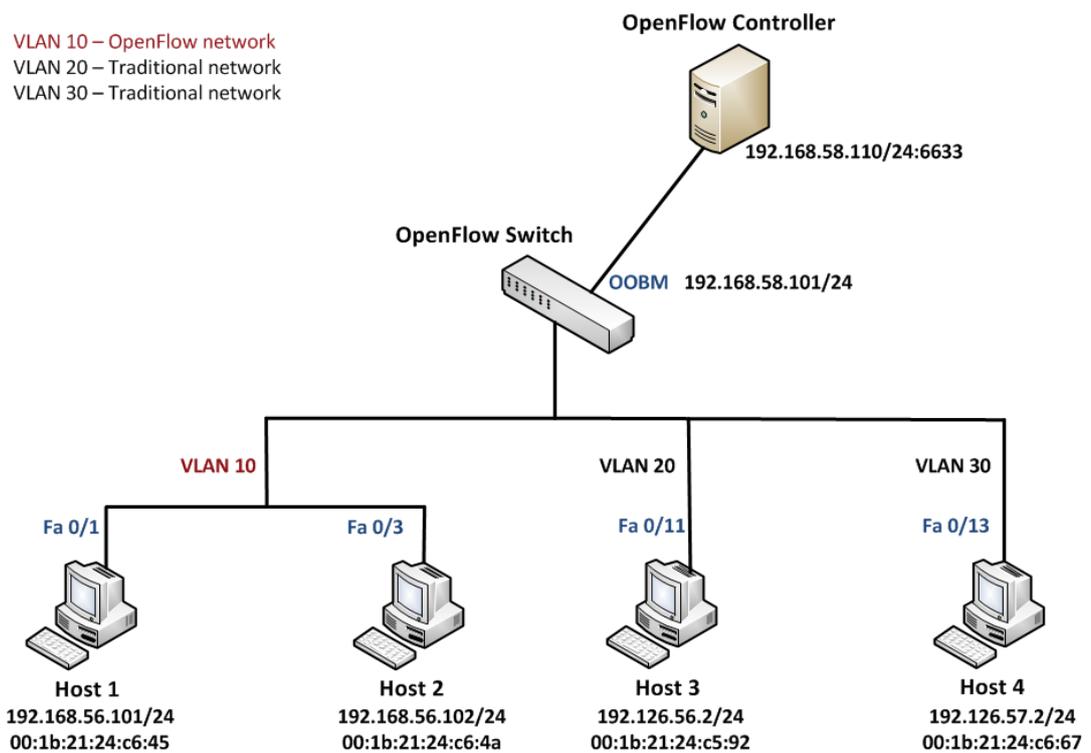


Figure 4.13. Network topology for Experiment 5

4.6.1. Flow entries

For this experiment, there are only two flow entries for OpenFlow network, i.e., VLAN 10, which can be seen in Table 4.11 below.

Table 4.11. Flows for Experiment 5

| Flow # | Matching criterion | Actions |
|--------|----------------------------------|------------|
| 1 | in_port 1, dst_ip 192.168.56.102 | out_port 3 |
| 2 | in_port 3, dst_ip 192.168.56.101 | out_port 1 |

4.6.2. Results

OpenFlow network has been tested using ping, iperf and netperf utilities and it yielded the expected throughput of 941 Mbits/sec. The traditional network VLAN 20 and VLAN 30 can reach each other, but cannot reach OpenFlow network. Both OpenFlow and traditional network run in parallel, while maintaining network throughput. Furthermore, this configuration was tested using fail-standalone mode, and it showed smooth operations there as well. This experiment demonstrated the integration of OpenFlow with traditional networking environment.

4.7. Experiment 6: MAC based forwarding

This experiment has been conducted to explore about MAC based forwarding and software flows in an OpenFlow network consisting of multiple switches. Two hosts from a subnet 192.168.56.0/24 are deployed over three switches and flows are configured to demonstrate it. HP switch has been configured in the aggregation mode, and the network topology can be seen in Figure 4.14.

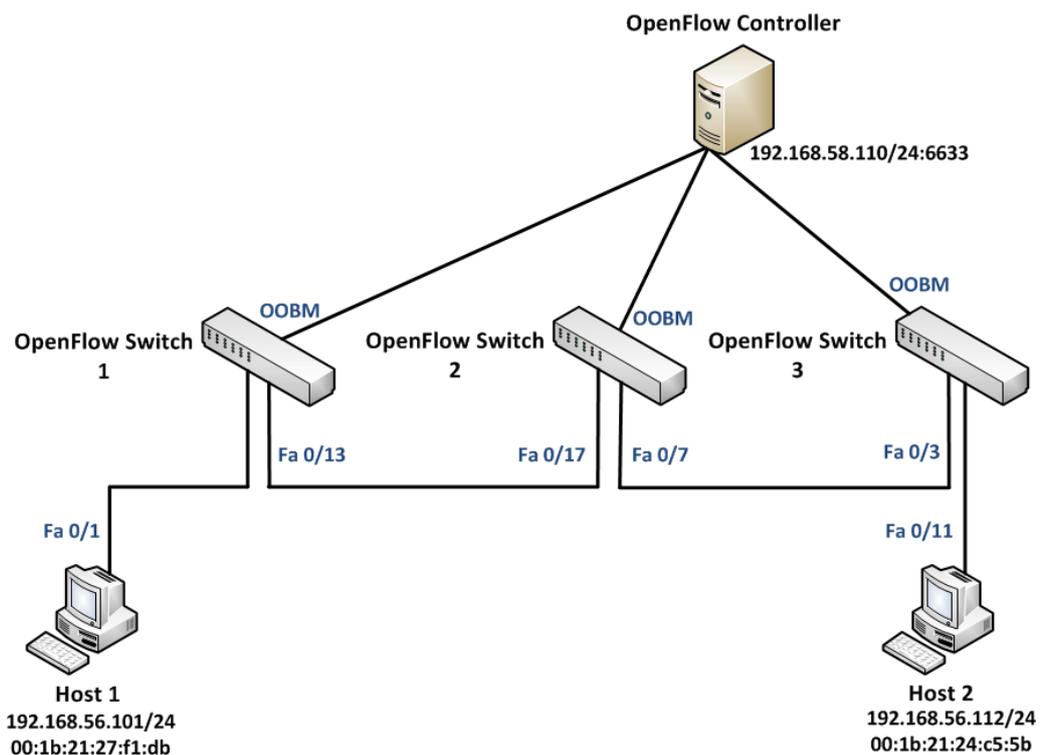


Figure 4.14. Network topology for Experiment 6

4.7.1. Flow entries

For this experiment, there are only two flow entries for each switch in OpenFlow network, which can be seen in Table 4.12, Table 4.13 and Table 4.14 respectively. In flow table MAC addresses 33:33:33:33:33:33 and 11:11:11:11:11:11 are arbitrary MAC addresses that do not belong to any device in the network, but they act as flow match field.

Table 4.12. Flows for switch 1 in Experiment 6

| Flow # | Matching criterion | Actions |
|--------|-----------------------------------|--|
| 1 | in_port 1, dst_ip 192.168.56.112 | out_port 13, set-dst-mac 33:33:33:33:33:33 |
| 2 | in_port 13, dst_ip 192.168.56.101 | out_port 1, set-dst-mac 00:1b:21:27:f1:db |

Table 4.13. Flows for switch 2 in Experiment 6

| Flow # | Matching criterion | Actions |
|--------|---------------------------------------|-------------|
| 1 | in_port 17, dst_mac 33:33:33:33:33:33 | out_port 7 |
| 2 | in_port 7, dst_mac 11:11:11:11:11:11 | out_port 17 |

Table 4.14. Flows for switch 3 in Experiment 6

| Flow # | Matching criterion | Actions |
|--------|-----------------------------------|--|
| 1 | in_port 3, dst_ip 192.168.56.112 | out_port 11, set-dst-mac 00:1b:21:24:c5:5b |
| 2 | in_port 11, dst_ip 192.168.56.101 | out_port 3, set-dst-mac 11:11:11:11:11:11 |

4.7.2. Results

OpenFlow network has been tested using ping, iperf and netperf utilities and it yielded the expected throughput of 944 Mbits/sec. If any incorrect flow is configured, then one node cannot reach the node at the other end, and troubleshooting is performed by looking at the flow table of each individual switch. The flow table packet count parameter in the flow table of HP switch helps to determine where the error lies. If the packets have not passed through a specific flow, the packet count parameter for that specific flow will not reveal any increment in statistics, and that's how an error can be detected. The possible reasons could be incorrect configuration for the flow especially with match fields.

Compared to earlier experiments where network included single switch, locating an error in flow configuration here takes more effort. Depending upon the nature of error in flow configuration, it may reach other node by broadcasting from OpenFlow controller at reduced throughput. The throughput of network can be seen in Table 4.15 below.

Table 4.15. Throughput for Experiment 6

| Situation | Network throughput |
|-----------------------------------|--------------------|
| Before deploying OpenFlow network | 944 Mbits/sec |
| No flows in OpenFlow network | 324 Kbits/sec |
| Flows in OpenFlow network | 944 Mbits/sec |
| Any flow not configured properly | 0 - 380 Kbits/sec |

MAC based forwarding and routing is beneficial for scalability, since MAC address is changed at origin, packet is processed through switches and at destination switch the MAC address is changed to destination host. When there are abundant nodes in the network, using this aggregate flow approach the flows need to be changed at source and destination only, while the flow configuration for intermediate switches remains constant unless new subnets are introduced or new switches are added into the network. This approach can be concluded as an easier flow rule to implement and a step forward towards the aggregate flows. In a typical network with multiple switches, this approach offers a challenge to select and map arbitrary MAC addresses to the real switches in the network, and a database must be built to cope with such issue so that like IP addresses, the correct and unique mapping can be achieved.

4.8. Experiment 7: VLANs with a router

This experiment has been conducted to explore about OpenFlow protocol in multiple VLANs with different subnets. Similar experiments have been conducted before in Section 4.4, but they could not reach other subnet due to absence of default gateway. In this experiment, a router is used to solve the issue that forwards traffic from one subnet to other in OpenFlow network. Two hosts from each subnet 192.168.56.0/24 and 192.168.57.0/24 are configured in VLAN 10 and VLAN 20 respectively. HP switch has been configured in the aggregation mode, and network topology can be seen in Figure 4.15 below. This experiment can also be conducted in the virtualization mode.

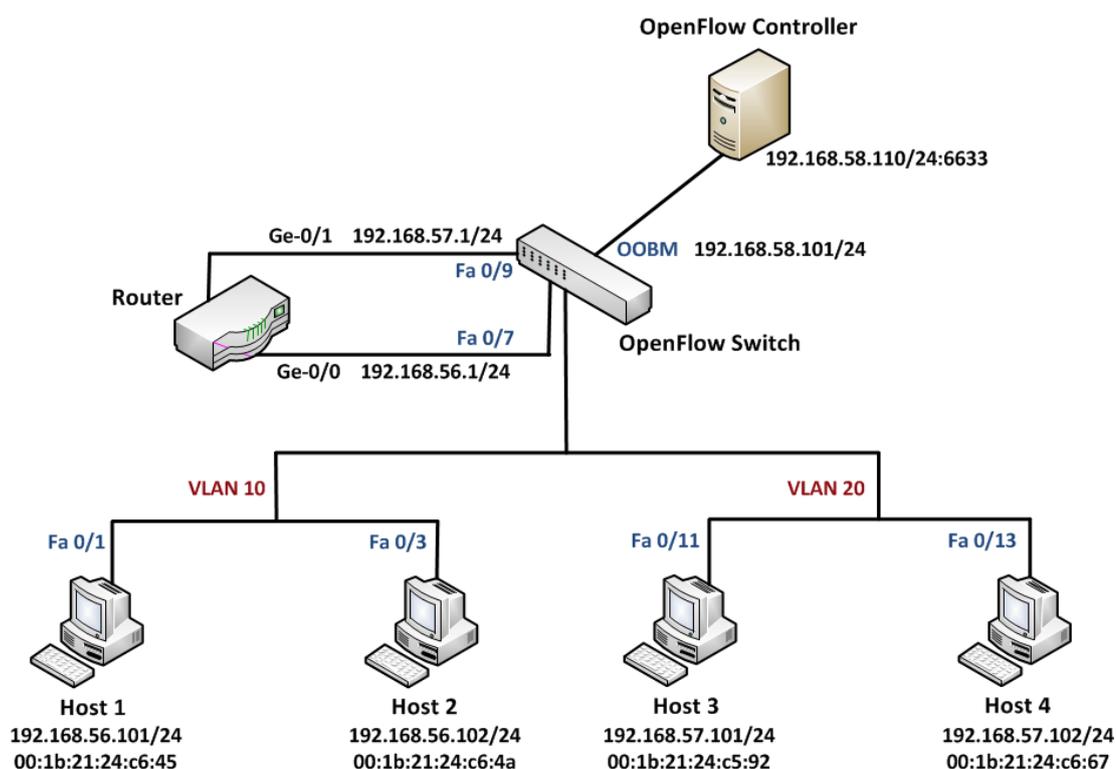


Figure 4.15. Network topology for Experiment 7

4.8.1. Flow entries

For this experiment, there are a total of fourteen flows entries needed: two within same VLAN for each VLAN and ten for router processing, i.e., carrying traffic from VLAN 10 to VLAN 20 via router and vice-versa. The flow entries can be seen in Table 4.16 below.

Table 4.16. Flows for Experiment 7

| Flow # | Matching criterion | Actions |
|--------|--------------------------------------|-------------|
| 1 | in_port 1, dst_ip 192.168.56.102 | out_port 3 |
| 2 | in_port 3, dst_ip 192.168.56.101 | out_port 1 |
| 3 | in_port 11, dst_ip 192.168.57.102 | out_port 13 |
| 4 | in_port 13, dst_ip 192.168.57.101 | out_port 11 |
| 5 | in_port 1, dst_ip 192.168.57.101/24 | out_port 7 |
| 6 | in_port 3, dst_ip 192.168.57.101/24 | out_port 7 |
| 7 | in_port 7, dst_ip 192.168.57.101/24 | out_port 9 |
| 8 | in_port 9, dst_ip 192.168.57.101 | out_port 11 |
| 9 | in_port 9, dst_ip 192.168.57.102 | out_port 13 |
| 10 | in_port 11, dst_ip 192.168.56.101/24 | out_port 9 |
| 11 | in_port 13, dst_ip 192.168.56.101/24 | out_port 9 |
| 12 | in_port 9, dst_ip 192.168.56.101/24 | out_port 7 |
| 13 | in_port 7, dst_ip 192.168.56.101 | out_port 1 |
| 14 | in_port 7, dst_ip 192.168.56.102 | out_port 3 |

4.8.2. Results

OpenFlow network has been tested using ping, iperf and netperf utilities and it yielded the expected throughput of 944 Mbits/sec. The availability of a router in the network makes it possible to reach from one subnet to other subnet. If the flows towards router are not configured properly and/or not present completely, then OpenFlow network is still functional but at a reduced rate. The reasons for reduced rate have been observed in earlier experiments and they are validated in this experiment. The throughput of network can be seen in Table 4.17 below.

Table 4.17. Throughput for Experiment 7

| Situation | Network throughput |
|---|--------------------|
| Before deploying OpenFlow network | 944 Mbits/sec |
| No flows in OpenFlow network | 328 Kbits/sec |
| Flows in OpenFlow network | 944 Mbits/sec |
| Flows to router not configured properly | 360 Kbits/sec |

Router processes OpenFlow packets smoothly, since OpenFlow fields are appended to normal IP packet at the trailer side. Router looks into layer 3 information in packet and processes it onwards. It can be concluded that routers are needed in the deployment of OpenFlow network with multiple subnets in LANs, and an OpenFlow switch cannot override the role of routers in traditional LANs.

4.9. Experiment 8: VLANs with RouteFlow

This experiment has been conducted to explore about RouteFlow in a physical network. The network topology is similar to earlier experiment, and can be seen in Figure 4.16 where OpenFlow controller has been replaced by RouteFlow server. Two hosts from each subnet 192.168.56.0/24 and 192.168.57.0/24 are configured in VLAN 10 and VLAN 20 respectively. HP switch has been configured in the aggregation mode, while this experiment can also be conducted in the virtualization mode.

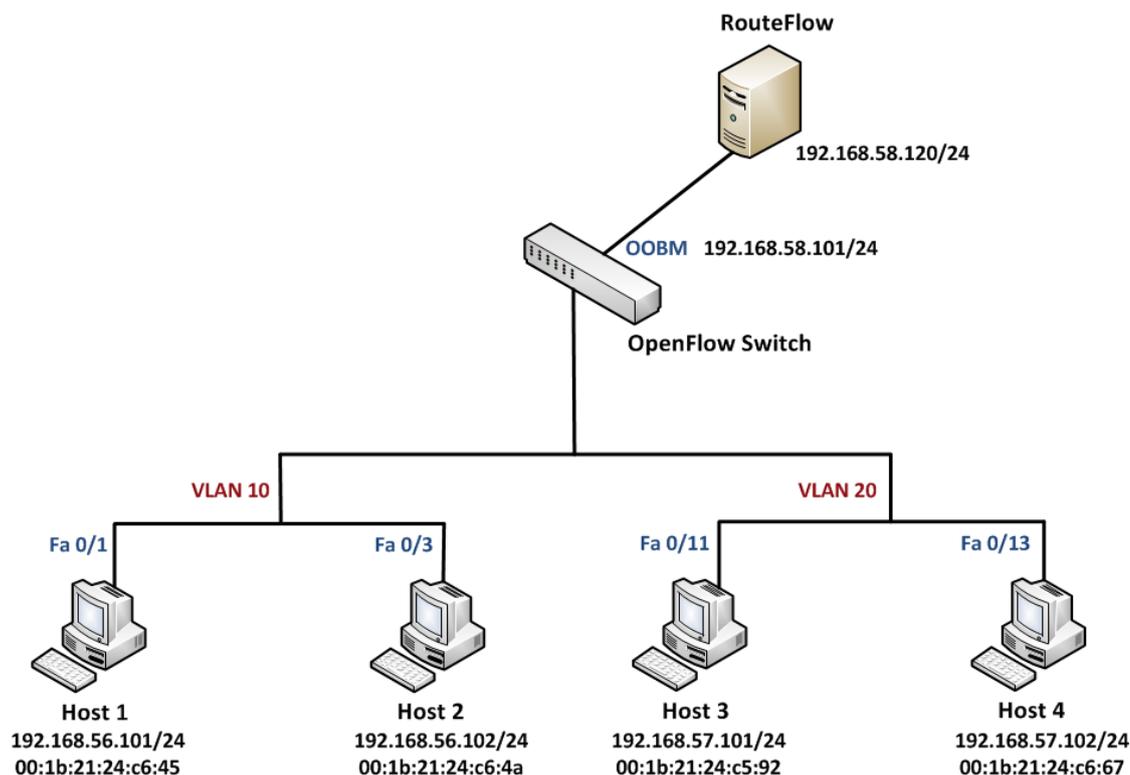


Figure 4.16. Network topology for Experiment 8

RouteFlow machine resides NOX controller, RouteFlow (RF) proxy server, Open vSwitch and virtual machine engines inside it. A bash script from Projectw (RouteFlow on a live DVD) contains the configuration for activating RF proxy server, Open vSwitch and virtual machine hosts using lxc-console for the hosts in network. For this network topology, four lxc hosts named b1, b2, b3 and b4 are created, and these virtual hosts are

linked to physical ports on a hardware switch via OOBM port rather than linking to virtual ports on virtual RF machines.

4.9.1. Flow entries

For this experiment, there are a total of twelve flows entries are installed into switch via a controller on RouteFlow machine, which can be seen in Table 4.18. A bash script in RouteFlow needs datapath ID (DPID) of the OpenFlow switch to install flows there. For the flows that are going towards the other subnet, they are directed towards the controller port (OOBM), where the RF proxy server takes them to gateway of other subnet and subsequently it reaches the host in other subnet or VLAN.

Table 4.18. Flows for Experiment 8

| Flow # | Matching criterion | Actions |
|--------|---|---------------------|
| 1 | in_port 1, dst_ip 192.168.56.102 | out_port 3 |
| 2 | in_port 3, dst_ip 192.168.56.101 | out_port 1 |
| 3 | in_port 11, dst_ip 192.168.57.102 | out_port 13 |
| 4 | in_port 13, dst_ip 192.168.57.101 | out_port 11 |
| 5 | in_port 1, dst_ip 192.168.57.101/24 | out_port controller |
| 6 | in_port 3, dst_ip 192.168.57.101/24 | out_port controller |
| 7 | in_port controller, dst_ip 192.168.57.101 | out_port 11 |
| 8 | in_port controller, dst_ip 192.168.57.102 | out_port 13 |
| 9 | in_port 11, dst_ip 192.168.56.101/24 | out_port controller |
| 10 | in_port 13, dst_ip 192.168.56.101/24 | out_port controller |
| 11 | in_port controller, dst_ip 192.168.56.101 | out_port 1 |
| 12 | in_port controller, dst_ip 192.168.56.102 | out_port 3 |

4.9.2. Results

RouteFlow offers virtualized gateway for a virtual network consisting of virtual switch and virtual hosts. After making the needed changes in configuration of RouteFlow source code for mapping virtual interfaces to physical interfaces in HP switch, OpenFlow network could not establish connectivity. When the bonding of lxc-console virtual hosts to physical ports in switch was taking place, it didn't succeed.

RouteFlow defines software flows for forwarding to controller for gateway lookup and further processing that may restrict throughput depending upon ASICs that HP switch have been used inside it. Some recent manufactures use "V2 ASICs" that supports more matches in hardware than older switches. HP switch ASICs can be one of the reasons for incompatibility with RouteFlow operation; however RouteFlow has been tested successfully with Pronto switches earlier [51].

4.10. Discussion on results

For each experiment, the flows were configured, and then ICMP ping messages were used to determine the establishment of connectivity. After that, flow counters in the flowtable of switch were looked up, and throughput analysis were carried out to determine whether flows are correctly configured and packets are forwarded using flows. All the flows were manually configured and installed to switches via OpenFlow controller. SDN aims at developing a programmable network, however flow installations in an OpenFlow network appear to be all manual work at this stage. Floodlight controller keeps track of switches and hosts attached to it, but the newly added host cannot be detected instantly unless it reaches the Floodlight controller's IP address via HTTP or ICMP ping requests. On the other side, switches upon activating OpenFlow mode start sending TCP sync packets to controller at the TCP 6633 port; and upon getting TCP sync acknowledgements from the controller, the switch(es) and controller pair each other and the switch(es) is instantly joined into the OpenFlow network.

When a flow in the flow table is not matched against the newly added host, the packet goes to Floodlight controller via OpenFlow enabled switch for further processing in encapsulated packet. Floodlight controller registers the host with its IP address, MAC address, port number and switch attached to, and sends the packet back to switch along with the action to be performed. Usually, the decision made by the controller include installing the relative flow in case of reactive flows or broadcast the packets if a flow match is not found. With the decision to broadcast, the packets ultimately reach their destination via broadcasting but the overall throughput is reduced significantly from 941 Mbps to approximately 340 Kbps.

In cases, where due to incorrect flow configuration, an OpenFlow network can not establish connectivity or a reduced throughput is achieved, then the primary troubleshooting lies in looking at the flow counters such as packets or bytes passed through a specific flow. In a typical OpenFlow network, plenty of switches will be residing, and checking flow counters at each switch will be an effortful task. However, similar information can also be checked from Floodlight controller's web interface, where it displays a summary of each flow. Other approach include looking into the Wireshark capture to inspect Packet-Out messages with the flooding decision made that results into the reduced throughput.

Initially, each newly attached node in the network must browse internet so that HTTP packets are sent ultimately to its relative OpenFlow controller, and thereby the controller registers newly added node to its database. Furthermore, an automation framework or program in Python or other script can be developed that would be running on the OpenFlow controller. The automation framework would be periodically looking into Floodlight controller database, and detecting if any changes are made.

Upon detecting any changes in the network, the automation framework must formulate the flows using switch DPID, and other necessary parameters and install those flows immediately into the flow table of the switch. As the network scales, the flow

computation method must be scalable, fast and intelligent enough to populate aggregate flows rather than individual flows. Apart from it, an OpenFlow network can be sliced using FlowVisor approach to make the automation framework a smart framework, where switch DPID plays an important role. Certain subnet or a region of subnet can be allocated to a switch, and user must manually inform the automation framework that which switch DPID corresponds to which region of OpenFlow network. The experiments conducted within the same subnet in LAN were successful, but when the network is distributed over different subnets as in typical LANs, the flows from one subnet cannot reach the other subnet. OpenFlow protocol does not provide any layer 3 services till v1.3, but it simply defines a path that a source host must take to reach a destination. When a router is utilized to solve the gateway issues, the network connectivity in LAN resumes.

Another interesting application, RouteFlow, can resolve the gateway issues as well by acting as a virtual gateway, but its complexity limits its advantages. When any new host is added or removed from the network, RouteFlow must be stopped, the configuration files need to be updated and it must be re-started. It takes a considerable time of about 5 - 10 seconds in the whole process, during that time the network connectivity breaks down and thereby making it unfeasible for the production networks.

OpenFlow controller is the heart of an OpenFlow network, and its failure tears down the whole network operations. Compared to centralized approach, distributed approach must be used and some applications like FlowVisor have been developed. The other approach with centralized controller infrastructure includes configuring the network with traditional configuration, and then enabling OpenFlow network in fail-standalone mode as in HP switch, and installing the needed flows. In such cases, when OpenFlow controller breaks down, the switch starts looking at its traditional forwarding table of switch rather than looking into OpenFlow flow table. With the traditional configuration in place, the network connectivity resumes after a significant delay of 4 seconds, but the OpenFlow network is totally vanished. An alert can be sent to the network administrator, and until the OpenFlow controller is fixed the network performance is not degraded and after fixing the controller connection, the OpenFlow network resumes.

OpenFlow protocol is still not mature enough to be deployed in the LANs due to its slower integration into hardware and lack of support for layer 3 protocols till v1.3. Furthermore, the deployment model does not suit the service providers, since hybrid deployment scenario is not feasible enough due to its compatibility and isolation with the existing LANs. Apart from hybrid model, replacing all the network equipments with OpenFlow enabled equipments is not a worthy solution from the economic as well as service aspects. Therefore, considering the current situation OpenFlow protocol can be deployed as test bed in smaller networks like campus area networks to isolate the traffic: experimental protocol traffic and research traffic. Such isolation has been tested with LAN and this has been one of the founding reasons for the OpenFlow protocol.

5. CONCLUSIONS AND FUTURE WORK

5.1. Conclusions

In this thesis, the integration and benefits of OpenFlow protocol in LAN environment have been analyzed. In the beginning of the thesis, a literature survey has been made to explore and understand more about SDN, OpenFlow protocol, their limits, pros and cons. Later, different vendors were searched for the support of OpenFlow protocol into the networking devices to find an OpenFlow enabled switch with better features offered. Although all the features of OpenFlow v1.0 have not been incorporated into networking hardware, at this point hardware poses limitations towards utilizing all benefits offered by OpenFlow.

In order to handle the growing data and make sustainable profits, the traditional networking infrastructure must adapt dynamically, but the limitations imposed by the complexity, vendor dependence and QoS requirements have lead to a static network. SDN overrides these limits and helps in making a dynamic network where the network control and management can be independent, programmable and less effortful.

OpenFlow protocol is a normal application layer protocol that is encapsulated inside TCP, IPv4 and Ethernet format. When switch is operated in OpenFlow mode, it makes forwarding decisions by looking into its flow table rather than looking into the traditional forwarding table of the switch. A flow table matches incoming packets by parameters including input port, source and destination MAC or IP address and applies relevant actions on it. In HP switches, the limitation for a specific input port for each flow to be matched multiplies the total number of flows needed that could have been reduced by using wildcard input port entry for each flow i.e., aggregate flows.

In cases, where incoming packets do not match with any flow in the flow table, they are sent to OpenFlow controller for further decision. Depending upon the flow type, controller decides either to install the relative flow or broadcast the traffic. In case of broadcasting decision, the packets eventually reach the destination at the significantly reduced throughput from 941 Mbps to approx. 340 Kbps. Such a drastic reduction in throughput is not well suited to production networks therefore a proper care must be taken to configure flows. However, in HP switches this consultation mode can be disabled at the risk of losing connectivity in the network.

Addition of the flows into the network did not help to make a dynamic network, and since the process of flow configuration and installation to respective switch(es) via OpenFlow controller(s) was a manual process. An automation framework can be developed to cope with the changing network state and adding or removing flows respectively. The controller poses a single point of failure and DoS attacks can lead to tear down

of network, thus Floodlight controller cannot be accessed outside of its subnet. Therefore, a secure tunnel or proper firewalls must be established for network administrators to install flows to the controller and monitor its state remotely. When network expansion takes place, then newly added switches are instantly paired by controller(s) in contrast to newly added nodes in the OpenFlow network. In an OpenFlow network, controllers must be configured in distributed mode rather than centralized mode where single break down affects the whole network performance. Furthermore, load balancing and standby mechanism can also be applied to it.

The flow concept can be interpreted as avoiding routers in a network, but in fact flows provide a path for packets to follow, flows do not override the features of gateway and those offered by routing protocols such as Border Gateway Protocol (BGP). Therefore routers will be there in LANs, but the flow concept offers more scalability and flexibility to a network in cases where network upgradation is needed. An interesting project called RouteFlow provides a virtual gateway among different subnets in an OpenFlow network, and changes the flow table in accordance with routing decisions in popular protocols including RIP and OSPF. The limitation of re-starting RouteFlow for each change in configuration makes it unsuitable for a scalable network and offers serious risk to the network performance.

Considering the integration of OpenFlow with traditional LANs, the successful integration of OpenFlow inside a single subnet in LANs have been observed in the conducted experiments. OpenFlow v1.0 and beyond till v1.3 do not support layer 3 protocols thereby OpenFlow is not fully integrated with the traditional LANs at layer 3. Regarding its deployment, both networks can run side by side in hybrid mode, but flows only control the OpenFlow network. In hybrid approach, an OpenFlow controller does not control the whole hybrid network for scalability or upgradation, adding no more benefits to traditional LANs. Therefore, whole networking hardware must be replaced with OpenFlow enabled hardware to gain a programmable and independent control over the network, which again is not an optimal solution from the economic as well as service aspect. However, an interesting project LegacyFlow aims at utilizing a few OpenFlow enabled devices with the traditional networking infrastructure, and offers the benefits of an OpenFlow network by translating flows into the relative CLI configuration commands for the traditional switches.

OpenFlow protocol is still not at an acceptable level to be deployed in the LANs due to its lack of support for layer 3 protocols till v1.3 and its slow integration into hardware. Furthermore, its deployment models are not feasible for the network operators and service providers. OpenFlow protocol is more applicable for the data centers or backbone networks where flows can be effectively used to handle the growing data as in Google and NEC deployed networks. Furthermore, OpenFlow protocol can also be deployed in testbeds e.g., in smaller networks like campus area networks to isolate the traffic such as research traffic and experimental protocols test traffic, and such isolation has been tested with LANs.

5.2. Future work

OpenFlow protocol has been implemented in a few vendors' hardware; most of them have included v1.0, lacking some features of v1.0 specification as well. OpenFlow v1.0 on HP switches has been tested in traditional LAN environment with VLAN. In future, the other interesting protocols to check integration with OpenFlow include MPLS, BGP, OSPF and some secure protocols like OpenVPN, SSL etc. OpenFlow v1.0 does not include support for MPLS headers and IPv6 support as well. Therefore in future with more mature features of OpenFlow available in hardware, it needs to be explored more regarding integration with other protocols in LAN and other networks.

With IPv6 support for OpenFlow v1.2 and higher version implemented in hardware or in software, OpenFlow protocol must be checked for extensive routing protocols such as BGP where routing logic reside inside the router. With OpenFlow v1.0, it is evident that OpenFlow flows cannot replace router functionality, but it will be of interest to verify how OpenFlow protocol affects the routing logics and whether flows can be adaptive and reactive enough to make smart decision yielding improved network performance.

REFERENCES

- [1]. *The Control Plane, Data Plane and Forwarding Plane in Networks*. [www]. [Cited 11.11.12]. Available at: <http://networkstatic.net/the-control-plane-data-plane-and-forwarding-plane-in-networks/>
- [2]. *Knowledge Base*. [www]. [Cited 6.5.13]. Available at: <http://kb.iu.edu/data/agki.html>
- [3]. James F. Kurose, K. W. (2012). *Computer Networking: A Top-Down Approach* (6th ed.). Pearson.
- [4]. *Software-Defined Networking: The New Norm for Networks*. [www]. [Cited 4.3.13]. Available at: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>
- [5]. Kassner, M. (2013, April 8). *Software-Defined Networking: How it affects network security*. [www]. [Cited 10.4.13]. Available at: <http://www.techrepublic.com/blog/security/software-defined-networking-how-it-affects-network-security/9300>
- [6]. Ferro, G. *OpenFlow and Software Defined Networking ipspace webinar*. [www]. [Cited 13.12.12]. Available at: <http://demo.ipspace.net/get/OpenFlow>
- [7]. Goth, G. (2011). Software-Defined Networking Could Shake Up More than Packets. *Internet Computing, IEEE*, 15 (4), pp. 6-9.
- [8]. Pettey, C. (2013, March 25). *Software Defined Networking Creates a New Approach to Delivering Business Agility*. [www]. [Cited 2.4.13]. Available at: <http://www.gartner.com/newsroom/id/2386215>
- [9]. *Software Defined Networks*. [www]. [Cited 15.1.13]. Available at: <http://www.intunenetworks.com/res/?s=U0ROIGluIHRoZSBNZXRybyAtIEV4ZW1dG1Z2ZSBvdmVydmlld18xLnBkZnx8ZG9jdW11bnR8U0ROIEV4ZW1dG1Z2ZSBPdmVydmlldw%3D%3D>
- [10]. *Inter-Datacenter WAN with centralized TE using SDN and OpenFlow*. (2012). [www]. [Cited 19.5.13]. Available at: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/customer-case-studies/cs-googlesdn.pdf>
- [11]. *Kanazawa University Hospital: Case Studies | NEC*. (2012, June 28). [www]. [Cited 6.6.13]. Available at: <http://www.nec.com/en/case/kuh/index.html>
- [12]. *NEC Internal Data Center: Case Studies | NEC*. (2012, March 29). [www]. [Cited 6.6.13]. Available at: <http://www.nec.com/en/case/idc/>

- [13]. *Nippon Express Co., Ltd : Case Studies / NEC.* (29, March 2012). [www]. [Cited 6.6.13]. Available at: <http://www.nec.com/en/case/nittsu/>
- [14]. Soheil Hassas Yeganeh, A. T. (2013). On scalability of software-defined networking. *IEEE Communications Magazine* , 51 (2), 136-141.
- [15]. Amin Tootoonchian, Y. G. (2010). HyperFlow: a distributed control plane for OpenFlow. *INM/WREN'10 Proceedings of the 2010 internet network management conference on Research on enterprise networking* (p. 3). Berkeley, CA, USA: USENIX Association Berkeley, CA, USA.
- [16]. Soheil Hassas Yeganeh, Y. G. (2012). Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. *SIGCOMM 2012*. Helsinki, Finland.
- [17]. Rob Sherwood, G. G. *FlowVisor: A Network Virtualization Layer.* [www]. [Cited 8.2.13]. Available at: <http://www.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf>
- [18]. Hyojoon Kim, N. F. (2013). Improving network management with software defined networking. *IEEE Communications Magazine* , 51 (2), 114-119.
- [19]. Heller, B. (2008, 12). [www]. [Cited 10.8.12]. Available at: <http://www.openflow.org/wp/documents/>
- [20]. Georgia Kontesidou, K. Z. (2009). M.Sc. Thesis: *Openflow Virtual Networking: A Flow-Based Network Virtualization Architecture*. Stockholm, Sweden: Royal Institute of Technology (KTH).
- [21]. Proch, D. (2011). *Using OpenFlow protocol to control network flow*. EE Times-Asia. [www]. [Cited 21.1.13]. Available at: http://www.eetasia.com/STATIC/PDF/201202/EEOL_2012FEB17_NET_TA_01.pdf?SOURCES=DOWNLOAD
- [22]. L. Yang, R. D. (2004). *Forwarding and Control Element Separation (ForCES) Framework*. RFC 3746. IETF.
- [23]. T.V. Lakshman, T. N. (2004). The SoftRouter Architecture. *SIGCOMM Hot Topics in Networks III (HotNets III) workshop*. San Diego, CA.
- [24]. (2012, April). *Past Conferences - SDN Tutorial for Engineers*. [www]. [Cited 21.2.13]. Available at: <http://www.opennetsummit.org/archives/apr12/heller-mon-intro.pdf>
- [25]. (2009). *OpenFlow Switch Specification v1.0.0*. [www]. [Cited 17.2.13]. Available at: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf>

- [26]. (2012). *OpenFlow Switch Specification v1.3.1*. [www]. [Cited 19.2.13]. Available at: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf>
- [27]. *List of OpenFlow Software Projects*. [www]. [Cited 11.3.13]. Available at: <http://yuba.stanford.edu/~casado/of-sw.html>
- [28]. Wireshark Dissector for OpenFlow. [www]. [Cited 24.2.13]. Available at: https://github.com/noxrepo/openflow/tree/master/utilities/wireshark_dissectors
- [29]. *Overview of functionality and components*. [www]. [Cited 11.10.12]. Available at: http://openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=README;hb=HEAD
- [30]. Zdravko Bozakov, V. S. (2013). OpenFlow: A Perspective for Building Versatile Networks. In R. W. Alexander Clemm, *Network-Embedded Management and Applications* (pp. 217-245). SpringerLink.
- [31]. Fernando Farias, I. C. (2011, November 7). *LegacyFlow: Bringing OpenFlow to Legacy Network Environments*. [www]. [Cited 14.1.13]. Available at: http://changeofelia.info.ucl.ac.be/pmwiki/uploads/SummerSchool/Program/poster_004.pdf
- [32]. Fernando N. N. Farias, J. J. (2012). A proposal management of the legacy network environment using OpenFlow control plane. *Network Operations and Management Symposium (NOMS)* (pp. 1143-1150). IEEE.
- [33]. *RouteFlow*. [www]. [Cited 28.2.13]. Available at: <https://sites.google.com/site/routeflow/home>
- [34]. *Quagga Routing Suite*. (2013, March 9). [www]. [Cited 5.5.13]. Available at: <http://www.nongnu.org/quagga/>
- [35]. James Kempf, S. W. (2011). OpenFlow MPLS and the open source label switched router. *23rd International Teletraffic Congress (ITC)* (pp. 8-14). IEEE.
- [36]. *Pica8 Products*. [www]. [Cited 15.10.12]. Available at: <http://www.pica8.org/products/>
- [37]. *GSM7328S-200*. [www]. [Cited 17.10.12]. Available at: <http://www.netgear.fi/business/products/switches/fully-managed-switches/next-gen-edge/GSM7328S-200.aspx#two>
- [38]. *OpenFlow - GENI*. [www]. [Cited 19.11.12]. Available at: <http://groups.geni.net/geni/wiki/OpenFlow>
- [39]. *NDDI and Open Science, Scholarship and Services Exchange (OS3E)*. [www]. [Cited 24.11.12]. Available at: <http://www.internet2.edu/network/ose/>

- [40]. *OFELIA*. [www]. [Cited 29.10.12]. Available at: <http://www.fp7-ofelia.eu/>
- [41]. Hoelzle, U. (2012, April). *Past Conferences - OpenFlow @ Google*. [www]. [Cited 26.10.12]. Available at: <http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf>
- [42]. (2012). *HP OpenFlow Switches*. HP. [www]. [Cited 14.2.13]. Available at: <http://bizsupport2.austin.hp.com/bc/docs/support/SupportManual/c03512348/c03512348.pdf>
- [43]. *cURL*. [www]. [Cited 13.3.13]. Available at: <http://curl.haxx.se/>
- [44]. *Avior*. [www]. [Cited 13.3.13]. Available at: <http://openflow.marist.edu/avior.html>
- [45]. *Iperf*. [www]. [Cited 13.3.13]. Available at: <http://iperf.sourceforge.net/>
- [46]. *Care and Feeding of Netperf 2.6.X*. [www]. [Cited 13.3.13]. Available at: <http://www.netperf.org/svn/netperf2/tags/netperf-2.6.0/doc/netperf.html>
- [47]. *Floodlight OpenFlow Controller*. [www]. [Cited 15.3.13]. Available at: <http://www.projectfloodlight.org/floodlight/>
- [48]. *Floodlight Controller Documentation*. [www]. [Cited 17.3.13]. Available at: <http://docs.projectfloodlight.org/display/floodlightcontroller/The+Controller>
- [49]. *Beacon*. [www]. [Cited 9.5.13]. Available at: <https://openflow.stanford.edu/display/Beacon/Home>
- [50]. *FlowVisor*. [www]. [Cited 13.3.13]. Available at: <https://github.com/OPENNETWORKINGLAB/flowvisor>
- [51]. *RouteFlow mailing list*. [www]. [Cited 18.3.13]. Available at: <https://groups.google.com/forum/?fromgroups=#!forum/routeflow-discuss>
- [52]. *HP OpenFlow FAQs*. [www]. [Cited 13.10.12]. Available at: http://h17007.www1.hp.com/docs/openflow/openflow_faq.pdf
- [53]. *HP Switch Software – OpenFlow Supplement*. [www]. [Cited 18.10.12]. Available at: <http://h20000.www2.hp.com/bc/docs/support/SupportManual/c03170243/c03170243.pdf>
- [54]. *OpenFlow - Software Defined Networking (SDN) - HP Networking*. [www]. [Cited 21.10.12]. Available at: <http://h17007.www1.hp.com/us/en/solutions/technology/openflow/index.aspx>
- [55]. *HP Virtual Application Networks - HP Networking*. [www]. [Cited 24.10.12]. Available at: <http://h17007.www1.hp.com/us/en/solutions/technology/van/index.aspx>

- [56]. *Software Defined Networking*. [www]. [Cited 28.10.12]. Available at: <http://www.brocade.com/solutions-technology/technology/software-defined-networking/openflow.page>
- [57]. *OpenFlow*. [www]. [Cited 4.11.12]. Available at: <https://developer.juniper.net/content/jdn/en/learn/technologies/openflow.html>
- [58]. *Cisco Open Network Environment*. [www]. [Cited 5.11.12]. Available at: http://www.cisco.com/web/solutions/trends/open_network_environment/index.html
- [59]. *NEC Programmable Flow Networking*. [www]. [Cited 6.11.12]. Available at: <http://www.necam.com/pflow/>
- [60]. *SDN (Software-Defined Networking) | NEC R&D*. [www]. [Cited 6.6.13]. Available at: <http://www.nec.com/en/global/rd/research/cl/sdn/>
- [61]. Jon Oltsik, B. L. (2012, January). *IBM and NEC Bring SDN/OpenFlow to Enterprise Data Center Networks*. [www]. [Cited 6.6.13]. Available at: http://www.nec.com/en/global/prod/pflow/images_documents/ESG_Brief_IBM_and_NEC_Bring_SDN_OpenFlow_to_Enterprise_Data_Center_Networks.pdf
- [62]. Gude, N. a. (2008). NOX: Towards an Operating System for Networks. *SIGCOMM Comput. Commun. Rev.* , 38 (3), 105-110
- [63]. *Trema*. [www]. [Cited 6.5.13]. Available at: <https://github.com/trema/trema>
- [64]. Dietz, T. *Trema Tutorial*. [www]. [Cited 6.5.13]. Available at: <http://www.fp7-ofelia.eu/assets/Uploads/201203xx-TremaTutorial.pdf>
- [65]. *NOX Classic Wiki*. [www]. [Cited 5.5.13]. Available at: <https://github.com/noxrepo/nox-classic/wiki>
- [66]. *POX Wiki*. [www]. [Cited 5.5.13]. Available at: <https://openflow.stanford.edu/display/ONL/POX+Wiki>
- [67]. Muntaner, G. R. (2012). M.Sc. Thesis: *Evaluation of OpenFlow Controllers*. Stockholm, Sweden: Royal Institute of Technology (KTH).

APPENDIX A: COMPARISON BETWEEN SWITCHES

This appendix compares switches from different vendors including HP, NEC, Brocade, Cisco and Juniper Networks in Table A.1 [52][53][54][55][56][57][58][59][60][61].

Table A.1. Comparison between switches

| Features | HP | NEC | Brocade | Juniper | Cisco |
|------------------------------------|--|-------------------------|---|---|--|
| Joined OpenFlow development | 2007 | 2006 | 2010 | | |
| Released first commercial hardware | 2008 | 2011 | 2010 | 2011 | 2012 |
| OpenFlow version support | v 1.0 | v 1.0 | v 1.0 | v 1.0 | v 1.0 |
| Products | series:3500,3500yl, 3800,5400zl, 8200zl. Also released firmware update for 6200yl and 6600 series. | PF5240, PF5820 switches | MLX series, NetIron XMR series, NetIron CER 2000 series | OpenFlow switch application (OF-APP) - an SDK compatible with MX series routers and others. | Cisco Open Networking Environment (Cisco + OpenFlow + OpenStack) onePK - an SDK to be released soon. Catalyst 3750 and 3560 switches to be supported soon. |

APPENDIX B: COMPARISON BETWEEN CONTROLLERS

This appendix compares popular OpenFlow controllers including NOX, POX, Floodlight and Trema in Table B.1 [47][48][62][63][64][65][66][67].

Table B.1. Comparison between controllers

| Features | NOX | POX | Floodlight | Trema |
|---------------------------|--|---|--|---|
| Open source | Yes | Yes | Yes | Yes |
| Programming language | C++ | Python | Java | Ruby and C |
| Released first version | 2008 | 2011 | Jan 2012 | 2011 |
| Supports APIs | Yes, but harder restrictions with libboost library. | Yes, but harder restrictions with libboost library. | Yes | Yes |
| Active development | No | Yes | Yes | Yes |
| Modular design | Yes | Yes | Yes | Yes |
| web interface | Yes | Yes | Yes | No |
| Proactive flows | No | No | Yes | No |
| Flow entries | via functions such as self.send_openflow() function for installing flows | via functions such as connection.send() function for installing flows | via Static Flow entry pusher API, external utilities curl and Avior available. | via functions such as send_flow_mod_add() function for installing flows |
| Benchmarking - throughput | With 32 million hosts, it yielded 57500 responses/sec to queries. | With 32 million hosts, it yielded 57500 responses/sec to queries. | With 30 million hosts, it yielded 64000 responses/sec to queries. | With 32 million hosts, it yielded 21400 responses/sec to queries. |
| Benchmarking - latency | With 32 million hosts, a latency of 60 msec observed. | With 32 million hosts, a latency of 60 msec observed. | With 30 million hosts, a latency of 11 msec observed. | With 30 million hosts, a latency of 3,79 msec observed. |

APPENDIX C: OPENFLOW PROTOCOL IN WIRESHARK

This appendix enlists the OpenFlow protocol messages available for capture in Wireshark using OpenFlow Wireshark dissector plugin. In the message type column, A refers to controller-to-switch messages, B refers to asynchronous messages and C refers to symmetric messages.

Table C.1. Wireshark messages for OpenFlow protocol

| Message type | Wireshark Message type (of.type) | Protocol | Message purpose |
|--------------|----------------------------------|----------|--|
| C | 2 | OFP | Echo Request |
| C | 3 | OFP | Echo Reply |
| A | 5 | OFP | Features Request |
| A | 6 | OFP | Features Reply |
| B | 7 | OFP | Stats Request |
| A | 7 | OFP | Get Config Request |
| A | 8 | OFP | Get Config Reply |
| A | 9 | OFP | Set Config |
| B | 10 | OFP+LLDP | Packet-In LLDP |
| B | 10 | OFP+0x89 | Packet-In Ethernet II |
| B | 10 | OFP+ARP | Packet-In ARP |
| B | 10 | OFP+MDNS | Packet-In DNS standard query for pointer (PTR) |
| B | 13 | OFP | Packet-Out |
| B | 13 | OFP+LLDP | Packet-Out LLDP |
| B | 13 | OFP+0x89 | Packet-Out Ethernet II |
| A | 14 | OFP | Flow Modification |
| B | 16 | OFP | Stats Request |
| B | 17 | OFP | Stats Reply |

APPENDIX D: FEATURES OF OPENFLOW IN HP

This appendix enlists about features of OpenFlow protocol v1.0 which have been implemented in HP switches. It further mentions some features which are not interoperable with OpenFlow protocol in HP switches.

D.1 Features available for OpenFlow

This section lists features of HP switches that are in line with OpenFlow v1.0 protocol specifications [42].

- Limiting percentage of policy engine resources
- Limiting amount of traffic
- Supports hardware flows
- Supports active and passive mode for flows.

D.2 Features not available for OpenFlow

This section lists the features of OpenFlow v1.0 protocol specifications that are not available in HP switches [42].

- Encrypted connection from controller to switch using TLS
- The `enqueue` action that forwards a packet through a queue attached to a port
- Handling of IP fragments
- Stripping VLAN header
- `TABLE` action that performs actions in a flow table and used only for packet-out messages
- `IN_PORT` action that sends the packet out to the same incoming port.
- Some port commands not supported such as:
 - `Disable (OFPPC_PORT_DOWN)`
 - `Disable 802.11D Spanning Tree Protocol (STP) on port (OFPPC_NO_STP)`
 - `Drop all packets excluding 802.11D STP packets on port (OFPPC_NO_RECV)`
 - `Drop all packets on a port (OFPPC_NO_RECV_STP)`
 - `Drop all packets forwarded to a port (OFPPC_NO_FWD)` .

D.3 Features not interoperable with OpenFlow

This section enlists some features of HP switch that cannot be used when the switch is configured in OpenFlow mode [42].

- Q-in-Q mode, i.e., 802.11Q tunnelling
- Meshing
- Transparent Router (TR) mode of operation, usually used with Storage Area Networks (SANs).