Addis Ababa University
College of Natural Science

Analysis of OpenFlow-Hybrid mode Network Architecture

Seblewongale Zerihun

A Thesis Submitted to the Department of Computer Science in Partial Fulfillment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia
June 2018

Addis Ababa University
College of Natural Science

Seblewongale Zerihun

Advisor: Dr. Mulugeta Libsie

This is to certify that the thesis prepared by Seblewongale Zerihun, titled*: Analysis of OpenFlow-Hybrid Mode Network Architecture* and submitted in partial fulfilment of the requirements for the Degree of Master of Science in Computer Science compiles with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

Name	Signature	Date

Advisor:_____

Examiner:_____

Examiner:_____

Abstract

Lack of flexibility and scalability on traditional networks require more studies and researches in the area and this leads to the introduction of Software Defined Networking (SDN). The main objective of Software Defined Networking is to separate the control plane from the data plane and introduce the concept of Programmable Network. OpenFlow is a protocol used in Software Defined Networking architecture as a communication channel between the data plane and the control plane. Different applications can easily be integrated with the control plane in order to avoid vendor specificity issues with greater speed and greater scalability in terms of routing and forwarding. Completely shifting from traditional networks to this new architecture may not be an easy task and it is not also the target of Software Defined Networking, instead it encourages to shift step by step. The OpenFlow specification introduces two forms of implementations: pure and hybrid OpenFlow implementations. The hybrid implementation supports both traditional packets forwarding as well as OpenFlow network architecture.

In this work, we present a performance analysis of the hybrid OpenFlow architecture in terms of throughput, end-to-end delay, and scalability on Local Area Network (LAN) side by side with the pure OpenFlow implementation. The aim is to explore how OpenFlow hybrid network can be implemented on the simulation environment and to see what enhancements it can bring to the OpenFlow network.

Keywords: Software Defined Networking, OpenFlow, Hybrid Implementation, HP VAN SDN controller, Open Virtual Switch.

Acknowledgements

First and foremost, I would like to thank God for being with me in all the things I did in my life. Next, I want to give my honor to my advisor Dr. Mulugeta Libsie for giving me his time when I needed him and for his valuable comments.

I also appreciate persons who shared their knowledge on the Internet for teaching and guiding us on the things we would like to do

Table of Contents

List of Figures

List of Tables

Acronyms and Abbreviations

API:	Application Interface
DoS:	Denial of Service
I/O:	Input/Output
IPV6:	Internet Protocol Version 6
HP:	Hewlett-Packard
MPLS:	Multi Protocol Label Switching
MAC:	Media Access Control
ONF:	Open Networking Foundation
OVS:	Open vSwitch
PTR:	Packet Transmission Rate
QoS:	Quality of Service
RTT:	Round Trip Time
SDN:	Software Defined Networking
SSL:	Secure Socket Layer
TLS:	Transport Layer Security
VAN:	Virtual Application Network
VLAN:	Virtual Local Area Network
vSwitch:	virtual Switch

Chapter One: Introduction

1.1 Background

Networking is a backbone for organizations to provide services in a manageable way as it creates accessibility to external users and customers. Network performance and infrastructure expansion have been the main focus area of studies. Virtual Local Area Network (VLAN) was one of the technologies introduced to enhance traditional network infrastructure expansion by separating a single network into multiple isolated networks. Even though it improves network scalability at some level, it still has difficulties in flexibility.

Software-defined networking (SDN) is an emerging networking paradigm that gives hope to change the limitations of current network infrastructures. It breaks the vertical integration by separating the network's control logic (the control plane) from the underlying routers and switches that forward the traffic (the data plane) [1]. With the separation of the control and data planes, network switches become simple forwarding devices and the control logic is implemented in a logically centralized controller. This simplifies network reconfiguration by introducing a more flexible architecture. It is an approach to building computer networks that separates and abstracts elements of these systems [2].

Furthermore, SDN eliminates vendor specificity and compatibility issues by having a central control device and interfacing API.

The SDN architecture and the OpenFlow standard provide an open architecture in which control functions are separated from the network device and placed in accessible control server. This setup enables the underlying infrastructure to be abstracted for applications and network services, enabling the network to be treated as a logical entity. OpenFlow is an application layer protocol used by OpenFlow network to communicate with the controller. To turn the concept of SDN into practical implementation, two requirements must be met [3]. First, there must be a common logical architecture in all switches, routers, and other network devices to be managed by an SDN controller. This logical architecture may be implemented in various ways on different vendor equipment and in different types of network devices, so long as the SDN controller sees a uniform logical switch function. Second, a standard secure protocol is needed between the SDN controller and the network device. In SDN, there is a

concept called centralized or distributed controller architecture, which is based on the location and setup of the OpenFlow controller architecture.

The control plane dictates all aspects of network operation on the data plane in SDN, including forwarding, access, policies, and quality of service. Moreover, it is the responsibility of the data plane to implement and enforce decisions made by the controller. The OpenFlow protocol is typically used in one of the following two forms. Pure OpenFlow**,** where the controller explicitly makes all forwarding decisions or Hybrid forwarding**,** where the controller delegates some forwarding decisions to controlled switches [4].

1.2 Motivation

Because SDN brings a great flexibility for networking industries, it becomes a very interesting concept for people who spent more of their times in the area. The benefits that it adds to traditional networks initiate for further studying and understanding this architecture. Since it is still a hot issue in networking, lots of articles and journal papers are being written. OpenFlow protocol is one of the core components that SDN included in the architecture in order to provide a more flexible communication in networking [5].Applying the OpenFlow network to the existing infrastructure may require a total shift by replacing all network switches and routers with new devices that support the OpenFlow protocol. This is one of the challenges to the adoption of SDN; it adds additional budgets to organizations. Hybrid OpenFlow network is a solution provided by SDN for the above problem. It allows both OpenFlow traffics and traditional network traffics to communicate with in a single LAN. Thus, the underline motive of the work is to simulate and make analysis to the hybrid OpenFlow implementation.

1.3 Statement of the Problem

New software and hardware products are developed by different vendors. However, it is no secret that applying these new products to the business environment is not an easy task. Since OpenFlow of SDN will become a feature that most networking hardware vendors included in their design, lots of tests and analytical researches should be performed in order to identify the positive and negative effects it has. Introducing it to an existing infrastructure without simulating it first may create chaos to network administrators as well as company owners.

OpenFlow-Hybrid mode architecture is introduced to integrate traditional networking concepts with the SDN architecture. The problem here is knowing what effects it will have on the performance and scalability of the LAN.

1.4 Objectives

General Objective

The general objective of this thesis is to make performance and scalability analysis of OpenFlow-hybrid mode architecture and discuss about the advantages and the drawbacks it will have on the OpenFlow network by comparing it with the pure OpenFlow network.

Specific Objectives

To achieve the general objective, the following specific objectives are identified:

- Configure the controller to support both types of implementations,
- Capture traffics exchanged between network devices and the controller,
- Perform traffic analysis in terms of performance based on throughput and bandwidth,
- Analyze network delay between nodes in the LAN, and
- Measure scalability of the network based on flow table content, and
- Documenting the analysis result.

1.5 Methods

The simulation will be implemented on a virtual machine by installing Ubuntu Linux OS on it. Following this, the simulation software; in our case Mininet and Open vSwitch, will be installed on the virtual machine and requires some setup to make it ready for the design. The LAN environment that includes client computers, OpenFlow switch, and the controller will be designed based on the preliminary topology using the simulation software. Succeeding this, connectivity between devices will be activated and tested. The OpenFlow controller is configured to support the hybrid-forwarding mode using its web interface.

For the data analysis purpose, the packet capturing software will be installed on our machine and it will start capturing the traffics when packets are exchanged between devices in the LAN. This will be followed by the analysis of sent and received packets of both the OpenFlow

and traditional Layer 2 or Layer 3 networks by considering different scenarios and analysis tools.

At the conclusion of the thesis, the simulation will be able to serve as experimental output of the architecture.

1.6 Scope and Limitations

This thesis is limited to simulate OpenFlow network that will be used for analyzing OpenFlow-Hybrid mode architecture. The experiment will only be tested with forwarding packets that are exchanged between hosts in a LAN. In addition to that a single controller is used in the simulation environment for performing forwarding decisions on the sent and received packets. Multiple controllers can be used for load balancing the network traffics as well as avoiding a single fail over. Moreover, the Mininet simulation software that is going to be used has also the capacity to be used for emulation. An emulator builds a testing network field consisting of physical and virtual network devices [6]. This work is bound to use only the simulation version of it.

1.7 Application of Results

Simulating this new network architecture by considering different scenarios using simulation software has a great significance for organizations that are thinking to shift from traditional networks to this new architecture with a clear understanding. Performing technical analysis on this architecture will also provide a chance to identify which parts of this product are technically good and which parts require attention for further investigation. Furthermore the findings may open a door for the research community.

1.8 Organization of the Rest of the Thesis

The remaining part of this document is organized in six chapters. Literature review is dealt with in Chapter two. Works related with ours are presented in the third Chapter. Chapter four presents the experimental environment to analyze the architecture together with result summaries and findings we get from the experiment. Open vSwitch implementation will be discussed in Chapter five and the final Chapter concludes the whole work.

Chapter 2: Literature Review

The goal of this Chapter is to provide insight on the purpose of a new network paradigm, Software Defined Networking. The chapter starts by explaining why SDN is needed and what inherited problems it should overcome. Secondly, an explanation of the ONF basics is provided. The chapter also discusses about OpenFlow protocol architecture and OpenFlow controller. Finally an overview of OpenFlow Hybrid Forwarding and Open vSwitch are discussed.

2.1 Software Defined Networking (SDN)

Software-Defined Networking (SDN) was suggested in 2005 to overcome conventional network architecture limitations. It is expected that SDN will enhance programmability and makes the network easily manageable by decoupling the control plane from the data plane. SDN is an architectural approach that optimizes and simplifies network operations by more closely binding the interaction (i.e., provisioning, messaging, and alarming) among applications and network services and devices, whether they are real or virtualized [1].

SDN is a general term encompassing several kinds of network technology aimed at making the network as agile and flexible as the virtualized server and storage infrastructure of the modern data center. The goal of SDN is to allow network engineers and administrators to respond quickly to changing business requirements. In a SDN, a network administrator can shape traffic from a centralized control console without having to touch individual switches, and can deliver services to wherever they are needed in the network, without regard to what specific devices a server or other device is connected to. The key technologies are functional separation, network virtualization and automation through programmability. Introducing these networking features allow to achieve the following three main advantages [7]:

- Interoperability: Centralized control over the SDN enables devices from any vendor throughout the whole network.
- Simplicity: the complexity issues are eliminated. The network control is easier and more fine grained, which results in an increased reliability and security, in a more granular network control. Flow based control allows control over each session, user, device, and application.

- Innovativeness: The abstraction of the network services from the network infrastructure allows network operators to tailor the behavior of the network and program new services faster than before. The entire structure becomes much more evolvable.

Adopting this new architecture is not going very fast since it requires modification on network hardware devices for supporting the OpenFlow protocol. The limitations that the OpenFlow protocol had on supporting protocols like IPv6 and MPLS was also another reason but performing an intensive research on the area leads to invent more enhanced version that has the ability to support and integrate with traditional services. According to CISCO, a research survey found that 45% of respondents plan to have SDN live in production data centers in 2015, growing to 87% in 2016 [8].

2.1.1 Open Networking Foundation (ONF)

Open Networking Foundation (ONF) is a user-driven organization dedicated to the promotion and adoption of SDN, and implementing SDN through open standards where such standards are necessary to move the networking industry forward [9]. As part of its quest to make SDN a commercial reality that meets customer needs, ONF is developing open standards such as the OpenFlow Standard and the OpenFlow Configuration and Management Protocol Standard. The OpenFlow Standard is the first and only vendor-neutral standard communications interface defined between the control and forwarding layers of an SDN architecture. ONF working groups are also paving the way for interoperable solution development by collaborating with the world's leading experts on SDN and OpenFlow regarding SDN concepts, frameworks, architecture, and standards.

2.1.2 SDN Architecture

Traditional network architectures are less suited to meet the requirements of today's enterprises, carriers, and end users. Thanks to a broad industry effort spearheaded by the ONF, SDN is transforming networking architecture. As shown in Figure 2.1 [4, 6], the architecture has three major parts. The Infrastructure Layer contains the OpenFlow network where all the network devices reside on. The Control Layer is where the control and management of the OpenFlow network is performed and it communicates with the infrastructure layer using the OpenFlow protocol. The Application Layer is the interface that allows integration of different applications to the network.

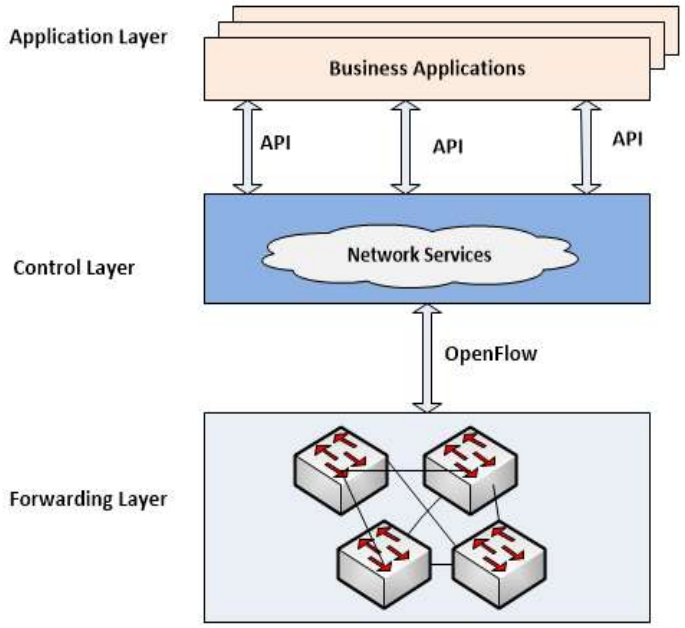

Figure 2.1: *SDN Architecture*

2.2 OpenFlow

OpenFlow is the first standard communications interface defined between the control and forwarding layers of an SDN architecture [10]. OpenFlow allows direct access to and manipulation of the forwarding plane of network devices such as switches and routers, both physical and virtual (hypervisor-based).

The OpenFlow protocol is implemented on both sides of the interface between network infrastructure devices and the SDN control software. OpenFlow uses the concept of flows to identify network traffic based on pre-defined match rules that can be statically or dynamically programmed by the SDN control software. It also allows system administrators to define how traffic should flow through network devices based on parameters such as usage patterns, applications, and cloud resources [4]. The OpenFlow protocol is a key enabler for Software Defined Networks and currently is the only standardized SDN protocol that allows direct manipulation of the forwarding plane of network devices.

2.2.1 OpenFlow Architecture

The OpenFlow network architecture consists of three basic concepts:

- ✓ OpenFlow-compliant switches that compose the data plane.

- ✓ The control plane consists of one or more OpenFlow controllers.
- ✓ A secure control channel connects the switches with the control plane.

The Switches communicate with the hosts and with each other using the data path, and the controller communicates with the switches using the control path as shown in Figure 2.2.

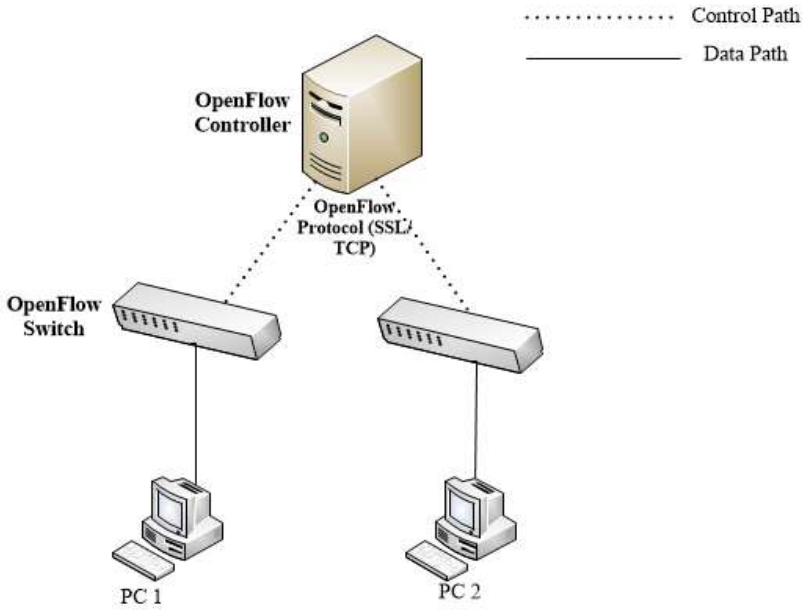

Figure 2.2: *OpenFlow Network Architecture*

The connection between the OpenFlow controller and the switch is secured using SSL or TLS cryptographic protocols, where the switch and the controller are mutually authenticated by exchanging certificates signed by both sides' private key. Although this is a very powerful security algorithm, the controller may be vulnerable to Denial of Service (DoS) attack, or man in the middle attack. Therefore, appropriate security practices must be implemented to prevent such attacks [11].

2.2.2 OpenFlow Tables

There are three types of flow tables in OpenFlow network and each table has different functionality as discussed below.

a, Flow Table

A switch in OpenFlow network has one (or more) flow table(s) that contains a set of entries, each of which consists of fields named match, action, and counters. All packets processed by

the switch are compared against the flow table. If a packet header matches a flow entry, an action for that entry is performed on the packet (e.g., the action might be to forward a packet out to a specified port). If no match is found, the packet is forwarded to the controller over the secure channel. The counters are reserved for collecting statistics about flows. They store the number of received packets and bytes, as well as the duration of the flow. The flow entry that wildcards all fields (all fields omitted) and has priority equal to 0 is called the table-miss flow entry. Possible OpenFlow actions taken on a flow are listed in Table 2.1 [12].

Table 2.1: *List of Possible OpenFlow Actions*

Action Type	***Description***	
Forwarding	ALL:	Forward the packet to all ports except the received port.
	CONTROLLER:	Encapsulate the packet and send it to the controller.
	LOCAL:	Send the packet to the local networking stack for the switch.
	TABLE:	Perform actions described in the flow table. Only for packet-out messages.
	IN_PORT:	Send the packet to the received port.
Drop	A required action indicated by an empty action list. All packets that match an empty action list are dropped.	
Enqueue	This optional action can be used to put packets in a queue, which is associated with a port in order to provide Quality of service.	
Modify-Field	This optional action is employed to modify a specific header field for the incoming packet.	

b, Group Table

A group table is the same as a flow table; the only difference is that a flow entry points to a group. It has also major components like Group Identifier, Group Type, Counters, and Action Buckets. There are four group types used by the switch in a flow table and the switch should only support the required one. Table 2.2 [12] describes the detail as follows.

Table 2.2: *Group Table Types*

Group Type	***Mark***	***Description***
All	Required	Used for multicast or broadcast forwarding and all packets in a bucket are executed.
Select	Optional	Execute one bucket in a group.
Indirect	Required	Execute the one defined bucket in this group. This group only supported a single bucket.
Fast failover	Optional	Execute the first live bucket. This group type enables the switch to change forwarding without requiring a round trip to the controller.

c, Meter Table

A meter table consists of meter entries that are used to identify some patterns on the sent packets. It enables OpenFlow to implement various QoS operations. It has main components such as: Meter Identifier, Meter Bands, and Counters.

2.2.3 Flow Table Messages

There are three classes of communication in the OpenFlow protocol:

- ✓ **Controller-to-switch:** Controller-to-switch messages are established by the controller and sent to the switch.
- ✓ **Asynchronous:** These messages are initiated by the switch and sent to the controller.
- ✓ **Symmetric:** Symmetric messages are initialized by both the switch and the controller in either direction.

2.2.4 Flow Table Pipeline

An OpenFlow switch includes one or more flow tables. If there is more than one flow table, they are organized as a pipeline as shown in Figure 2.3 [10].

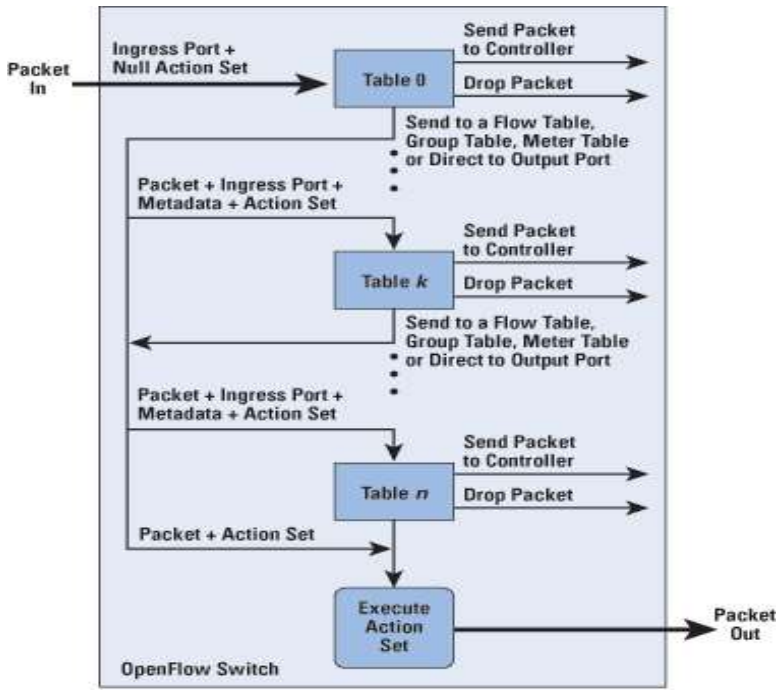

Figure 2.3: *Flow Table Pipeline*

When a packet is presented to a table for matching, the input consists of the packet, the identity of the ingress port, the associated metadata value, and the associated action set. For Table 0, the metadata value is blank and the action set is null. Processing proceeds as follows [9, 10]:

1. Find the highest-priority matching flow entry. If there is no match on any entry and there is no table-miss entry, then the packet is dropped. If there is a match only on a table-miss entry, then that entry specifies one of three actions:
   - Send packet to controller. This action will enable the controller to define a new flow for this and similar packets, or decide to drop the packet.
   - Direct packet to another flow table farther down the pipeline.
   - Drop the packet.
2. If there is a match on one or more entries other than the table-miss entry, then the match is defined to be with the highest-priority matching entry. The following actions may then be performed:

- Update any counters associated with this entry.
- Execute any instructions associated with this entry. These instructions may include updating the action set, updating the metadata value, and performing actions.
- The packet is then forwarded to a flow table further down the pipeline, to the group table, or to the meter table, or it could be directed to an output port.

For the final table in the pipeline, forwarding to another flow table is not an option. If and when a packet is finally directed to an output port, the accumulated action set is executed and then the packet is queued for output.

2.3 Proactive vs. Reactive Flow Entry Management

In the SDN architecture, the control plane is responsible for the configuration of the forwarding devices. With OpenFlow, the controller installs flow table entries in the forwarding tables of the switches. An entry consists of match fields, counters and forwarding actions. The OpenFlow match fields are wildcards that match to specific header fields in the packets. Wildcards are typically installed in ternary content-addressable memory (TCAM) to ensure fast packet matching and forwarding. However, TCAM is very expensive, so that it needs to be small; as a consequence, only a moderate number of flow entries can be accommodated in the flow table [13].

The controller is able to install flow entries permanently and in particular before they are actually needed. This approach is referred to as proactive flow management. However, this approach has a disadvantage: flow tables must hold many entries that might not fit into the expensive TCAM. To cope with small flow tables, flow entries can also be installed reactively, *i.e.*, installed on demand.

2.4 SDN Controller

SDN Controller is where the function of a network control plane is performed and it is where network programmability is applied in SDN architecture. The SDN controller serves as a sort of operating system for the network. By taking the control plane off the network hardware and running it as software instead, the controller facilitates automated network management and makes it easier to integrate and administer business applications.

2.4.1 SDN Controller Types

Various network hardware vendors developed different SDN controllers and the major differences areon the programming language each vendor uses for the development.Table2.3[14]lists open source controllers available together with the programming language they are developed.

Table 2.3: *List of SDN Controllers with their Properties*

Name	*Programming Language*	*Description*
NOX	C++	Initially developed at Stanford University.
POX	Python	Forked from the NOX controller. POX is written in Python and runs under various platforms.
Beacon	Java	Initially developed at Stanford.
Floodlight	Java	Forked from the Beacon controller and sponsored by Big Switch Networks.
Maestro	Java	Multi-threaded OpenFlow controller developed at Rice University.
NodeFlow	JavaScript	JavaScript OpenFlow controller based on Node.JS.
Terma	C and Ruby	Plugins can be written in C and in Ruby. Trema is developed by NEC.
Open Daylight	Java	Open Daylight is hosted by the Linux Foundation, but has no restrictions on the operating system.

2.4.2 SDN Controller Performance

The performance of controllers is highly dependent on the programing language they are developed. Researches have made a lot of studies to evaluate which controller is best by considering different scenarios. In OpenFlow network two of the key performance metrics associated with an SDN controller are the flow setup time and the number of flows per second that the controller can setup [15]. The key factors that affect flow setup time include the processing power of the switches that are attached to the controller and processing and I/O

performance of the Controller. The processing and I/O performance of the controller are influenced by a number of factors. For example, a controller whose software is written in C programming language will be faster than the one whose software is written in Java.

2.5 Controller Clustering

Clustering is a mechanism that enables multiple processes and programs to work together as one entity. For example, when we go to google.com and search for something, it may seem like our search request is processed by only one web server. In reality, our search request is processed by thousands of web servers connected in a cluster. Similarly, we can have multiple instances of the controller working together as one entity. Having multiple controller instances in a network will have the following advantages [16]:

Scaling: If we have multiple controllers running, we can potentially do more work with or store more data on those controllers if they are clustered. We can also break up our data into smaller chunks and either distribute that data across the cluster or perform certain operations on certain members of the cluster.

High Availability: If we have multiple controllers running and one of them crashes, we would still have the other instances working and available.

Data Persistence: We will not lose any data gathered by the controller after a manual restart or a crash.

2.6 Hybrid Forwarding

Hybrid forwarding is when the OpenFlow switch supports both OpenFlow enabled flow forwarding and traditional Ethernet switch bridging and routing. The controller may choose to delegate some portion of the data plane forwarding decisions to the controlled switches [3]. With hybrid SDN the controller still retains control over all packets forwarding on the data plane; however it chooses to delegate the forwarding decision to controlled switches for the following reasons:

- ✓ It reduces the complexity and scope of the forwarding decisions that the controller makes.

- ✓ It reduces the amount of traffic on the control plane between the switches and the controller.

Figure 2.4 shows all packet types on a traditional network [17]. In hybrid OpenFlow network the main point is who is responsible for making the forwarding decision for packet types below.

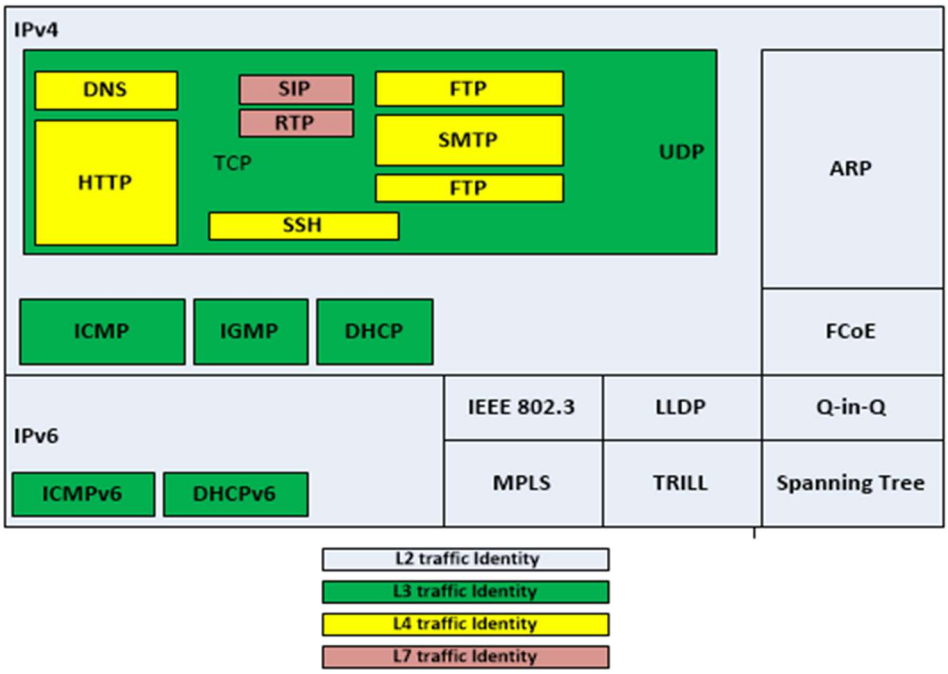

Figure 2.4: *Packet Types in a Traditional Network*

2.7 Open vSwitch

2.7.1 Open vSwitch Architecture

Open vSwitch is an open-source virtual switch software designed for virtual servers. Open vSwitch is designed to be compatible with modern switching chipsets so that it can be ported to high-fanout switches, thus providing the same flexibility of control to physical infrastructure as virtual ones [18]. Open vSwitch is commonly used as an SDN switch, and the main way to control forwarding is OpenFlow. As we can see from Figure 2.5 [19] it has user and kernel modes.

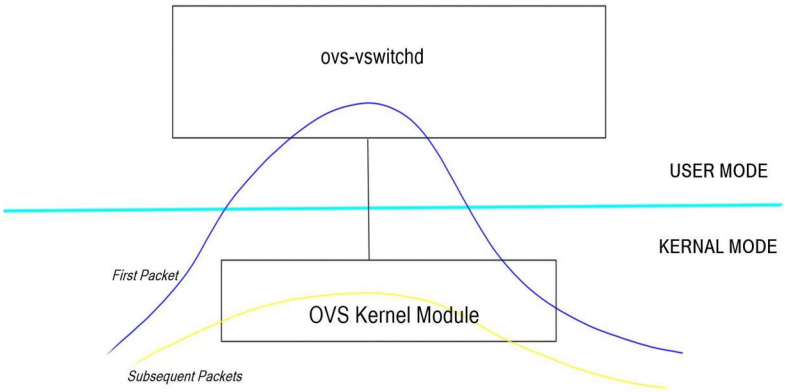

Figure 2.5: *Open vSwitch Modes*

The OVS kernel module that is also called kernel datapath module receives the packets first, from a physical NIC or a VM's virtual NIC and it sends the packet to the ovs-vswitchd of the user mode for instructions on how to handle it. In Open vSwitch, ovs-vswitchd receives OpenFlow flow tables from an SDN controller.

2.7.2 Components of Open vSwitch

Open vSwich has components that provide different functionalities for the OpenvSwitch distribution [20]:

- ovs-vswitchd, a daemon that implements the switch, along with a companion Linux kernel module for flow- based switching.
- ovsdb-server, a lightweight database server that ovs-vswitchd queries to obtain its configuration.
- ovs-dpctl, a tool for configuring the switch kernel module.
- ovs-vsctl, a utility for querying and updating the configuration of ovs-vswitchd.
- ovs-appctl, a utility that sends commands to running Open vSwitch daemons.
- ovs-ofctl, a utility for querying and controlling OpenFlow switches and controllers.
- ovs-pki, a utility for creating and managing the public-key infrastructure.

2.8 Summary

OpenFlow is the first standard communications interface defined between the control and forwarding layers of an SDN architecture and it is a set of specifications maintained by the Open Networking Forum (ONF). OpenFlow allows us to dive deep into the forwarding plane and set rules on how traffic should be forwarded, based on flow rules sent from an SDN controller. This means it was able to bypass the control plane of a switch, as a result creating a more open, simplified switch. In the OpenFlow network the controller plays an important role for forwarding the incoming packets. In general SDN eliminates the drawbacks that exists on traditional networks especially on vendor dependency and flexibility challenges.

Chapter 3: Related Work

This Chapter deals with research works that are published by different researchers to combat the performance analysis of OpenFlow. Even though there are lots of networking areas that OpenFlow is applied to, in this Chapter we point out the most related categories that the performance of OpenFlow protocol is analyzed for the purpose of understandability.

The related works we have reviewed are discussed in the following three broad categories: OpenFlow Protocol Performance in LAN, Performance Analysis of OpenFlow controllers, and Analysis of OpenFlow in Hybrid Implementation.

3.1 OpenFlow Protocol Performance in LAN

The significance of having OpenFlow protocol on campus network becomes a very interesting and advantageous action taken to increase flexibility and scalability on the existing traditional networks. Articles, researches, and thesis study the feasibility of having OpenFlow protocol in the current network infrastructures by considering different scenarios.

The research in [21] discussed the OpenFlow-based network architecture on three topologies (Single, Linear, and Tree). The analysis is done by comparing all the three network topologies on the basis of bandwidth utilization, packet transmission rate, round-trip propagation delay between end nodes, and maximum throughput. The work stated that the performance of single topology is better as compared with the other two with some limitations. In single topology if the number of hosts in the network increases, the performance of the network will automatically get reduced. It is because the entire load is on a single OpenFlow switch, and also the number of flow entries in the switch will be very large. In linear topology, the overall network performance is improved as compared with the single topology on the basis of network load. In linear topology, the network load is distributed on different switches, and each switch is responsible for handling limited flow entries. All switches should have almost equal number of flow entries, which is controlled by the controller. In a Tree topology the performance on the basis of speed consideration is improved as compared to linear topology but limited as compared to single topology. Unlike single topology, in tree topology the entire network load is distributed into the switches, which is not in the case of single topology network.

The research in [22] deals with the initial steps taken to evaluate the performance of

OpenFlow. The authors made an attempt to measure the performance in terms of scalability and adaptability of OpenFlow in the current network. This study was based on measuring switching time of OpenFlow hardware, reflecting the forwarding speed and blocking probability of OpenFlow switch attached to an OpenFlow controller. In addition to that, the authors used three different OpenFlow implementations, a Pronto 3290 24 port gigabit switch, a NetFPGA OpenFlow switch as well as the Open vSwitch software switch together with the OpenFlow Nox 0.9 controller. The authors concluded by explaining the time taken by the controller to respond to the data plane requests with respect to the number of flows received by the switch. This model explains the significance of the performance of controller in installing new flows in the switch with high forwarding speeds.

A work in [23] also performs analysis of OpenFlow in LAN. The author tried to elaborate the OpenFlow architecture by implementing it on a LAN environment. Multiple implementation scenarios were considered for the analysis purpose and also OpenFlow switches provided by different vendors are used. According to the author different Linux tools like Curl, Avior, Iperf, and Netperfare used for bandwidth measurement and flow configuration and administration purpose. Flows are installed on the switches manually using the Curl and Avior tools and the throughput is measured by capturing the traffics after ping request is issued between end nodes. Results are tabulated in three categories as Before Deploying OpenFlow, No Flows in the OpenFlow Network, and Flows in the OpenFlow Network. The paper finally concluded that the network has throughput of 941 Mbits/sec before and after deploying OpenFlow that means, deploying OpenFlow on the network doesn't affect the network throughput. But, when no flows are installed on the switches, throughput result shows different output.

3.2 Performance Analysis of OpenFlow Controllers

In the OpenFlow network the role of the controller is crucial for the entire network communication. Since the controller performs most packet forwarding decisions and flow installation tasks, it is unquestionable to have a light and scalable controller in the network.

The work in [24] performs throughput and latency performance evaluation of three controllers (SNAC Controller, Beacon Controller and Trema Openflow Controller).The test was performed on the same device and uses a Cbench tool for taking a benchmark for the

evaluation. The paper stated that throughput is tested by increasing the number of switches from 6 initial switches to 12 and then to 24. This indicates that requests to the controller are also increase.

The result shows that the Beacon Controller has the capability to receive twice number of requests as compared with the Terma Controller and it shows a better throughput. As mentioned in the paper the result of SNAC controller is not listed because of the difficulties faced to install the controller software. The latency is also evaluated between the Beacon and Terma Controllers and the Terma Controller shows a better result.

According to the authors the latency difference comes because of the programming language that the controller is developed and the number of lines that the code has.

The work in [12] also tries to present a performance evaluation in both throughput and latency perspectives for the current well-known OpenFlow controllers (NOX, Beacon, Floodlight, Maestro, OpenMul, and OpenIRIS). The paper stated that performance evaluation was performed by increasing the number of switches connected to the controller. Finally the paper concluded that lowest throughput was showed by Floodlight and NOX controllers with approximately throughput of 900,000 flows/sec, while NOX throughput decreases severely on 128 switches, indicating that NOX controller is not suitable for large scale networks. The OpenMuL and OpenIRIS controllers have the highest throughput. According to the paper the performance difference relies on the algorithm that the controller uses.

3.3 Analysis of OpenFlow in Hybrid Mode

Even though applying OpenFlow architecture to the current environment will have many benefits as discussed by different researchers, getting rid of the traditional environment is not the main target in SDN. The basic concept in SDN is increasing flexibility to the current networking environment and this concept is highly supported by the hybrid-forwarding feature of OpenFlow. Hardware vendors published articles about their OpenFlow-hybrid mode supported switch products and tried to explain the hardware architecture and the benefits that the product will provide to the networking industries [4, 25].

As we can understand from the above reviewed papers, studying the performance of SDN brings enhancement to this new technology. Our work relies on making performance and scalability analysis to the OpenFlow hybrid architecture, which is a concept discussed in

Section 2.6 of this thesis. In addition to that all the above tests are used or performed on open source controllers. But, here we are going to use the controller developed by HP, this makes our work more close to the real world.

3.4 Summary

We separated the work in to three categories and discussed the studies done previously by different authors. As we can understand from the above works, performance analysis is a major task for the OpenFlow network for improving the architecture. In section 3.1, the review focused on the performance of OpenFlow protocol majorly on LAN, as stated above the evaluation is performed by taking different scenarios such as implementing OpenFlow for different topologies and measure the throughput, comparing OpenFlow network performance with the traditional network.

In section 3.2, we tried to review papers that are focused on controllers' performance. The papers perform the analysis on open source controllers such as NOX, Beacon, Terma, Floodlight and so on that is why we choose to use the HPE controller which is not an open source controller. Section 3.3 summarizes the work done on OpenFlow Hybrid architecture. We can't get enough paper that analyzes the hybrid concept of OpenFlow architecture. The paper we reviewed has a concept gap from the one that we are going to simulate and analyze in this paper. The work analyzes the hybrid architecture by configuring different VLAN's on the network but our work will measures the throughput of OpenFlow hybrid network by changing the controller configuration in order to support both traditional network traffics as well as OpenFlow traffics with a single network instance.

Chapter 4: Simulation for OpenFlow Hybrid Network Implementation

This Chapter analyzes throughput, delay, and scalability of OpenFlow Hybrid network implementation side by side with the pure OpenFlow network implementation.

4.1 Environment Setup

For this experiment two virtual environments with installed Ubuntu version 14.04 OS are prepared, one is for the HPE Virtual Application Network SDN controller setup and the other one is for the Mininet emulation software setup.

4.1.1 HPE VAN SDN Controller

There are different types of free and licensed OpenFlow controllers and HPE VAN SDN Controller is one of the controllers that require a license from HP. For this experiment we downloaded and configured the free version of the software in our Ubuntu machine. Since HP is one of the known network hardware vendors, it makes our work closer to the real world. In addition to that HPE VAN SDN Controller has a user-friendly interface that allows us to manage and control our network easily. Commands used to install this controller are listed in Annex A – Mininet and HP VAN SDN Controller. As we described in Section 3.3 using this controller makes our work result closer to the real world. The HPE VAN SDN Controller provides a unified control point in an SDN network, simplifying management, provisioning, and orchestration [17]. The VAN SDN Controller is designed to operate in campus, datacenter, or service provider environments. It enables a hybrid of both OpenFlow and normal packet processing with the HPE SDN architecture. Access control lists can be provisioned centrally, for example, while Layer2 or Layer3 forwarding decisions can be made using standard network protocols. This allows SDN concepts to be applied incrementally to the network, starting with the application of network policy and extending to exception-based forwarding, adding value without replacing traditional switching or routing. Figure 4.1 shows the screen capture of the HPE VAN SDN Controller Login page that we get after we finished the controller setup.

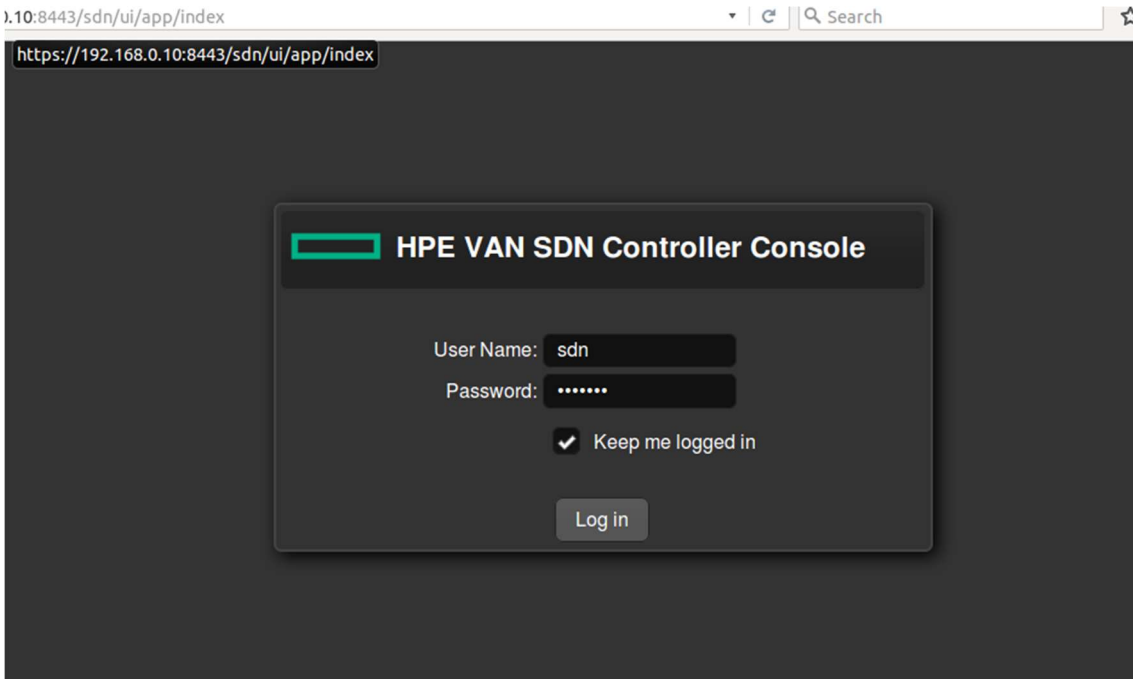

Figure 4.1: *HPE VAN SDN Controller Login Page*

4.1.2 Mininet

Mininet has the capability to emulate different kinds of network elements such as host, Layer 2 switches, Layer 3 routers, and links. It works on a single Linux kernel and it utilizes virtualization for the purpose of emulating a complete network utilizing only a single system [26]. Mininet has capabilities that enable researchers or network programmers to create Software Defined Network prototype in a simple manner. With the capability to interact, customize and share it provides a smooth path to running it on hardware. The following is a simple Mininet command that creates a tree topology with depth 2.

Mininet>sudo mn- -topo=tree,2- -controller=remote,ip=192.168.0.10

As we can understand from the command our VAN SDN controller is found on a remote machine with an IP address of 192.168.0.10. Hosts will be created automatically while the network devices are created. By default each switch will have two hosts unless we explicitly define the number of hosts we want the switch to have and hosts will be visible on the VAN SDN controller topology page when a connection is established between hosts. For simplicity, Mininet allows us to create a connection between each host with a single command *pingall*. The HPE VAN SDN Controller has a graphical interface that simplifies OpenFlow network

management. It has different modules used to configure and monitor OpenFlow network. For the analysis we will use the initial topology of Figure 4.2.

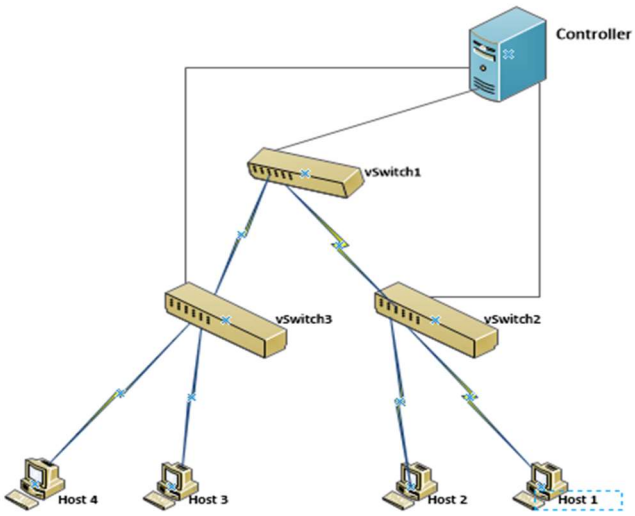

Figure 4.2: *Initial topology*

4.1.3 JPerf

JPerf is a tool to measure network performance for measuring bandwidth provided by the network. It is a tool to measure maximum TCP bandwidth, allowing the tuning of various parameters and UDP characteristics and reports bandwidth, delay jitter, and datagram loss [27]. It is an application that requires a Client/Server setup for measuring the network bandwidth. We will use it for measuring performance and delay of OpenFlow-Hybrid network. Figure 4.3 shows the graphical interface of JPerf network performance measurement tool.

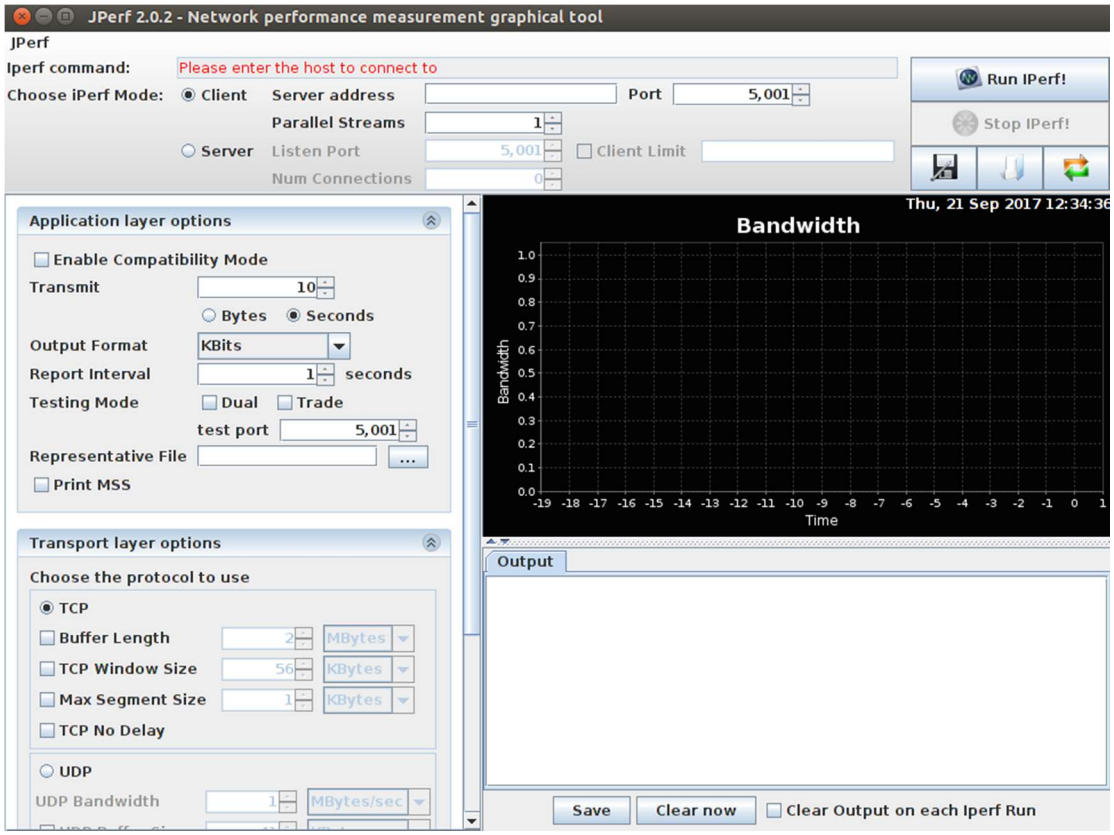

Figure 4.3: *JPerf Network Performance Measurement Graphical Tool*

4.2 OpenFlow-Hybrid mode Analysis on LAN

The target of OpenFlow-Hybrid mode is to integrate both the OpenFlow network with the traditional network. Controllers include a feature that allows us to implement this concept. The SDN VAN controller has a configuration section to change the type of OpenFlow mode that we want to implement. The default value of hybrid mode on the HPE VAN SDN Controller is 'true' which means if we add an OpenFlow switch to the network, the controller allows the switch to act as a normal Layer2 switch and can make forwarding decisions to the traffics when the controller connection is interrupted. We can change the key value to 'false' if we want to setup pure OpenFlow network. After we make a change restarting the controller service is required and the command for restarting the service is /*sudo service sdnc restart*/.

4.2.1 OpenFlow Traffic Analysis

In this experiment we will analyze the OpenFlow traffic and also assess TCAM table contents (Flow Rules) that an OpenFlow switch has for both the hybrid and the pure OpenFlow implementation.

The VAN SDN Controller has sections called OpenFlow Monitor and OpenFlow Trace that are used to analyze flows generated by the Controller and loaded to the vSwitches. We will use these two features to analyze traffics for both the hybrid and the pure OpenFlow implementation mode.

After creating our initial topology shown in Figure 4.2, we tried to monitor flows using the OpenFlow Monitor section of the HPE VAN SDN controller. For simplicity we executed the *pingall* command on the Mininet environment and it creates connectivity between all hosts on the network. Figure 4.4 shows flows installed on a switch for a hybrid OpenFlow network. The Flow Class ID column shows the class of flow rules where the packet is categorized and there are four class types where the packet is going to be categorized. The Actions/Instructions column shows how packets in each flow class are forwarded and as we can see the output path is either through the CONTROLLER or through the NORMAL port. The NORMAL port tells the switch to forward the packet according to the traditional networking pipeline.

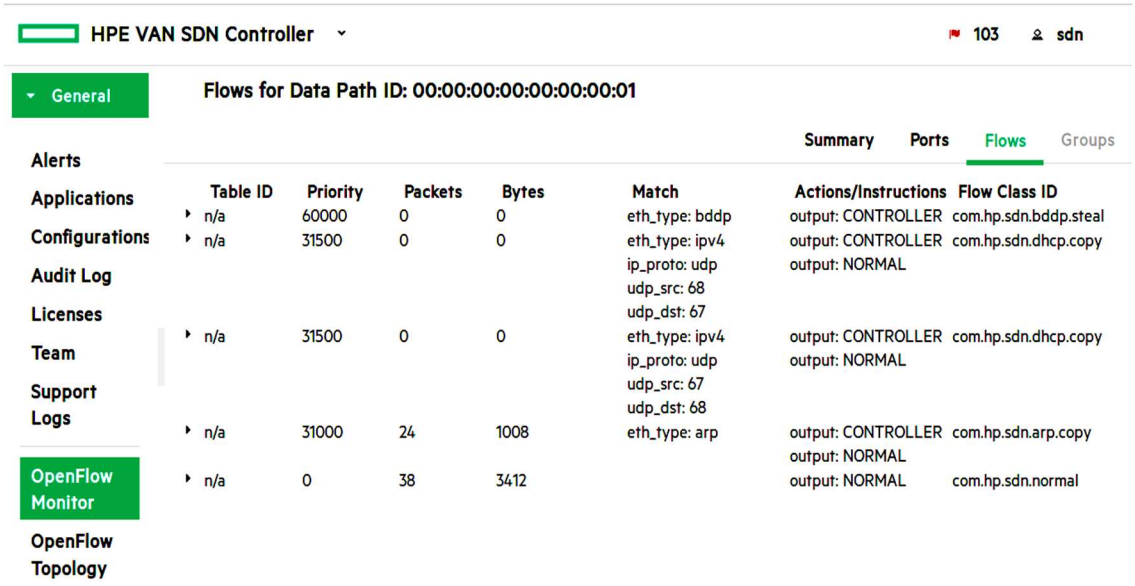

Figure 4.4: *Flows for Node 1 for Hybrid OpenFlow Implementation*

When we see Figure 4.5, Number of flows loaded on the switch is larger than the one we saw in Figure 4.4. This is because list of flows are loaded on each node by the controller for the pure OpenFlow implementation. Switches forward traffic according to the actions set by the controller. The Match field shows the *port number* where the packet comes in, the *source and*

destination address of hosts, and *Ethernet type.* We captured a portion of a flow table that can show both Layer3 and Layer2 traffics.

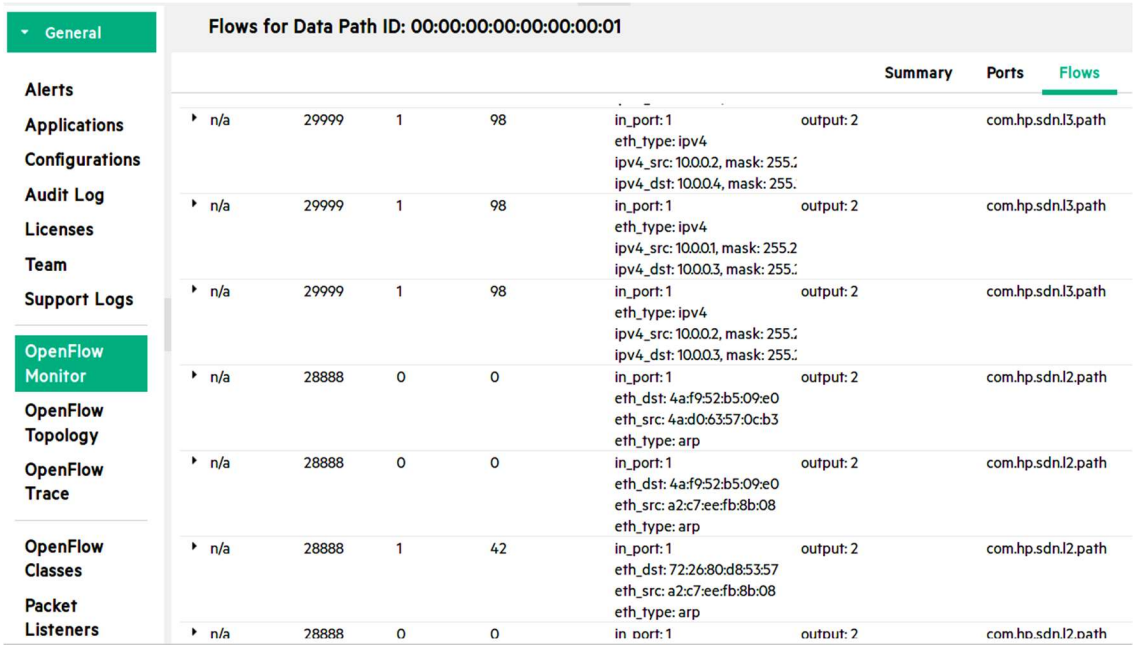

Figure 4.5: *List of Flows Generated for Node1 on Pure OpenFlow Implementation*

4.2.2 Network Throughput Analysis

We are using Jperf application described in Section 4.1.2for measuring the throughput difference between the pure and hybrid OpenFlow implementations. Jperf allows us to measure the actual rate of information transferred for the two implementations. We are going to use the initial topology in Figure 4.2for the test. We set Host1 as a Server and Host4 as a Client node from our initial topology. Accessing each host and run JPerf on hosts are described in Annex B–Runing JPerf on Hosts. The throughput measurement is performed through the TCP protocol and the results are printed in Mbytes/sec. Figure 4.6 shows throughput result we get for the hybrid OpenFlow network implementation.

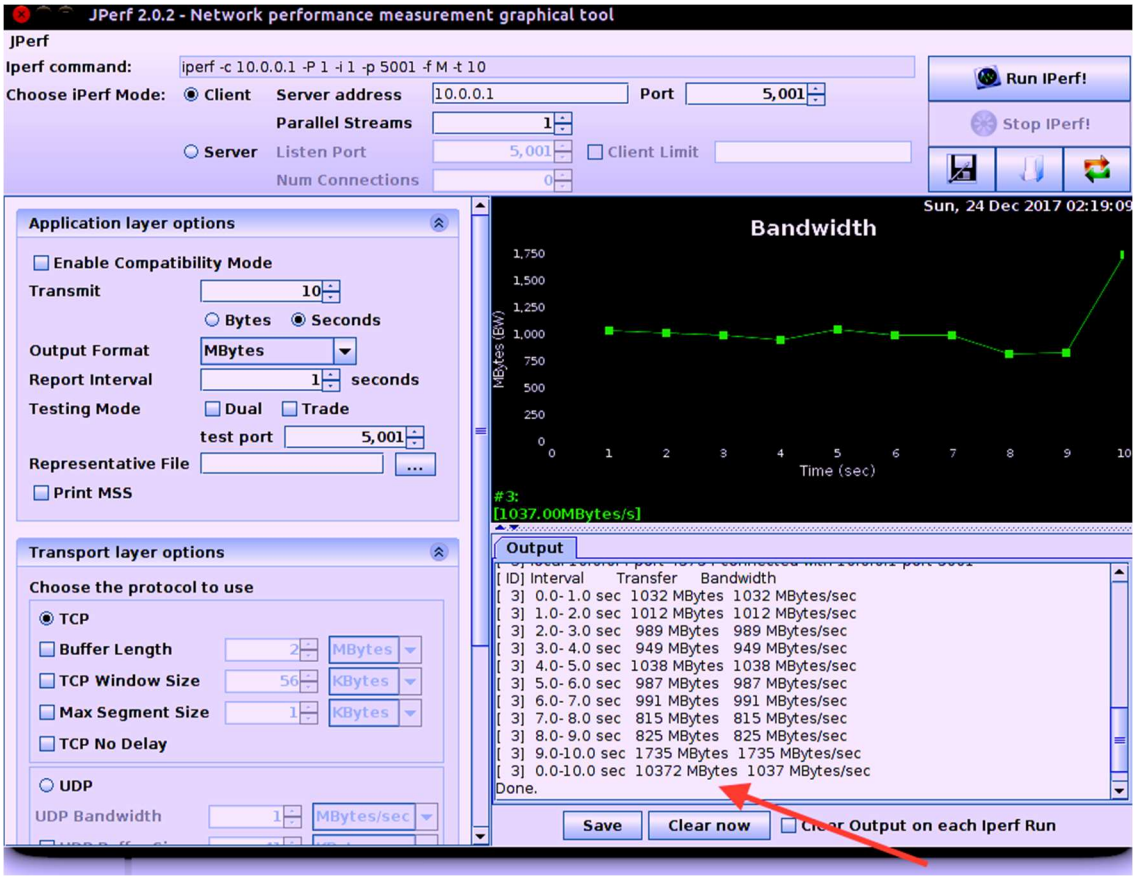

Figure 4.6: *Throughput of Hybrid OpenFlow Implementation*

As we can see, for the hybrid implementation a total of 10372 Mbytes of data is transferred within 10 seconds and bandwidth result shows 1037 Mbytes of data is transferred in a second. For the pure OpenFlow implementation the throughput result in Figure 4.7 shows that24 Mbytes of data is transferred in 10 seconds and the bandwidth utilization is 2.42 Mbytes per second.

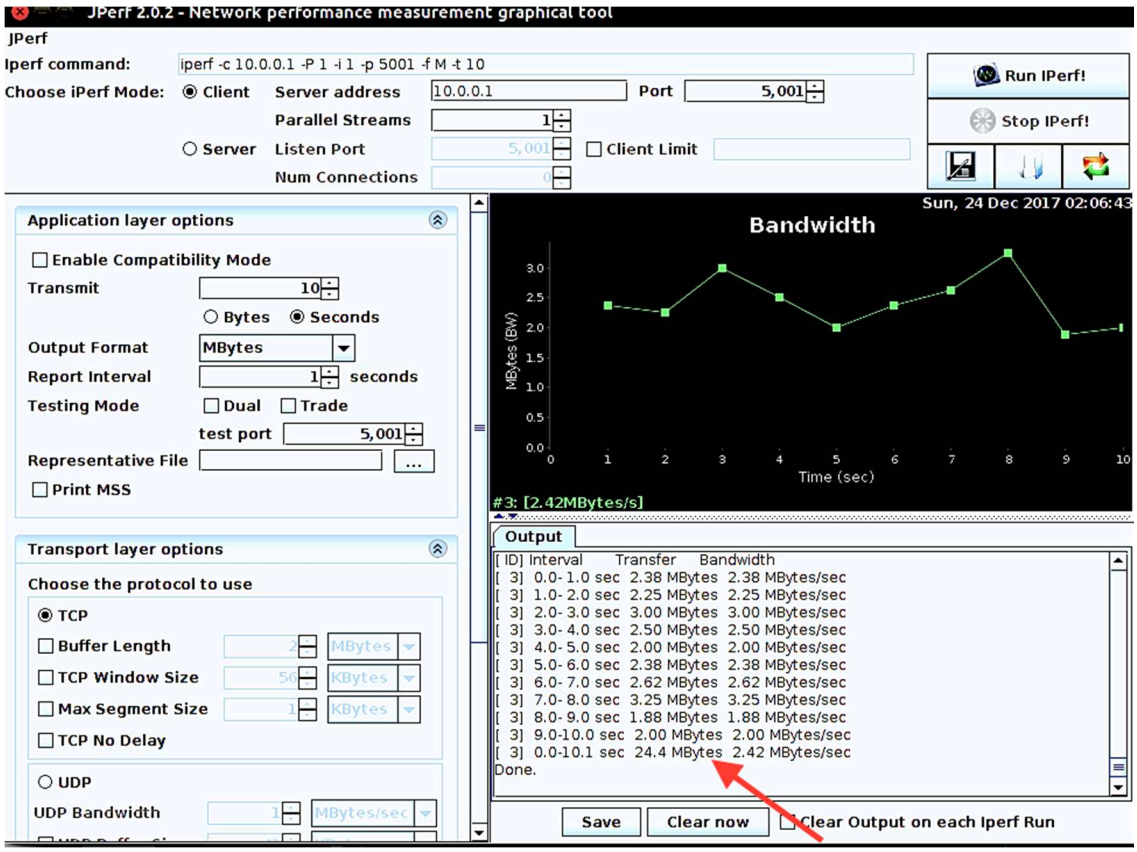

Figure 4.7: *Throughput of Pure OpenFlow Implementation*

Based on the analysis result we get we can conclude that the hybrid OpenFlow network implementation has a massive performance than the pure OpenFlow network implementation. For assurance of the generated output, we used Jperf network throughput analysis parallel streams feature that allows us to send multiple parallel connection streams to the server. Figure 4.8shows the result we get by setting the parallel stream value equal to five.

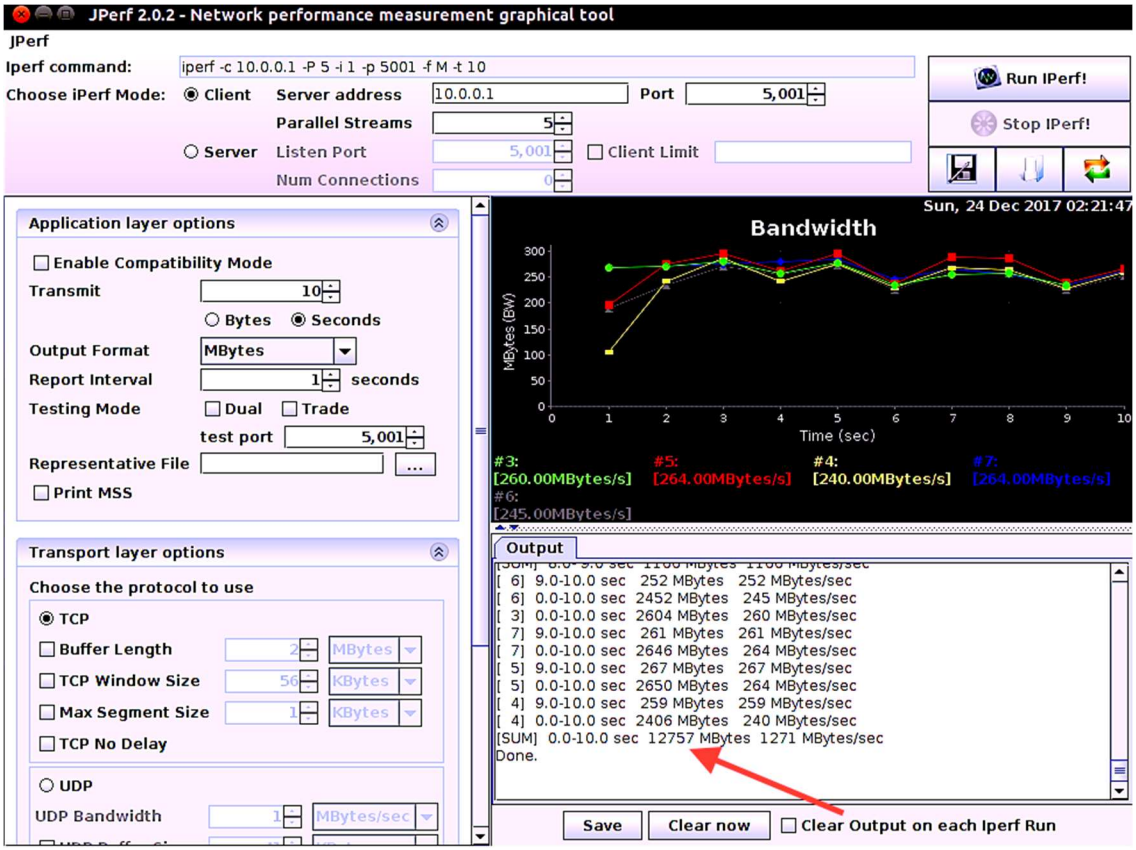

Figure 4.8: *Parallel Streaming Result of Hybrid OpenFlow Implementation*

When parallel connection stream is sent to the server, each connection shares the total bandwidth and it is obvious that data transfer rate will be decreased. In Figure 4.8 we can see that the total transferred data for each stream declined from 10,372 Mbytes to an average of 2500 Mbytes of data in 10 seconds. The main point here is how well our network utilizes bandwidth for the transferred data. Pure OpenFlow implementation has difficulty for utilizing the bandwidth when parallel connection is established between client and server. As we can see in Figure 4.8 the data transfer rate dramatically decreased when parallel streaming is used on the pure OpenFlow network because for each request, the switch asks the controller what action should be taken for the incoming packet and that creates too much traffic between them.

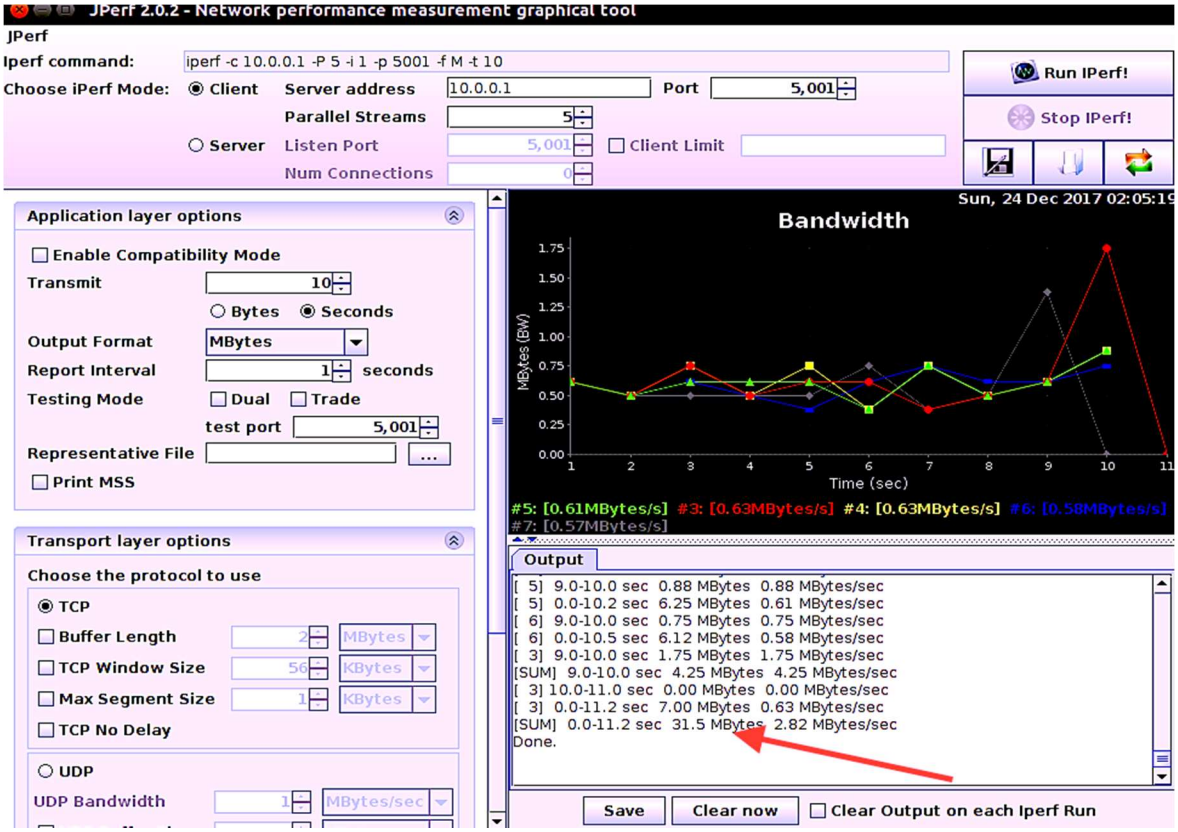

Figure 4.9: *Parallel Streaming Result of Pure OpenFlow Implementation*

4.2.3 Network Delay Analysis

In this section comparison of the two OpenFlow implementations is done on the basis of delay between nodes. This can be achieved by finding out the round-trip time (rtt) between nodes by executing 'ping' connectivity test. We use Host1 and Host4 from our initial topology for measuring the rtt. A round-trip delay between nodes for the two OpenFlow implementations with variable PTR is tabulated in Table 4.1.

Table 4.1: *OpenFlow Network Delay Summary*

No of Packets Transmitted	Pure OpenFlow Implementation		Hybrid OpenFlow Implementation	
	Max rtt (ms)	Min rtt (ms)	Max rtt (ms)	Min rtt (ms)
10	2.683	0.980	0.071	0.057
20	5.129	0.969	0.418	0.052
30	4.990	1.218	0.296	0.059
40	4.578	1.015	0.812	0.060

The value of umber of packets transmitted is used in the analysis only to show the consistency of our result it doesn't affect the rtt value. We tried to show the rtt result graphically in Figures4.10 and 4.11for maximum and minimum delay.

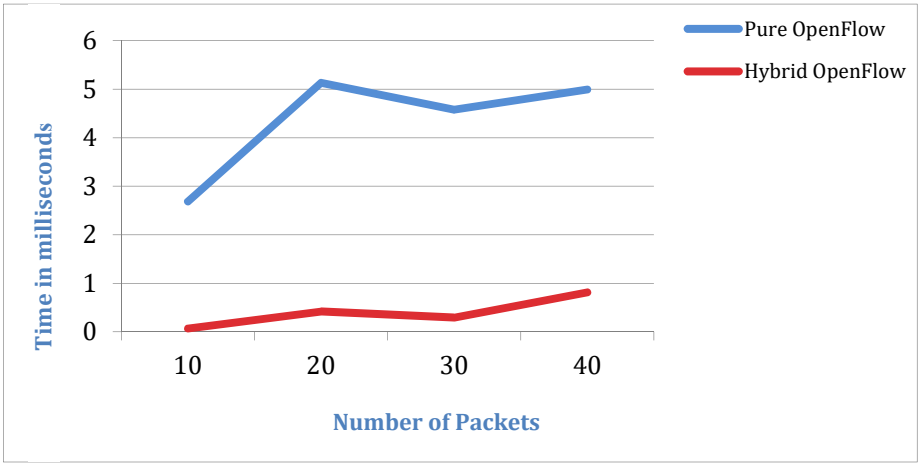

Figure4.10: *Maximum Delay Result*

The minimum and maximum delay results for both implementations show that the Pure OpenFlow implementation encounters more delay than the Hybrid OpenFlow implementation. It is also noticed from the graph that the Hybrid OpenFlow implementation has somehow a constant flow for both maximum and minimum results. The pure OpenFlow implementation has a variable delay result and this is the result of the connection that each Switch made with the controller for requesting flow rule action.

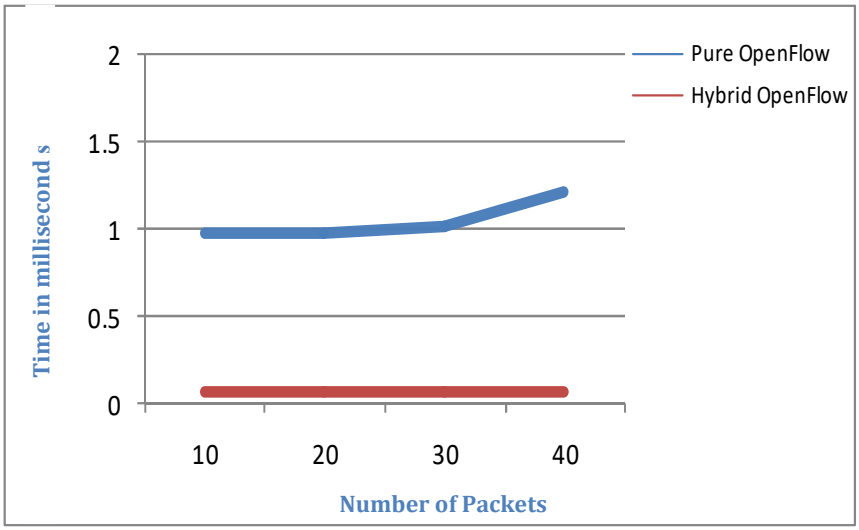

Figure 4.11: *Minimum Delay Result*

4.2.4 Scalability Analysis in terms of flow rules

This experiment will assess scalability of the OpenFlow-hybrid mode in terms of the number of OpenFlow rules that are loaded to the switches by the controller. This scalability assessment is measured based on the number of end nodes (hosts) available on the topology. We created a tree topology with depth one on Mininet environment using a command *sudo mn –topo tree,1,fanout*=[*Number of Node*s]. Table 4.2 describes the number of flow rules that the OpenFlow switch will have when number of hosts vary for the two implementations.

Table 4.2: *Number of Flow Rules*

Number of Nodes	Pure OpenFlow Implementation	Hybrid OpenFlow Implementation
	Number of Flow Rules	**Number of Flow Rules**
4	24	4
10	180	4
16	480	4
20	760	4
24	1104	4

Looking for a specific flow rule becomes very difficult and time taking when the flow table is big. The above scalability test is measured using a simple tree topology with depth value equals one and if we increase the depth value, the flow table will contain a large number of flow rules for the pure OpenFlow implementation and that makes the TCAM table space fully utilized to accept new flow rules. When we see the hybrid implementation, the switches have the same number of flow rules in their flow table even if the number of hosts is increased.

4.2.5 Connection Problem Identification

As we saw above, the NORMAL forwarding mechanism in switches is the default for handling traffics in the hybrid OpenFlow mode, i.e., packets are never sent to the controller for getting forwarding decisions. Hybrid OpenFlow is used to selectively override default forwarding in order to improve performance. This approach is extremely scalable and robust but there is minimal overhead associated with maintaining the OpenFlow connection between the controller and the switches. The network devices still continue to forward traffic even if the controller fails. This is because the switches that support the hybrid OpenFlow mode have the ability to change their operating mode to 'fail standalone' when a connection to the controller is lost [9].Fail standalone mode allows the switches to operate as traditional Layer2 switch that can make forwarding decisions for the incoming packets.

We use the initial LAN topology in Figure 4.2 for this section also. It is designed on Mininet for testing the connection between hosts by disabling the controller service. We tested a connection between Host1and Host4 using a simple ping command on the Mininet environment. In the case of pure OpenFlow implementation when Host1 sends a ping request, Host4 replies for the request since the controller service is up. When we stop the controller service, the ping reply continues because the flow between Host1 and Host4 is already installed on switches that the Hosts are connected but this flow entry is valid until the flow expiry time is reached. However, after stopping the controller service hosts other than Host1 and Host4 and hosts that are connected to different switches can't communicate with each other. This is because the controller is the decision maker for all flows in the network. The system administrator can identify and notice the failed controller easily. Figure 4.12 shows that Host1 is getting Destination Host Unreachable message when ping request to Host3 is sent after the controller service is stopped.

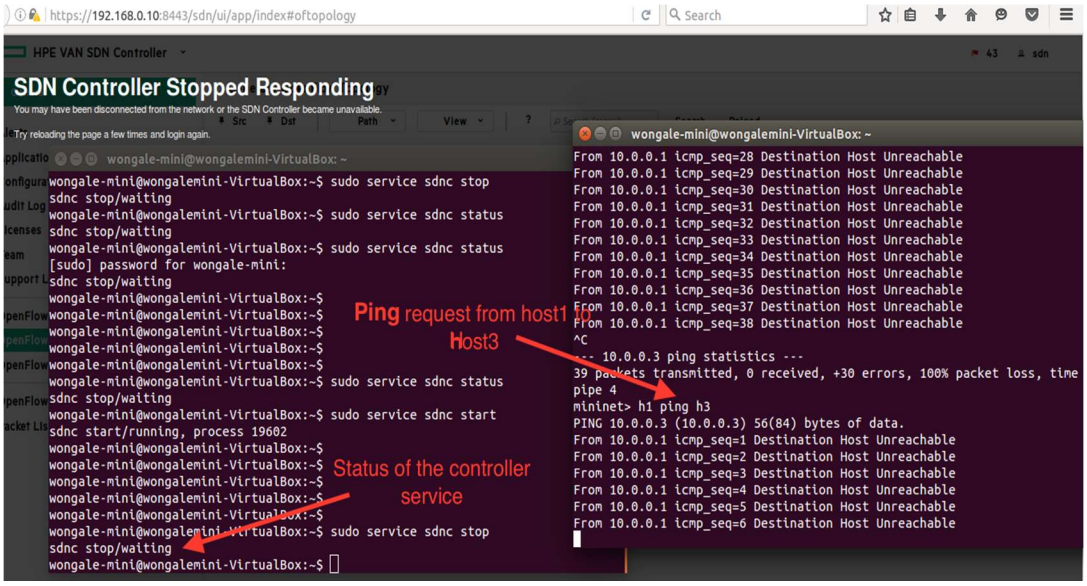

Figure 4.12: *Ping Request from Host1 to Host3 After The Controller Service Stopped*

In the case of hybrid OpenFlow network implementation; since switches can make forwarding decision for the incoming requests, the communication between hosts continues even if the controller is not active unless there are external APIs that need the existence of the controller. As we can see in Figure 4.13 we were pinging Host3 from Host1 while we are stopping the controller service but after stopping the service we tested the communication between Host1 and Host4 and it is not interrupted.

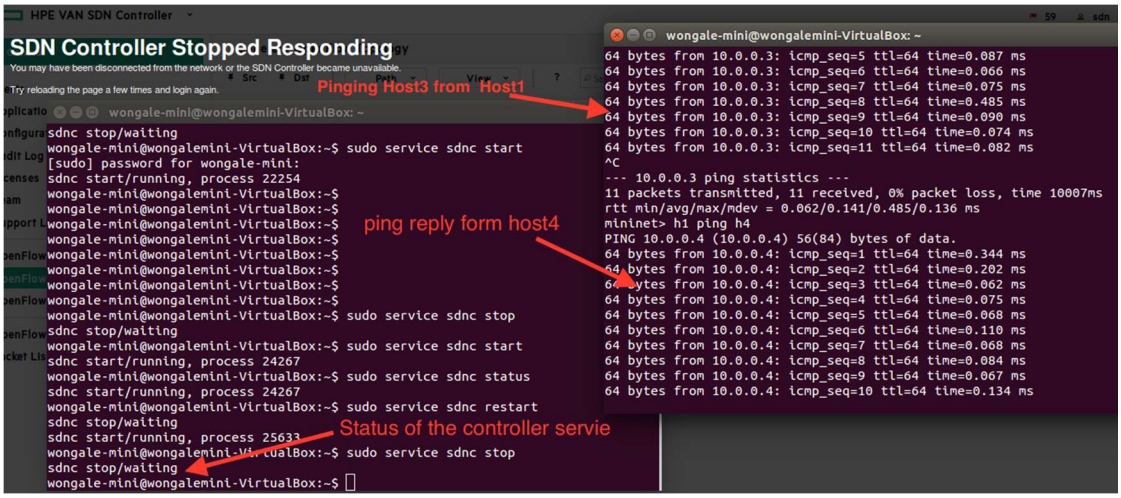

Figure 4.13: *Ping Request from Host1 to Host4After The Controller Service Stopped*

This is one of the drawbacks of the hybrid OpenFlow implementation because connection problem between the controller and the switch cannot be easily identified.

4.3 Summary

The idea of OpenFlow-Hybrid network comes to overcome the problem of integrating SDN with already existed traditional networks. In this chapter we setup a simulation environment on Mininet to experiment performance, delay, and scalability of the hybrid OpenFlow implementation side by side with the pure OpenFlow implementation on a simple LAN topology. HP VAN SDN controller is installed on our Ubuntu machine and used for the simulation. Our result shows that the Hybrid implementation provides a better performance with a minimum delay value than the Pure OpenFlow implementation. In addition to that, the Hybrid OpenFlow implementation is more scalable because the number of flow rules installed on each OpenFlow switch is very limited and this solves the storage problem existed on the switches TCAM table. As we saw, switches themselves can make forwarding decisions for the incoming packets and it minimizes the burden of the controller even if the controller is in charge of managing the network. We found that the Hybrid implementation has a problem on maintaining the failed connection between switches and controller. The communication between hosts look stable even if the controller is down unless there are external applications that require the presence of the controller and this is a drawback we found for the Hybrid OpenFlow implementation.

Chapter 5: Open vSwitch Implementation

In this Chapter we will discuss the Open vSwitch modules and the modification we made for the Open vSwitch source code for the problem identified in Section 4.2.5.

5.1 Open vSwitch Modules

Open vSwitch has different modules that collaborate to support and converge OpenFlow packets. These modules are mostly written in C programming language. For the installation process we followed commands in Annex C –Installation of Open vSwitch.

As we can understand from our analysis, the hybrid implementation of the OpenFlow network brings enhancement over the pure OpenFlow network implementation, mainly on scalability and performance. However, it has connection maintenance problem. The system administrator can't easily identify the problem unless there is a specific application that requires the existence of the controller. We tried to modify the source code of the Open vSwitch module, which is related to OpenFlow. We are using Open vSwitch components discussed in Section 2.7.2 for querying and modifying the source code.Figure5.1 shows the notification process.

Figure 5.1: *Activity Diagram for Failed Controller Notification Process*

If we choose the switch fail-mode to work as standalone, we are assuring the availability of our network by letting the switches to work as normal Layer2 forwarding switches. The

default fail-mode for the Open vSwitch is secure, that means the switch waits the controller to forward traffics. Fail-mode value can be changed using the following Open vSwitch command on Linux.

 # ovs-vsctl set-fail-mode s1 standalone

#ovs-vsctl show

The second command lists the current configuration for all virtual switches we have in our network and the output looks like:

```
wongale-mini@wongalemiffi-VirtualBox:~/openvswitch/openvswitch-2.0.0/ofproto$ sud
o ovs-vsctl show
[sudo] password for wongale-mini:
2cd224d0-5dab-42dc-b500-af581181da27
    Bridge "s1"
        Controller "ptcp:6654"
        Controller "tcp:192.168.0.10:6633"
        fail_mode: standalone
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1"
            Interface "s1"
                type: internal
        Port "s1-eth2"
            Interface "s1-eth2"
    Bridge "s3"
        Controller "tcp:192.168.0.10:6633"
        Controller "ptcp:6656"
        fail_mode: standalone
        Port "s3-eth2"
            Interface "s3-eth2"
        Port "s3-eth1"
            Interface "s3-eth1"
        Port "s3-eth3"
            Interface "s3-eth3"
        Port "s3"
            Interface "s3"
                type: internal
    Bridge "s2"
        Controller "tcp:192.168.0.10:6633"
        Controller "ptcp:6655"
        fail_mode: standalone
        Port "s2-eth3"
            Interface "s2-eth3"
        Port "s2-eth1"
            Interface "s2-eth1"
        Port "s2"
            Interface "s2"
                type: internal
        Port "s2-eth2"
            Interface "s2-eth2"
    ovs_version: "2.0.0"
```

As we can see the output of the show command contains information such as the switch name, current controller IP address, fail-mode value, interface port, and the version of the Open vSwitch currently installed.

As we discussed the Open vSwitch has different modules and we are using specifically the *connmgr.c* module for our work that is a sub module of *ofproto* module. The *connmgr.c* module is used in Open vSwitch to track and monitor the OpenFlow switch network connection. We modified the source code of a function called *connmgr_flushed()* to send alert message to a log file when the controller connection is interrupted while the switch fail-mode is standalone.

```
void
connmgr_flushed(struct connmgr *mgr)
    OVS_EXCLUDED(ofproto_mutex)
{
    if (mgr->fail_open) {
        fail_open_flushed(mgr->fail_open);
    }

    /* If there are no controllers and we're in standalone mode, set up a flow
     * that matches every packet and directs them to OFPP_NORMAL (which goes to
     * us).  Otherwise, the switch is in secure mode and we won't pass any
     * traffic until a controller has been defined and it tells us to do so. */
    if (!connmgr_has_controllers(mgr)
        && mgr->fail_mode == OFPROTO_FAIL_STANDALONE) {
        struct ofpbuf ofpacts;
        struct match match;
VLOG_INFO("The controller connection is down please check.....");
        ofpbuf_init(&ofpacts, OFPACT_OUTPUT_SIZE);
        ofpact_put_OUTPUT(&ofpacts)->port = OFPP_NORMAL;

        match_init_catchall(&match);
        ofproto_add_flow(mgr->ofproto, &match, 0, ofpacts.data,
                                                  ofpacts.size);

        ofpbuf_uninit(&ofpacts);
    }
}
```

For this test we downloaded and configured the current version of Open vSwitch software from the Open vSwitch official site [28] in our Ubuntu machine.

After we saved the changed source code we compiled the source code using a Linux command *make –j.* We need to commit the changes we made using the git graphical user interface. Git is a software version checker [29].The GUI can be called by writing *git citool* command in the command line.

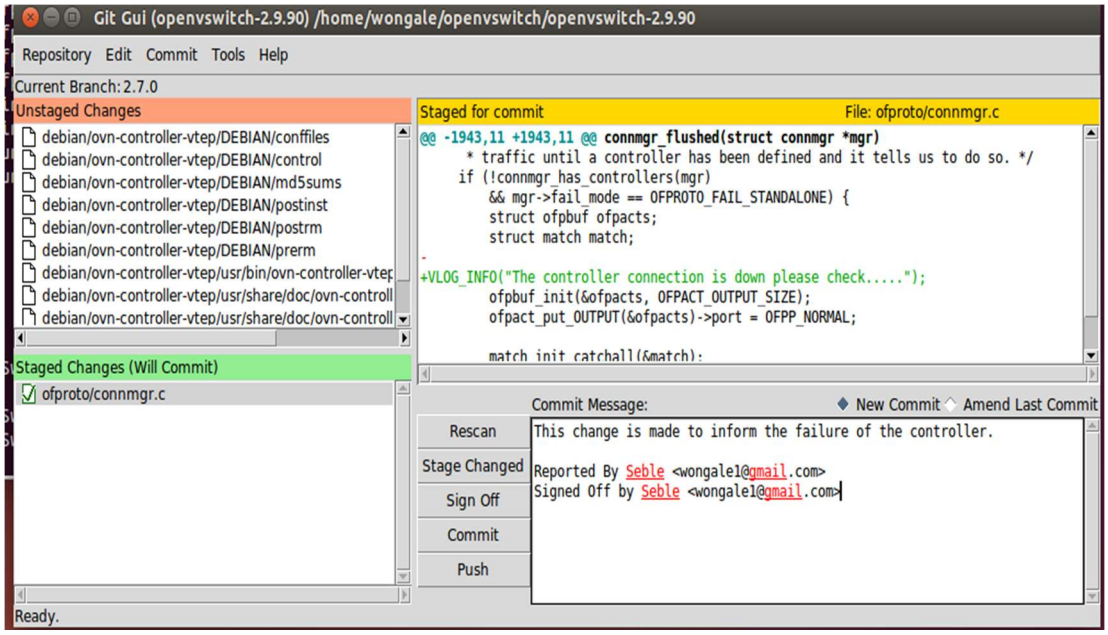

Figure 5.2: *Git GUI for Committing Changes*

As we can see from Figure 5.2 git already identifies the module we changed and we moved it from the Unstaged state to Staged state in order to commit the change. Once the change is compiled and committed we created our LAN topology on the Mininet and initiate traffic between devices. We stopped the controller service while the Open vSwitch fiail-mode is standalone to fulfill the *if()* condition. When we see the log file in */var/log/ovs-vswitchd.log*, it contains our alert message. Finally we can use a script in Annex D – Send Mail from Linux to extract and send the alert message in the log file to the system administrator's mail.

5.2 Summary

From our analysis in Chapter 4, we found that the Hybrid implementation has a drawback for maintaining connection interruption between switches and controllers. We tried to understand the Open vSwitch source code and made modification in a module called *connmgr.c* to send alert message to the log file when the connection between network devices and controller is lost while the switch fail-mode is standalone. The git commit interface tool is used to commit the compiled modules. Finally we extracted the alert message from the log file in the default directory /var/log/*openvswitch* and send it to the system administrator through mail using a linux script.

Chapter 6: Conclusion and Future Works

6.1 Conclusion

In this thesis work we tried to simulate and elaborate OpenFlow-Hybrid mode network architecture majorly focusing on performance, delay, and scalability of the architecture by comparing it with the Pure OpenFlow implementation. As discussed on this work, the hybrid architecture has the ability to introduce SDN concept on the existing traditional network. We tried to make performance, delay, and scalability analysis on this architecture by comparing it with the pure OpenFlow architecture, which is the predecessor. For both analyses we installed Mininet emulator software on Ubuntu machine. Mininet is used to create the topology we used for the whole work and it is very easy to integrate with the Controller, in our case; HPE VAN SDN Controller. HPE VAN SDN Controller is a java based OpenFlow controller owned by HP. It has a user-friendly interface that enables us to manage our devises as well as to trace the OpenFlow flows very easily and quickly. Using this controller makes our work more close to the real world. There are also additional tools like Jperf and git used for traffic and throughput analysis. From our analysis result, the Hybrid OpenFlow mode implementation has a better throughput than the pure OpenFlow implementation mode. This is the result of having minimum number of communication between the controller and the Switch when we compared it with the pure OpenFlow implementation. In addition to that, the OpenFlow network scalability is measured by increasing the number of hosts on the network by six starting with four hosts. Our analysis shows that the Pure OpenFlow implementation has a very big flow table because of the number of flow rules that are loaded on the Switches by the controller and it has a negative impact on the entire OpenFlow network scalability. The Hybrid OpenFlow implementation has a constant number of flow rules for variable number of hosts on the network. Switches can make decisions for the incoming packets without the involvement of the controller. This result shows that the Hybrid OpenFlow implementation brings a big advantage for OpenFlow network scalability since it allows adding more network devices in the network without worrying the TCAM table space. Finally we tried to modify the open vSwith source code in order to generate alert message when the connection to the controller is lost even if the Switches are still active and has the ability to make forwarding decisions for the incoming packets. This helps the system administrator to be alerted when the status of the OpenFlow network changes.

6.2 Future Works

We made the analysis on virtual network devices on Mininet. However, we found as future work, since Mininet has the ability to integrate real network devices with the virtual network, we can get the analysis result as accurate as the real world.

References

[1] Thomas D. Nadeau and Ken Gray, "*SDN (Software Defined Network)*", 1st edition, O'Relly Media, Augest2013.

[2] Bjorn R. Martinussen, "Introduction to Software Defined Networks (SDN) and its relevance in the DC", *Cisco Connect,* April 13, 2013.

[3] Diego Kreutz, "Software-Defined Networking": *A Comprehensive Survey*", January 2015.

[4] Shaun Wackerly, "OpenFlow-Hybrid Mode", *Linux Foundation Colaborative Projects*, *Open Daylight,* August 2014.

[5] Wolfgang Braun and Michael Menth, "Software-Defined Networking Using OpenFlow Protocols*", Applications and Architectural Design Choices* ", May 2014.

[6] Shie-Yuan Wang, Chih-Liang Chou, and Chun-Ming Yang, "EstiNet OpenFlow and Network Simulator and Emulator*", IEEE communications Magazininc*, September 2013.

[7] Open Networking Foundation, "SDN Architecture", white paper, June 2014, retrieved from https://opennetworking.org, Last accessed on May 2017.

[8] CISCO, "Empowering The Network: SDN", August 2014, [Online], retrieved from http://www.cisco.com/c/en/us/solutions/software-defined-networking/overview, Last accessed on December 4, 2016.

[9] ONF, "Open Networking Foundation"**,** September 2013, retrieved from www.open-networking.org, Last accessed on February 4, 2017.

[10] Nick McKeown, Guru Parulkar**,** "OpenFlow: Enabling Innovation in Campus Networks", *Stanford University, May 14, 2008.*

[11] Rowan Kloti,Vasileios Kotronis, and Paul Smith, "OpenFlow: A Security Analysis", *IEEE*, 2013.

[12] ONF, "OpenFlow Switch Specification v1.3.4", *Open Networking Foundation*, September 2012, retrieved from https://www.opennetworking.org/images/stories/ downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf, Last accessed on April 2017.

[13] William Stallings*,* "Software-Defined Networks and OpenFlow*", The Internet Protocol Journal, Volume 16, No. 1,* March 2013.

[14] Ashton, Metzler and Associates, "Ten Things to Look for in an SDN Controller*", Leverage Technology and Talent for Success*, 2013.

[15] Ahmed Sonba and Hassan Abdalkreim, "Performance Comparison of the State of the Art OpenFlow Controllers", *Halmstad University*, December 2014.

[16] Filipe Azevedo, "A Scalable Architecture for Openflow Controllers", 2015, retrieved from https://www.semanticscholar.org/paper/A-Scalable-Architecture-for-Openflow-Controllers-Azevedo, Last accessed on July 2017.

[17] Hewlett-Packard Development Company, “HP SDN hybrid network architecture”, *Hewlett-Packard*, 2015.

[18] Matt Conran, “Open vSwith (OVS) Basics”, November 2015, retrieved from http://network-insight.net /2015/11/open-vswitch-ovs-basics, Last accessed on March 4, 2018.

[19] Ben Pfaff, Justin Pettit, Teemu Koponen, “The Design and Implementation of Open vSwitch”, *The Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation*, *USENIX*, 2015.

[20] Serrnivas Makam, “Openvswitch and ovsdb”, January 2, 2014, retrieved from https://sreeninet.wordpress.com/2014/01/02/openvswitch-and-ovsdb/, Last accessed on October 2016.

[21] Idris Zoher Bholebawa, Upena D. Dalal, “Design and Performance Analysis of OpenFlow-Enabled Network Topologies Using Mininet”, *International Journal of Computer and Communication Engineering*, April 2016.

[22] Michael Jarschel, Simon Oechsner, Daniel Schlosser, Rastin Pries, Sebastian Goll, Phuoc Tran-Gia, “Modeling performance analysis of OpenFlow”, 2011.

[23] Khatri Vikramajeet, “Analysis of OpenFlow Protocol in Local Area Networks”, *Tampere University of Technology*, August 2013.

[24] Guillermo Romero and Tejada Muntaner, “Evaluation of OpenFlow Controllers”, October 15, 2012

[25] Dell Networking, “A look at a hybrid-enabled OpenFlow deployment”, *Dell*, 2015.

[26] Faris Keti and Shavan Askar,“Emulation of Software Defined Networking using Mininet in Different Simulation Environments”, *6th International Conference on Intelligent System, IEE Computer Society*, 2015.

[27] Jon Dugan, “*Summer Joint Techs,* Iperf Tutorial”, *Energy Sciences Network,* 2010.

[28] Open vSwitch Developers, “Open vSwitch Documentation” white paper, January 2017, retrieved from http://readthedocs.org/ovs-reviews.pdf, Last accessed on February 2018.

[29] Tutorials Point, “GIT fast version control”, 2014, retrieved from www.tutorials-point.com, Last accessed on March 2018.

Annex A– Installation of Mininet and HP VAN SDN Controller

Mininet and HP VAN SDN Controller installation steps on Ubuntu 14.04 LTS Operating System for simulation.

Mininet Installation Steps and Commands

- *~$ Sudo apt-get install mininet*
- *~$ Sudo mn -c*
- *~$ Sudo apt-get install git*
- *~$ git clone git://github.com/mininet/mininet*
- *~$ mininet/util/install.sh*

HP SDN VAN Controller installation steps

We downloaded the software from http://bit.ly/1H7h1Rg and install the software using the HP VAN SDN Controller using the installation guide provided by HP. In our case since Ubuntu 14.04 OS is using, we installed the controller by following the Debian Package and a local key stone server installation guide.

Key commands

- *~$ sudo apt-get update*
- *~$ sudo apt-get install python-software-properties*
- *~$ sudo apt-get install ubuntu-cloud-keyring*
- *~$ sudo add-apt-repository cloud-archive:juno*
- *~$ sudo apt-get install keystone*
- *~$ sudo dpkg --unpack hp-sdn-ctl_2.7.x.yyyy_amd64.deb*
- *~$ sudo apt-get install –f*

The above lists of command are used during the installation of the controller and some of the commands require a response from the user. Finally we can verify the installation using the command below.

- *~$ sudo dpkg -l hp-sdn-ctl*

Annex B – Running Jperf on Hosts

Network throughput analysis is performed using the software called Jperf. We downloaded the Jperf software and start the script on the hosts that we want to measure the throughput.

- *$ cd jperf-2.0.2*
- *$ sudo ./jperf.sh*

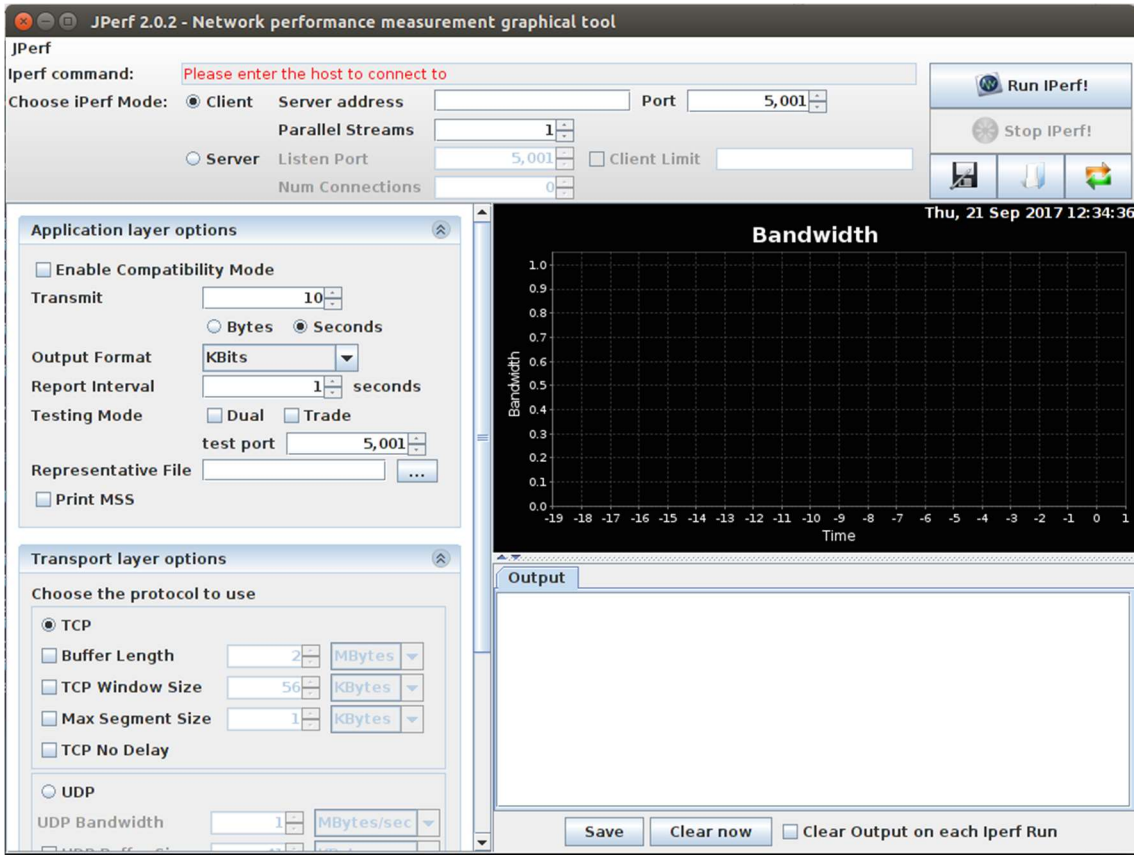

Figure B.1: *Jperf Performance Analyzer Window*

As we can see from the above window it has Client or Server mode options, whatever our choice is we should provide the address of the server. We can run the Jperf on each individual hosts and each host can be accessed using a command *xterm h1* on the Mininet window.

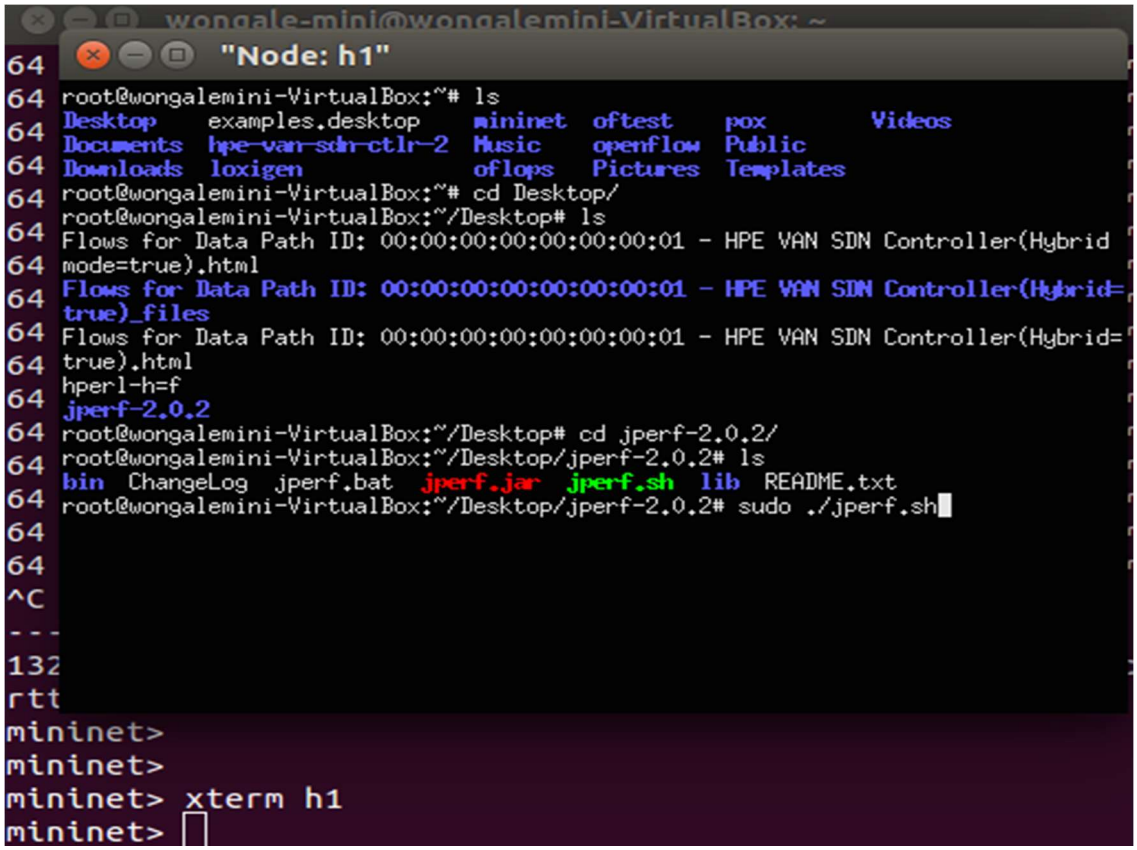

Figure B.2: *Accessing Node1 on Mininet*

Annex C – Open vSwitch Installation

In this Section the installation steps taken for Open vSwitch, git, and sendEmail will be listed.

Open vSwitch

Open vSwitch is available for us in two forms:

1. Open vSwitch Tarboal: It is a .tar file that contains zipped Open vSwitch software that can be installed direct from the Open vSwitch site or it can be downloaded to our machine and will be installed latter.

   - *$ sudo wget http://openvswitch.org/releases/openvswitch-2.7.0.tar.gz*

2. Open vSwitch git Repository: We can download a git repository if we want to update the source code and compiled the change. It has a gui that enables us to see the repository tree and each changes made on the repository.

   - *$ git clone https://github.com/openvswitch/ovs.git*

After downloading the tar file or the git repository, three basic commands are used to install the OVS. But there may be other dependency packages needed depends on the OS we are installing the OVS.

- *$ Sudo./boot.sh*
- *$ sudo ./configure*
- *$ sudo make*
- *$sudo /etc/init.d/open vswitch-switch start*

Git

Git is a distributed version control system. It is a free and open source system that anybody can use it freely over the Internet. In order to support git repository on Ubuntu machines, git packages should be installed first.

- *$ sudo get-apt install –y git*
- *$ git checkout v2.7.0*

The second command helps to check the software version that git contains in its repository.

Annex D – Send Mail from Linux

sendEmail

A linux command that allows us to send email messages from the command line interface. The following command is used to install the send Email application.

- *$ sudo apt-get install sendEmail libio-socket-ssl-perl libnet-ssleay-perl*

A bash script used to send mail on linux:

#!/bin/bash

FROM:wongale1@gmail.com
TO:wongale1@yahoo.com
SERVER:smtp.googlemail.com:587
USER:wongale1
PASSWORD: mypassword
SUBJECT:"Alert Message"

sendEmail –f $FROM –t $TO –s $SERVER –u $SUBJECT –xu $USER

–xp $PASSWORD –o message-file=/var/log/openvswitch/ovs-vswitchd.log

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:

Name: _____

Signature: _____

Date: _____

Confirmed by advisor:

Name: _____

Signature: _____

Date: _____