2012

DYNAMIC NETWORK TRAFFIC ISOLATION THROUGH OPENFLOW

Anisha Kolasani
University of Nebraska-Lincoln

DYNAMIC NETWORK TRAFFIC ISOLATION THROUGH OPENFLOW

by

Anisha Kolasani

A PROJECT REPORT

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science and Engineering

Under the Supervision of Dr. Byrav Ramamurthy

Lincoln, Nebraska

August, 2012

Department of Computer Science & Engineering, University of Nebraska-Lincoln,
Technical Report, TR-UNL-CSE-2012-0009

DYNAMIC NETWORK TRAFFIC ISOLATION THROUGH OPENFLOW

Anisha Kolasani, M.S.

University of Nebraska, 2012

Adviser: Dr. Byrav Ramamurthy

OpenFlow is the latest and most widely accepted networking technology which is used to realize the paradigm changing concept of Software Defined Networking (SDN). OpenFlow strongly advocates the separation of a switch's control plane from the data plane and a centralized controller to control the entire network.

Traffic isolation enables greater security for network communication, along with managing the bandwidth of the network more efficiently and providing logical separation between hosts that need to work together. But, dynamically managing the traffic isolation in a network is a very tedious task. Network management applications using OpenFlow for addressing this problem are not widely available. We propose two approaches to solve this problem using OpenFlow.

We developed two OpenFlow controller applications 'OFModifyVLAN' and 'OFWhiteListing', for addressing the above problem in short-term and long-term dynamic scenarios, respectively. We configured multiple OpenFlow network platforms using Mininet, Open VSwitch and HP Procurve switch to test the working and performance of the two OpenFlow controller applications. We tested 'OFModifyVLAN' on the Open VSwitch network, while 'OFWhiteListing' has been tested on all three platforms. We measured the round trip time of the packets in all the above mentioned scenarios. By observing the experimental results, we conclude that the two applications are capable of handling traffic isolation in real networks. Further, we conclude that 'OFWhiteListing' is more efficient than 'OFModifyVLAN.'

ACKNOWLEDGMENTS

I am greatly thankful to Dr. Byrav Ramamurthy for his invaluable guidance throughout my Masters education. It was his direction, research insights, generous encouragement, and financial support that helped and motivated me to successfully complete this project report. I feel very fortunate that I had a chance to work and learn from him throughout my Masters education.

I would like to thank Dr. David Swanson and Dr. Ziguo Zhong for serving on my Masters committee. I would like to thank Mr. Kent G. Christensen (IS, UNL), Mr. Garhan Attebury (HCC,UNL) for helping me with switch access and configurations on several occasions. I would like to thank Mr. Ivan Lai and Mr. Jim Archuleta of Ciena for helping with the firmware upgrade on the Ciena CoreDirector switch. I would like to thank my labmates Adrian Lara and Lin Liu for their valuable help with my Masters project. I would like to thank my many great friends at UNL who made my study enjoyable and lively throughout my Masters. I would like to deeply thank my parents and my sister for their love, affection, continuous support and encouragement that they gave me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of OpenFlow

A recent approach to programmable networks is the Software Defined Networking (SDN) architecture. SDN consists of decoupling the control and data planes of a network. It relies on the fact that the simplest function of a switch is to forward packets according to a set of rules. However, the rules followed by the switch to forward packets are managed by software. One motivation of SDN is to keep the design of network devices simple. Another is to perform network tasks that could not be done without additional software for each of the switching elements. Developed applications can control the switches by running on top of a network operating system, which works as an intermediate layer between the switch and the application.

OpenFlow [42] was proposed to standardize the communication between the switches and the software based controller in an SDN architecture. The authors identify that it is difficult for the networking research community to test new ideas in current hardware. This happens because the source code of the software running on the switches cannot be modified and that the network infrastructure has been "ossified"

[42], as new network ideas cannot be tested in realistic traffic settings. By identifying common features in the flow tables of the Ethernet switches, the authors provide a standardized protocol to control the flow table of a switch through software. OpenFlow provides a means to control a switch without requiring the vendors to expose the code of their devices.

OpenFlow networks have specific capabilities. For example, it is possible to control multiple switches from a single controller. It is also feasible to analyze traffic statistics using software. Forwarding information can be updated dynamically as well and different types of traffic can be abstracted and managed as flows. These capabilities have been exploited by the research community to experiment with innovative ideas and propose new applications. Ease of configuration, security, availability, network and data center virtualization and wireless applications are those that have been investigated the most using OpenFlow. They have been implemented in different environments, including virtual or real hardware networks and simulations.

OpenFlow was initially deployed in academic campus networks [42]. Today, at least seven Universities have deployed this technology [26]. The goal of OpenFlow was to provide a platform that would allow researchers to run experiments in production networks. However, industry has also embraced SDN and OpenFlow as a strategy to increase the functionality of the network while reducing costs and hardware complexity. Table 3.2 shows a list of OpenFlow compliant switches available in the market. The Open Networking Foundation (ONF) [16] was founded in 2011 by Deutsche Telekom, Facebook, Google, Microsoft, Verizon, and Yahoo to promote the implementation of SDN and OpenFlow based networks. Currently, ONF has 59 members including several major vendors.

1.2 Motivation

In today's world, a network plays a very crucial role in the operation of various businesses. In order to accommodate the ever changing requirement of the businesses, the network configurations would need to be changed dynamically. One such ever changing requirement in a network is the setup of new working groups in a business office. That is for instance as a new project comes in, certain staff in accounting department and marketing department of a business organization may need to collaborate together. The requirement of these working groups may be that all the hosts are attached to the same broadcast domain. In some situations, the communications traffic among these hosts in the working groups may be needed to be strictly isolated. In fact, in the real world situations, hosts would have to be added and removed from a particular working group dynamically as the business needs change.

One solution that is being extensively used to attain a subset of the above requirement is the concept of Virtual Local Area Network (VLAN). While using VLANs to establish working groups, many constraints come into play. Not all the switches may be supporting VLANs. Even if that is the case, the dynamic requirement to establish the working groups is highly complex. Even to just establish a working group as a first step, each of the switches in the network's subnet has to be configured separately. Further, it is very difficult to keep up with the dynamics of the working groups as the requirements change.

OpenFlow provides a centralized control of the network and more generic ways to identify flows. Hence, using the OpenFlow technology to carry out the traffic management to establish the working groups described above would be a good solution to the present problem. The motivation for the project is the scope and impact of the OpenFlow technology. The aim of the project is to develop an OpenFlow application,

evaluate its ease of development, and test its applicability in multiple environments.

1.3 Summary of Contributions

The contributions of our project can be viewed in two parts. The first part of the project constitutes a thorough study of the OpenFlow technology and hands on work with some of the currently existing OpenFlow switches and development environments. There are OpenFlow enabled software switches. An Open VSwitch is a prominent OpenFlow software switch. Mininet and OpenFlow VMWare are virtualization software that can emulate an OpenFlow capable network for testing. As part of the project, Mininet and Open VSwitch were studied and experimented upon. OpenFlow enabled hardware switches from major vendors are rapidly appearing in the market. As part of the project, HP Procurve 5406Zl switch and Ciena Coredirector switch were OpenFlow enabled and experimented upon.

The second part deals with developing OpenFlow controller applications to perform the network management of the working groups concept and evaluating the performance in various development environments. One of the applications uses the existing VLAN tags in the packets and controls their flow by adding and deleting the VLAN tags accordingly. The second application does not use VLANs, instead uses source and destination addresses in the packet to define and manage the work groups. The round trip times of the packets in multiple environments for each of the application were measured. It is observed that the initial packets take longer to be transmitted as the flow is being set up. After this the packets go through the networks at a much faster rate, making OpenFlow a viable solution for centralized network management applications.

1.4 Outline of the report

The rest of the project report is organized as follows. Chapter 2 describes the OpenFlow specification. Chapter 3 contains OpenFlow related work. Chapter 4 describes OpenFlow applications and environments in which we tested. Chapter 5 contains the description of the applications along with their evaluation. Chapter 6 concludes the project report and discusses future work.

Chapter 2

OpenFlow Specification

2.1 Overview of OpenFlow

OpenFlow [42] is a technology developed to enable a special kind of software application to dynamically control the way in which each of the switching elements in the network would function, thereby controlling the way in which the network itself has to work. The special kind of software application could be developed by anyone and is independent of the vendor specific switching elements present in the physical network on which these software applications run. In other words, OpenFlow can be viewed as a realization of the software defined networking methodology[38][21].

The OpenFlow technology constitutes of a number of components that work together to enable the working of the network to be controlled by the users of the network, such as researchers. The three main components are [42]:

- An OpenFlow switch: In the subnet of any network, there would be one or more switching elements present. Each of these switches might have been manufactured in different ways. That is, different switch vendors have different hardware and software in them. Nevertheless, for the same software application

to control them, each of these switches has to present itself in the same way to the software application. This could be achieved by each of the switches by implementing a standardized OpenFlow specification on them.

- An OpenFlow controller: The special kind of software application which controls the entire network is developed and run within an OpenFlow controller. An OpenFlow controller is a network operating system which simultaneously runs a number of these applications called as OpenFlow controller components. The OpenFlow controller is itself software running on a machine (which could be a regular personal computer). This machine must be able to communicate with the switches in the network.

- A secure channel: The OpenFlow controller and the switches have to communicate with each other. This is done through a secure channel established between a switch and the controller.

A simple OpenFlow network example is given in Figure 2.1.

2.1.1 Control plane and data plane

Any switching element in a network, be it a layer 2 ethernet switch or a multilayer switch that does switching according to the layer 3 and layer 4 headers as well, could be viewed as being comprised of two parts. They are: a control plane and a data plane.

The control plane is the software of the switching element which constitutes the logic behind the processing and forwarding of packets in the switch. In other words, from time to time, the control plane updates the data plane with information about how to process and forward a packet.

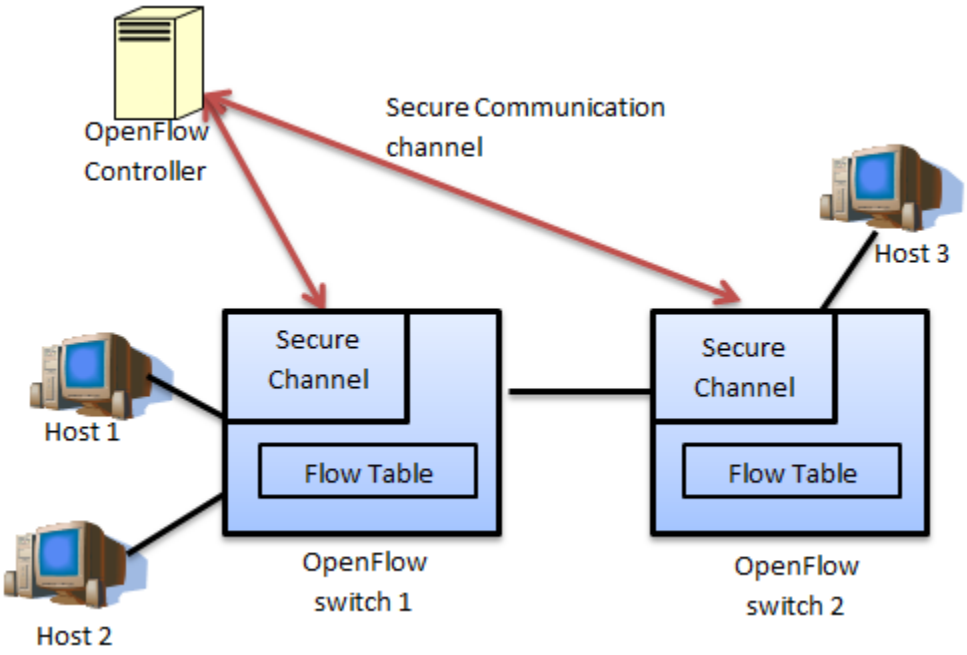


Figure 2.1: OpenFlow components.

The data plane is essentially the physical hardware of the switching element which does the processing of the incoming packets and then forwarding of packets out of the switch appropriately. With respect to an OpenFlow switch, the part of the data plane that does the processing of the packets is of interest to us, in this section.

Every normal switch has a forwarding table. In a layer 2 switch, content addressable memory (CAMs) [38] are used to store the MAC tables. In layer 3 and multilayer swithes, Ternary CAMs or TCAMs are used to store routing tables. Access control lists (ACLs), firewalls, QoS and various statistical counters are implemented by using TCAMs.

As mentioned in [42], the various networking elements in today's networks run software that is being developed by the corresponding companies. In fact the running environment is also developed by the switch companies. That means if somebody outside the switch company wants to develop a software that can run on the switch,

the interface using which to interact with it is not known. It leads to a situation where, new networking ideas could not be implemented on the production networks.

An OpenFlow switch is abstracted as an OpenFlow flow-table to the OpenFlow controller software application. So, now the question is what exactly is an OpenFlow flow-table. It consists of:

- A common set of already existing packet header fields which are being identified, such that, they are already used by most of the existing switches to perform matching. An OpenFlow flow can be defined using them.
- A set of Actions and counter variables associated with each flow in the flowtable.

Without the OpenFlow protocol running on the switch/router, the user would not have access to the switch. But with OpenFlow running, an interface to indirectly change the rules in the TCAMs according to the user's requirement is achieved.

It is clearly not required for the switch vendor to reveal anything. They just have to add a piece of code, coded according to the OpenFlow specification provided to the switch software. After which the switch will be able to communicate with a controller software application.

In the following sections, the OpenFlow switch specification is explained in further detail.

2.2 OpenFlow 1.0.0

2.2.1 Flow table: Rules, Actions, Counters

Currently the most widely deployed OpenFlow specification version is the 1.0.0 [25]. In fact the figure 2.1, which shows an example OpenFlow network, can be called an

OpenFlow 1.0.0 network. Essentially in OpenFlow specification 1.0.0, there is one OpenFlow flow-table in a switch and one controller is responsible for the switch. As mentioned before, in an OpenFlow compatible switching element, the control plane is abstracted to the user as an OpenFlow Flow table. In OpenFlow switch specification 1.0.0, this flow table constitutes of 3 segments:

- Rules
- Actions
- Counters

The following is a brief description of the three segments present in an OpenFlow flow table.

Rules are a set of header fields present in the headers of the packets. A rule for a packet to match with the flow can be defined using this set of header fields. Some examples of the header fields may be Source MAC address, TCP port number, Destination IP address, Etc. So, a sample rule for a packet to match the flow might be: All packets that have the source MAC address X would match this flow.

Actions are a set of operations that could be performed on the packets which match the Rule corresponding to the flow. An example action would be to: Forward the packet to all the ports on the switch.

Counters are the set of numbers that represent various statistics with respect to the particular flow or table or port or queue. For instance, the number of received packets corresponding to a flow is an example.

2.2.2 Packet processing in an OpenFlow 1.0.0 switch

As a summary, the figure 2.2 shows how a packet goes through the processing systems in a switching supporting Version 1.0.0.

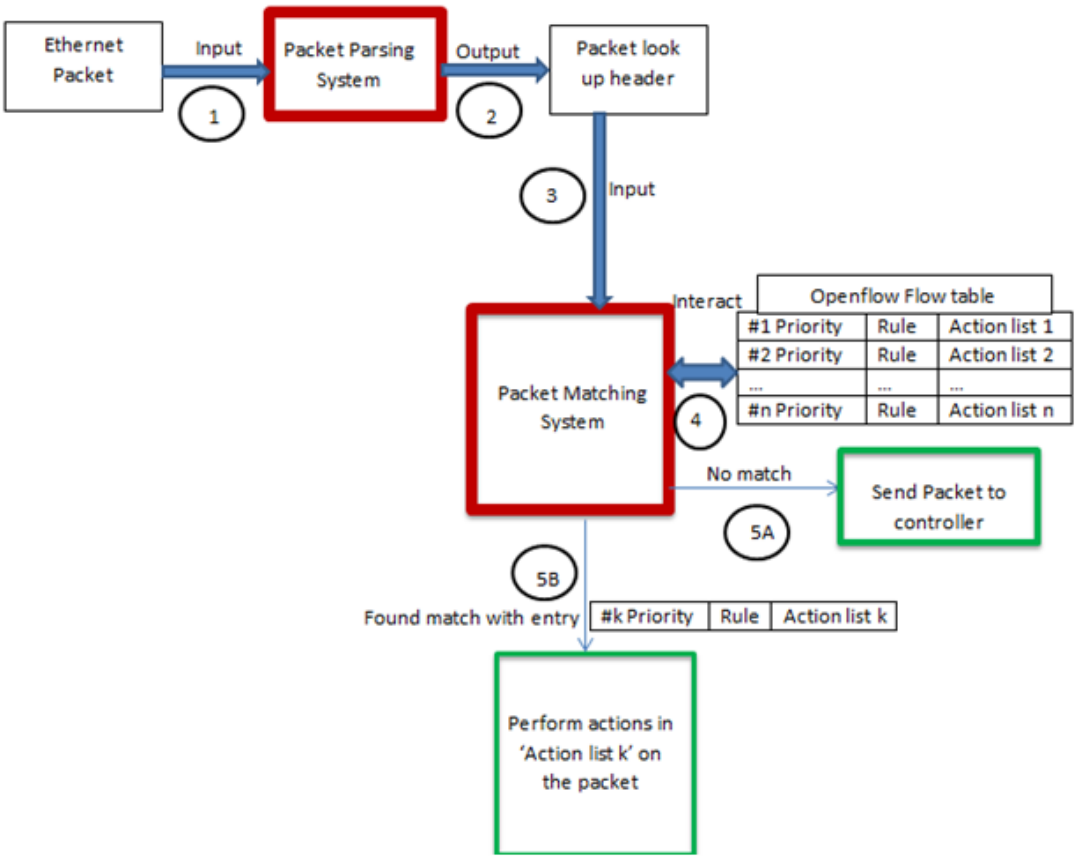


Figure 2.2: Packet processing and forwarding in an OpenFlow 1.0.0 switch

.

Figure 2.2 shows how a packet goes through the processing in a switch supporting OpenFlow specification 1.0.0. In the step 1, the Ethernet packet entering the switch goes to a packet parsing system. In the packet parsing system, the header fields present in an Ethernet packet, supported by OpenFlow specification 1.0.0 to perform matching with the flow table, are being extracted and place in a packet look up header.

The same is shown as step 2 in the above figure. In the step 3, the packet look up header generated is being sent in to a packet matching system. The packet lookup header is being compared with the rules defined for each flow entry in the OpenFlow flow table. A rule in the flow table contains the values of the header fields present in the packet lookup header in order to match with the flow entry. The interaction with the flow table is shown as step 4 in the diagram. An important point to be noted here is that, the flow entries in the table are present in the descending order of priority. So, the comparison of the packet lookup header is done starting from the first flow entry on the flow table. If a match is found in the flow table, then the packet goes through the step 5B and the action in the matched flow entry are performed on the packet. If no match is found, then the packet goes through the step 5A to the controller for processing.

2.3 OpenFlow 1.1.0

2.3.1 Basic Structure

As described in Section 2.2, according to OpenFlow specification 1.0.0, an OpenFlow switch constitutes of a single flow table, a secure channel through which it could communicate with the remote controller. The messages in the communication are defined by the OpenFlow protocol.

In the OpenFlow specification 1.1.0 [28], the switch constituents have changed significantly. In the OpenFlow switch specification 1.1.0, there are multiple flow tables present in the switch. And there is a Group table present in the switch in addition to the multiple flow tables. Figure 2.3 shows an OpenFlow 1.1.0 switch. Although, the secure channel to communicate with the controller and the OpenFlow

protocol are not very different from the specification 1.0.0. But, additional features have been added to the OpenFlow protocol to accommodate the changes in the flow table structure of the switch.

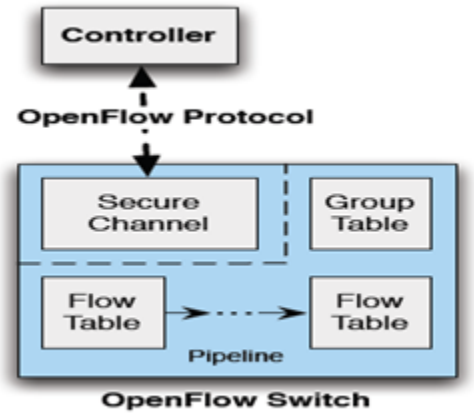


Figure 2.3: OpenFlow Switch 1.1.0

The packet processing of the packet entering the switch has changed as there are multiple flow tables available in the switch. The flow tables in the switch are linked to each other through a process termed as "pipeline processing".

Pipeline processing involves a set of flow tables linked together to process the packet coming in. When the packet fist enters the switch, it enters a Table 0. It is then sent to the first table to look for the flow entry to be matched. If there is a match, the packet gets processed there and if there is another table that the particular flow entry points to, the packet is then next sent to that flow table. This happens until a particular flow entry does not point to any other flow table.

The flow entries in the flow tables can also point to the group table in addition to the ability to point to the next table. The group table is a special kind of table designed to perform operations that are common across multiple flow. That is,

actions pertaining to a set of flows are grouped together. Further, the set of flows are controlled to perform various actions collectively under a single group. Complex forwarding actions such as multipath, link aggregation are enabled.

Another important concept that is present in the specification 1.1.0 is the meta-data field that is used to pass information between the table as the packet traverses through them.

2.3.2 Packet Matching

In this subsection, the matching performed on a packet in an OpenFlow 1.0.0 switch and 1.1.0 switch are being explained to illustrate the significant differences between the two.

According to the OpenFlow specification 1.0.0, the packet matching takes place as described below:

- The incoming packet is being parsed.
- As there is a single flow table, each flow entry in the flow table is being traversed until a match for the parsed packet is found. And then the actions corresponding to the matched flow entry are performed on the packet.
- If no match, the packet is forwarded to the controller.

And according to the OpenFlow specification 1.1.0, the packet matching takes place as described below :

- When the packet enters the switch, the packet parsing is the same as in 1.0.
- The matching process starts from the table 0 (the numbering of the tables starts from 0 to n). If a match is found, the set of instructions are executed.

As explained earlier, the instructions would process the packet as desired and pass on to the next table after updating the actions set and the metadata field.

- If in a particular table, no match is found, according to the switch configuration, three options are available: Packet can be forwarded to controller, Packet can be dropped, Packet can be forwarded to the next table.

- If in a particular table, there are no processing instructions to perform pipelining by pointing to the next table, the actions to be performed on the packet for that table are performed and the processing of the packet in the switch is complete.

By having a look at the matching process in OpenFlow specification 1.0.0 and 1.1.0, it is clear as to how the structural differences in the switch with respect to OpenFlow specification 1.0.0 and 1.1.0 are accommodated into the matching process.

2.4 Evolution of specification 1.2

In December 2011, the Open Networking Foundation has issued the OpenFlow specification 1.2 [29]. There have been few major features that have been added to the 1.2 specification compared to the specification 1.1. In specification 1.2, IPv6 addressing is being supported. Matching could be done using the IPv6 source and destination addresses. Another important feature supported is the possibility of connecting to multiple controllers concurrently. The switch maintains connections with all the controllers it is configured to connect to and the controllers communicate with themselves to do hand overs among themselves. The multiple controllers provide fast recovery during failure of a controller and also intend to provide load balancing.

2.5 OpenFlow as a specification, protocol, architecture

OpenFlow has been viewed as a specification, a protocol, architecture in various contexts.

OpenFlow can be viewed as a specification when it is in the context of an OpenFlow switch. The abstraction in an OpenFlow switch is achieved by implementing the requirements specified in the OpenFlow specification to make it an OpenFlow switch. For instance, in the OpenFlow specification, it is specified that the switch has to support the FLOOD action on the packets entering the switch. The internal implementation of this requirement is up to the vendor of the switch. But, the functionality has to be provided in order to be called as an OpenFlow switch.

The OpenFlow protocol is the part that deals with defining the format of the messages passed between the controller and the OpenFlow switch. The format of the messages has to be understood as well as generated by both the entities. This standard format of message passing is defined by the OpenFlow protocol. In fact, the OpenFlow protocol is part of the OpenFlow specification. It applies to the OpenFlow controller along with the OpenFlow switch.

OpenFlow is viewed as architecture when viewed in the context of an entire network. In the network where OpenFlow switches are being controlled by OpenFlow controller, such a network can be viewed as supporting OpenFlow architecture.

Chapter 3

OpenFlow Components and Related Work

3.1 Overview of implementing applications using OpenFlow

In order to run applications on top of a controller to manipulate the flow table of a switch, a network operating system is required (See Fig. 2.1). It acts as an intermediate layer between the OpenFlow switch and the user application. The network operating system communicates with the switch using the OpenFlow protocol and notifies the application of network events. Nox [35], Beacon [32] and Maestro [50] are examples of network operating systems. Recently, Big Switch released Floodlight [3], an open source Java based controller. Foster et al. [33] proposed Frenetic, a network programming language that simplifies the development of applications on top of network operating systems. Table 3.1 summarizes comparative data for some OpenFlow controllers.For more detailed description, we refer the reader to [39], our survey paper

(under review).

There are at least three possibilities to implement OpenFlow based applications. First, an OpenFlow compliant hardware switch can be used. It is also possible to implement an OpenFlow compliant software based switch using Open vSwitch [45, 18]. Finally, a third option is to deploy virtual networks using Mininet [38].

Using OpenFlow, experimental and production traffic can share the same OpenFlow switch. The action of a flow table entry of an OpenFlow switch can be to send the packet to the switch data path. On the other hand, a different flow entry can be defined for experimental traffic. This way, experimental traffic can be tested without interfering with the production traffic [42]. In order to further enhance this, Sherwood et al. proposed FlowVisor [47]. Using this technique, it is possible for several controllers to share the control of a switch. A centralized OpenFlow based controller "slices" the network and acts as an intermediate layer between the switch and all the OpenFlow controllers that manipulate the switch.

3.2 OpenFlow compliant switches

A number of commercial switch vendors have developed switches that support OpenFlow protocol. A non-exhaustive list of hardware switch vendors are Arista, Ciena, Cisco, Juniper, HP, NEC, Pronto, Toroki, Quanta. In addition to the hardware switches, Open VSwitch is an OpenFlow software switch which runs on linux operating system. Table 3.2 provides more details on some of the prominent OpenFlow switch vendors such as OpenFlow compatible series, the capability of the switch.

Table 3.1: OpenFlow controllers.

Controller	**L**anguage	**C**reated by	**C**omments
NOX	C++	Nicira Networks	NOX was donated to the research community in 2008. It has several branches at Stanford University, such as classic NOX, new NOX and POX. New NOX is the version that will be further developed. POX supports Python and it is used for educational or research applications [14].
Beacon	Java	Stanford University	Supports both event-based and threaded operation. Mostly used for research and experimentation [32].
Maestro	Java	Rice University	Licensed under licensed under LGPL v2.1. Not as common as other controllers such as NOX [11].
Floodlight	Java	Big Switch Networks	Forked from Beacon and extended for enterprise usage. Apache-licensed [3].

3.3 OpenFlow based appliations

OpenFlow has been used to provide ease of configuration, security and availability. It has also been used to achieve network and data center virtualization. We also describe some wireless applications and others.

3.3.1 Ease of configuration

OpenFlow based applications can simplify the configuration of the network. Common approaches include access control lists and complicated configuration files. By using SDN, it is possible to use software to take care of this. Yamasaki et al. [48] proposed using OpenFlow to manage the VLANs of a campus network. They describe how the

Table 3.2: OpenFlow Switches

Switch Company	**S**eries	**N**otes
Arista	Arista extensible modular operating system (EOS), Arista 7124FX application switch	Gigabit Ethernet switching through 24 10-gigabit ports; software solution for SDN.
Ciena	Ciena Coredirector running firmware version 6.1.1	A special firmware version 6.1.1 supporting OpenFlow has been developed for OpenFlow experimental demos.
Cisco	Cisco cat6k, catalyst 3750, 6500 series	Prototypes used in OpenFlow demos.
Juniper	Juniper MX-240, T-640	Prototypes used in OpenFlow demos.
HP	HP procurve series- 5400 zl, 8200 zl, 6200 yl, 3500 yl, 6600	Gigabit Ethernet switching through 288 gigabit ports or 48 10-gigabit ports. Deployed in several places including Stanford
NEC	NEC IP8800	Gigabit Ethernet switching through 48/24 gigabit ports + 2 10-gigabit ports. Supports multiple OpenFlow switch instances. The switches are being used in various OpenFlow deployments including ESNet ANI test bed, Stanford.
OpenvSwitch	Latest version: 1.4.1	Software switch supporting OpenFlow version 1.0
Pronto	Pronto 3240, 3290	Bare switch with software support from Stanford and Toroki. Pronto 3290 is used in ORBIT testbed deployment.
Toroki	Toroki Lightswitch 4810	Gigabit Ethernet switching through 48 gigabit ports + 4 10-gigabit ports
Quanta	Quanta LB4G	Gigabit Ethernet switching through 48 gigabit ports

number of VLAN ids is limited and how the configuration tasks are tedious. In their approach, the controller analyzes incoming traffic and detects if the communication should be allowed or not, based on virtual group ids (GID) instead of VLANs. Using this approach, the number of VLANs limitation is overcome and the configuration of the network is simplified.

Reitblatt et al. [46] describe how updating network policies can lead to inconsistencies when packets are processed by both the old and the new policy. The authors note that achieving per-packet and per-flow consistency is critical to avoid inconsistencies and they describe techniques to implement both features.

Casado et al. [30] proposed Ethane , an SDN architecture explicitly designed to simplify the management of the network in an enterprise. Ethane relies on the idea that the network policy should be known by the controller and enforced in all switches. The main requirement is that all communications between two hosts require explicit permission. Instead of creating configuration files for all the switches in the network, these devices are kept simple and the rules are managed by the controller. An implementation of an Ethane switch in hardware is described in [36].

3.3.2 Other applications

OpenFlow has also been used in other areas not listed above, such as routing and network congestion control. Liu et al. [41] proposed a method to control congestion using queuing systems and a centrally controlled network. Yap et al. [49] also consider network congestion, as well as bandwidth reservation and multicast. Nascimento et al. [43] proposed QuaqFlow, a Quagga implementation using OpenFlow. Quagga is a routing package that provides implementation of TCP/IP routing protocols. RouteFlow [44], an architecture that provides routing as a service, was proposed as an

extended work of Quagga. Egilmez et al. [31] proposed an architecture to provide routing for video streaming.

3.4 Deployments

Deployments of OpenFlow based networks mainly include campus networks and test beds, as well as deployments undertaken by the industry.

Stanford University has deployed an OpenFlow based network in one of its buildings. The network includes production, experimental and demonstration traffic. It connects approximately fifty switches and around 25 users, both wired and wireless. Details of the topology can be found at [27]. Other universities have also deployed OpenFlow based networks. The full list is available at [26] and it includes Clemson University [2], Georgia Tech [5], Indiana University [6], Kansas State University [10], Rutgers University [22], University of Washington [23], University of Wisconsin [24] and Princeton University [26].

At a larger scale, the Global Environment for Network Innovations (GENI) [4] provides a research infrastructure where OpenFlow experiments can be conducted. The OpenFlow core of this network consists of several interconnected OpenFlow compliant switches on both Internet2 [7] and National LambdaRail (NLR) [13] networks. The connection to the NLR network is achieved through HP6600 switches deployed at Sunnyvale, Seattle, Denver, Chicago, and Atlanta and through NetFPGA switches in Sunnyvale, Houston, Chicago, and New York [37]. Internet2 has OpenFlow compliant switches installed in Los Angeles, New York, Washington DC, Atlanta [34]. Campus networks can connect to the GENI deployment to run larger scale experiments.

The Energy Science Network (ESNet) based in Berkeley Lab has also deployed an OpenFlow test bed as part of the Advanced Networking Initiative (ANI). As

stated in [1], the ESnet ANI is an investment in next-generation technology infrastructure to speed scientific discovery. It has two test beds: Long Island Metropolitan Area Network (LIMAN) and 100G. The LIMAN test bed includes four NEC IP8800 programmable flow OpenFlow switches[8]. The OpenFlow network operates on the VLAN 101. There are two ways of running an experiment on the test bed. One option is to connect the controller directly to the OpenFlow switches through the management VLAN. The second option is to connect to the flow visor controller and getting a partition of the network to run the experiments. The first option requires the researches to reserve the test bed beforehand. The second option does not require any reservation of resources. The flow visor configuration file has to be sent to the administrator to get connected. Figure 3.1 shows the topology of the ANI OpenFlow test bed.

Another smaller deployment is the Open Access Research Testbed for Next-Generation Wireless Networks (ORBIT) test bed [19], which is being developed and operated by WINLAB, Rutgers University. It is intended to be used to test and evaluate innovative protocols in real-world settings and it includes an OpenFlow based network. The deployment consists of an OpenFlow compliant switch Pronto 3290 connected to nine nodes. Out of the 9 nodes, 7 of them are connected to one NetFPGA each. Each of the NetFPGA is connected to the Pronto 3290 OpenFlow switch through four 1GbE connections. All of the 9 nodes are connected to the Pronto 3290 OpenFlow switch and they are connected to a control plane through which the nodes can be accessed through telnet/ssh sessions by the experimenter. Figure 3.2 shows the topology of the ORBIT OpenFlow test bed.

Similar test beds have been deployed in Europe and Japan as well. Ofelia is a project funded by the European Union that provides an OpenFlow based network with nodes in Belgium, Switzerland, UK and Spain [15]. Also, the Dynamic Network

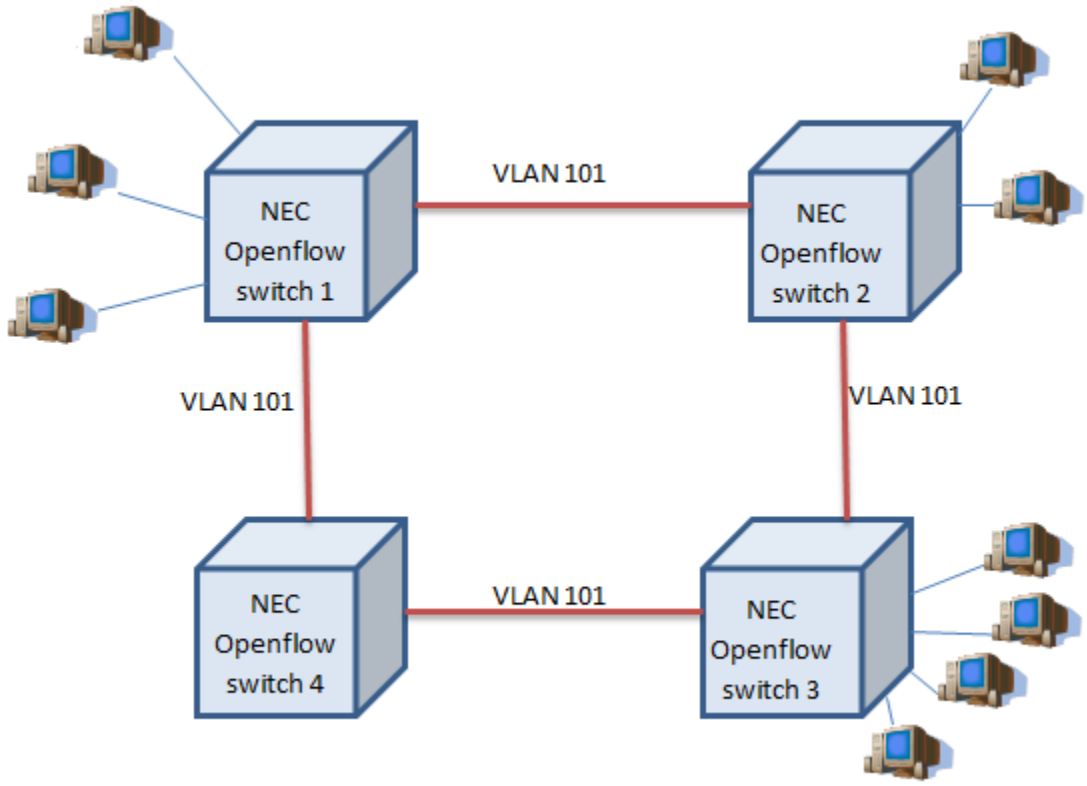


Figure 3.1: Topology of ANI OpenFlow Testbed.

System (DYNES) project [12], funded by the National Science Foundation (NSF), is exploring technologies such as OpenFlow to interconnect campus, regional and backbone networks. Other future deployments also include the Network Development and Deployment Initiative (NDDI) and the Open Science, Scholarship and Services Exchange (OS^{3}E) [12]. OpenFlow has also been deployed by several companies, as seen in the keynote lectures of the 2012 Open Networking Summit [17]. As an example, Google has deployed OpenFlow in the inter-datacenter backbone network that carries all the traffic between the different datacenters. Currently, this network is completely OpenFlow based. According to the speaker, adopting OpenFlow has been the most significant change in networking in the company [40].

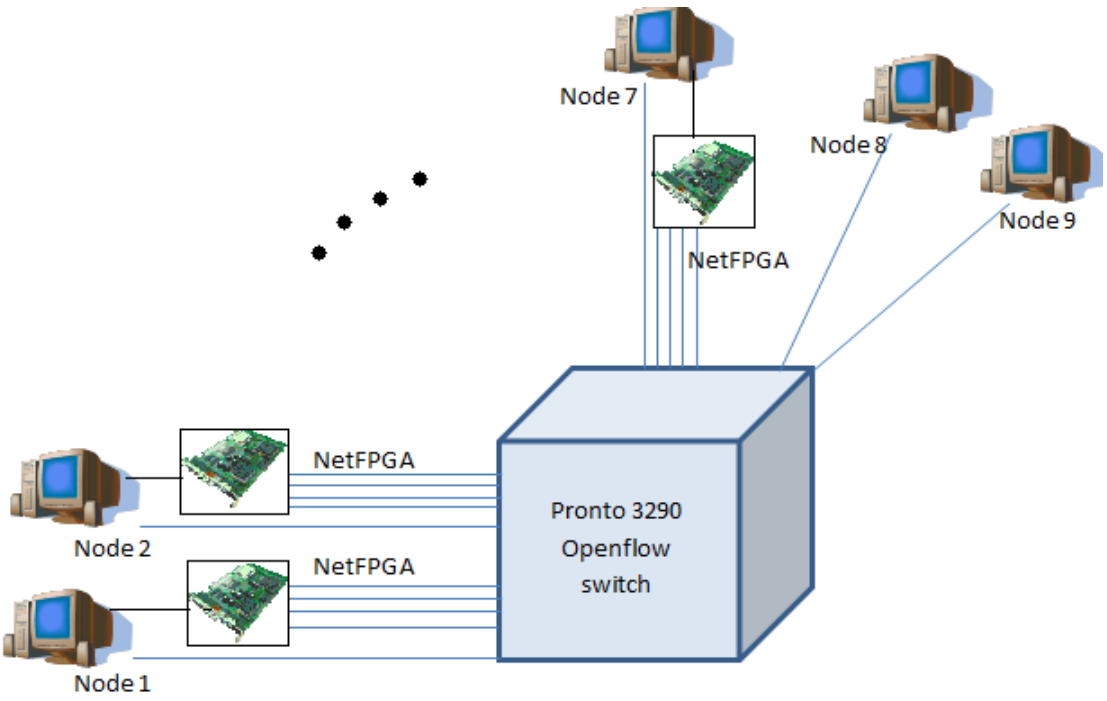


Figure 3.2: Topology of ORBIT OpenFlow Testbed.

Chapter 4

Our OpenFlow Applications and Platforms

4.1 Problem Definition

In this section, the problem addressed by the OpenFlow applications developed by us are being discussed. Given a network, isolating the traffic among multiple hosts is being addressed by the OpenFlow applications. This is achieved by grouping together the hosts in the networks that are supposed to communicate with each other. The groups are created in order to enable the group of host to work together. Each host in the network may be part of multiple work groups.

To elaborate on the problem addressed, let us consider the example network show in the Figure 4.1. The network constitutes of a 2 switch subnet with 4 hosts connected to each switch.

The desired basic functionality is that, given this network, we need to be able to set up a working group grp1 with say hosts A, C, D in it. As the situation become more complex, a new requirement comes in where F has to be added to the gr1 on

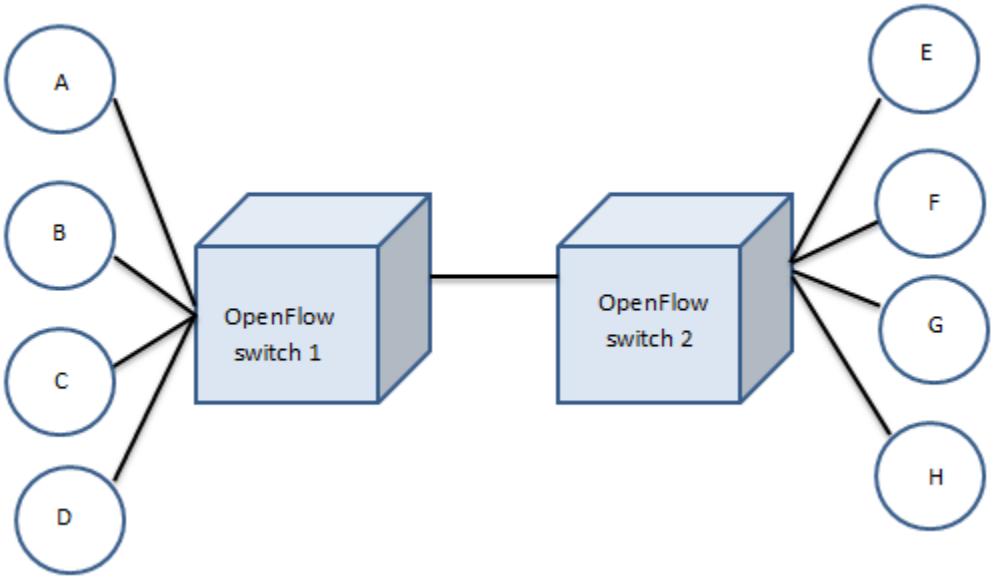


Figure 4.1: Example Network

07/16/2012 from 9AM to 3PM. Another requirement may be that the grp1 is no longer needed and so needs to be deleted. OpenFlow can be used to dynamically manage the network where such traffic isolation and management is needed.

Two approaches to solve this problem were developed and tested for compatibility across multiple switching platforms. The first application assumes that the network is VLAN capable. The existing VLAN capability of the network is being used to centrally manage the network through an OpenFlow controller. The second application does not assume any VLAN capability in the network and addresses the problem by using data structures in the centralized OpenFlow controller.

4.2 Beacon OpenFlow Controller

In order to develop OpenFlow applications, there are multiple controller choices available. NOX, Beacon, Maestro, Floodlight are some of the popularly used controllers.

In this project Beacon controller was used to develop the OpenFlow controller applications.

Beacon is a Java based OpenFlow controller. Availability to develop Beacon controller applications through integrating with Eclipse IDE makes the development of new applications convenient. The applications in Beacon are developed using Java Spring framework [9]. The Spring framework enables modular development of components that can be reused. In conjunction with Spring frame work to develop modular components, Open Services Gateway initiative (OSGi) framework[20] is used to make the modular components as services. By doing so, the components can be used by each other while running simultaneously as services. In OSGi framework the modular components are called bundles. In Beacon controller, each of the controller application is developed as a bundle. Each of the bundles can register themselves as services. So, other bundles can use a bundle registered as a service in their execution.

Beacon OpenFlow controller has been developed using the Spring and OSGi frameworks. An API is being published by the Beacon developers to develop new OpenFlow controller applications. The API has been developed by strictly following the OpenFlow specification. The OpenFlow applications described below are developed using the API and framework provided by Beacon OpenFlow controller.

4.3 OFModifyVLAN Application

In this section, we describe the first application that uses the existing VLAN support of hardware in the network. This OpenFlow application approaches to solve the problem described in the section 4.1 by modifying the VLAN tags present in the appropriate packets.

The network is assumed to be in a particular default configuration before the

new dynamic requirements come in. The default configuration constitutes of a setup, where in the work groups are initially setup by assigning appropriate VLAN between the hosts. But as pointed out in the initial chapters, VLANs are not very suitable when the requirements of the network change very quickly. The network management task to isolate the traffic through VLANs is very tedious. To constantly change the VLAN configurations to cope with the dynamics is not easy. So, our OpenFlow controller application manages the network which is initially running on a default configuration. In other words, as the new requirements come in, this OpenFlow controller application makes adjustments to the current default flows in the network to accommodate the new requirements. The whole network management by the controller is carried out centrally.

Algorithm 4.1 shows the step-by-step processing of the packet as it enters the OpenFlow controller

Algorithm 4.1:

Input: Packet, SourceIP, DestinationIP, vlanID, Action

Output: Appropriate OpenFlow flow entry modifications to the subnet switches.

Step 1: Controller waits for packet arrival. If packet arrives, goto step 2

Step 2: Load the packet headers into 'match' data structure

Step 3:if network source IP address in 'match' = 'SourceIP' & network destination IP address in 'match' = 'DestinationIP'

Step 3.1:if 'Action' is AddToVLAN

Step 3.1.1: Create a flow entry 'FEntry' with rules = match, actions = OF Action Modify Virtual LAN Indentifier to 'vlanID' + OF Action Output to port OFPP_NORMAL

Step 3.2: if 'Action' is DeleteFromVLAN

Step 3.2.1: If VLAN ID in 'match' = 'vlanID'

Step 3.2.1.1: Create a flow entry 'FEntry' with rules = match, actions = OF Action Strip Virtual LAN Indentifier + OF Action Output to port OFPP_NORMAL

Step 3.2.2: Else goto step 4.1

Step 4: Else **Step 4.1:** Create a flow entry 'FEntry' with rules = match, actions = OF Action Output to port OFPP_NORMAL

Step 5: For all Switches in the network of the subnet

Step 5.1: Insert the flow entry 'FEntry' in to the switche's OpenFlow flow table

Step 6: Output the packet through the incoming port

Step 7: Goto step 1

In Algorithm 4.1, the inputs are the packet, the source and destination IP address that should match with the packet and Action that determines the flow to be entered. The Action could indicate that a packet coming from a source IP x and destination IP y should not belong to VLAN z. Or the Action can indicate that a packet coming from a source IP x and destination IP y should belong to VLAN z. If the packet coming in does not have match, then it just goes through the normal processing of the switch. OFPP_NORMAL is an OpenFlow specification defined OpenFlow port. It means the packet will be sent through the normal processing of the switch. According to this input, the OpenFlow flow entries are entered on to all the switches in the network. The next packet matching the flow entry would not go to the controller any more.The time complexity of the algorithm is in the order of O(n) where n is the number of switches connected to the OpenFlow controller.

This application is very useful to handle a situation where in requirements change during a short duration. That is, given a network with certain VLANs setup, temporary modification (such as removing a particular host from the VLAN or adding a host temporarily into the VLAN although it is not part of the VLAN) can be done very easily. But, as the flow entries involve adding and deleting VLAN ID from a packet, it slows the packet transit. For long term changes it is not very preferable. Also, this application requires that the OpenFlow switches also support normal Layer 2 and 3 processing, which is the case with most of the networks today. But, it is a limitation. All the above limitations are addressed by the OFWhiteListing application which does not use VLANs.

4.4 OFWhiteListing Application

This application uses a data structure in the OpenFlow controller application to manage the working groups in a network. While using this application, it is assumed that there is no initial default configuration setup. So, when the OpenFlow controller takes charge, it has to make sure the communication links between the hosts are explicitly setup. That is, the switches in the networks subnet do not automatically function according to their layer 2 and 3 functionalities. This application takes advantage of this scenario to control the network. As the work groups are formed, the links between the hosts in the network are white listed. That is, flow entries are entered to enable communication.

The data structure in the application that holds a particular working group is a list of IP addresses. Each of these work groups are maintained in a list called 'listOfGroups'. In a similar way, there are data structures that bind each work group with ID, start time and end time.

With these data structures, the application is implemented according to Algorithm 4.2. The start time and end time are used to handle the expiry time of the flow entry being inserted into the switches. The above algorithm 4.2 does not show it. Also a user interface has been developed to enable the user to Add, Modify, Delete work groups.

Algorithm 4.2:

Input: Packet

Output: Appropriate OpenFlow flow entry modifications to the subnet switches.

Step 1: Controller waits for packet arrival. If packet arrives, goto step 2

Step 2: Load the packet headers into 'match' data structure

Step 3: Get network source IP address in 'match' into a varibale 'SourceIP'

Step 4: Get network destination IP address in 'match' into a variable 'DestinationIP'

Step 5: For each work group in the listOfGroups

Step 5.1: If 'SourceIP' is in the current work group

Step 5.1.1: Get the 'SourceGroupID', 'startTime', 'endTime' of the work group

Step 5.2: Else goto setp 5

Step 6: For each work group in the listOfGroups

Step 6.1: If 'DestinationIP' is in the current work group

Step 6.1.1: Get the 'DestinationGroupID', 'startTime', 'endTime' of the work group

Step 6.2: Else goto step 6

Step 7: if SourceGroupID not null and SourceGroupID = DestinationGroupID

Step 7.1: Create a flow entry 'FEntry' with rules = match, actions = OF Action Output to port OFPP_NORMAL

Step 8: For all Switches in the network of the subnet

Step 8.1: Insert the flow entry 'FEntry' in to the switche's OpenFlow flow table.

Step 9: Output the packet through the incoming port

Step 10: Goto step 1

The time complexity of the algorithm is in the order of O(mn) where m is the number of WorkGroups in the network and n is the number of hosts in the network.

The following are the differences between the controller applications 'OFModifyVLAN' and 'OFWhiteListing'. The first difference is the usage of the OpenFlow action in the each of the applications. The applications 'OFModifyVLAN' uses the existing VLAN feature in the network and hence uses the OpenFlow actions modify VLAN and strip VLAN. In the case of 'OFWhiteListing', no VLAN features are used. Instead the links between the hosts in the network are being enabled when a communication is desired. So, OpenFlow action forward is being used in the 'OFWhiteListing' application.

The next major difference between the two applications is the initial network configuration. In the application 'OFModifyVLAN', the network is assumed to be in an initial default configuration. That is, VLANs that are required to be available for the long term are already setup. The 'OFModifyVLAN' application makes adjustments to this configuration to accommodate the temporary requirement changes. Whereas, in the case of OFWhiteListing, there is no initial default configuration in the network. The application takes care of allowing communication between the hosts. If requirement changes are intended to be long term, 'OFWhiteListing' is more suitable to use. Although, 'OFModifyVLAN' integrates easily with the VLAN features already in use.

4.5 Network Platforms for testing

4.5.1 Mininet

Mininet is software that runs on a single laptop and can be used to prototype large networks. In other words, it is a platform that is capable of hosting large virtual networks on a single laptop. Mininet has been developed to provide an inexpensive virtual network test bed to experiment with new ideas such as new network architecture, or a new address scheme, or a new mobility protocol etc. It is ideal to test new ideas on virtual environment enabled by software such as Mininet, before transferring them on to real hardware. Mininet has been developed to provide an environment to perform research in the area of Software Defined Networking. More specifically, it can used to very conveniently bring up OpenFlow enabled network to test new OpenFlow applications. Mininet has a set of commands that are used to set up a network with certain number of OpenFlow enabled switches and hosts attached to them. External OpenFlow controllers such as NOX, Beacon, Floodlight etc. could be used in conjunction with Mininet to totally emulate a real world situation where a network comprising of OpenFlow enabled switches is controlled by an OpenFlow controller.

The working of Mininet is documented in [38]. Briefly, Minnet creates a network of Virtual Machines (VMs) on a single Linux Operating system. It brings up each VM as a process in the OS. It uses the support of network namespace provided in the Linux kernels. It places a VM under a particular network namespace and connects the network namespaces with virtual Ethernet feature available in Linux kernel. In this manner, Mininet claims to bring up a network of hundreds of switches. Mininet is under extensive development to fine tune itself as a flexible virtual network platform to perform OpenFlow experiments. It provides features to bring up custom topologies

although it also provides direct commands to bring up parameterized topologies. It also provides a Python API to create networks as per the requirement of the experiment.

4.5.2 Open VSwitch

In order to experiment with OpenFlow, a step ahead of virtual network platforms such as Mininet, is an Open VSwitch. It is an OpenFlow enabled software switch. It is a production quality multilayer software switch. It is most suited to be a virtual switch in virtual environments such as virtualized servers. Like a hardware multilayer switch, it can be accessed through a standard management interface to perform several configurations. It is a common practice to perform virtualization of servers to support multiple functionalities on a single physical server. An Open VSwitch is designed to provide visibility interfaces to such virtualized physical servers. Virtualization of the resources is done by several virtualization software. Open VSwitch supports Linux-based virtualization technologies such as Xen/XenServer, KVM, and VirtualBox. As mentioned in [18], the following are some of the most useful features in an Open VSwitch:

- Standard 802.1Q VLAN model with trunk and access ports
- NetFlow, sFlow(R), and mirroring for increased visibility
- QoS (Quality of Service) configuration, plus policing
- Generic Routing Encapsulation (GRE), GRE over IPSEC, and CAPWAP tunneling
- OpenFlow 1.0 plus numerous extensions

- Transactional configuration database with C and Python bindings
- Compatibility layer for Linux bridging code
- High-performance forwarding using a Linux kernel module

Also, following are some of the important components in an Open VSwitch as mentioned in [18]:

- 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-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-controller, a simple OpenFlow controller.
- ovs-ofctl, a utility for querying and controlling OpenFlow switches and controllers.

The latest version of Open VSwitch is 1.5.0. From Linux kernel version 3.3 on, the Open VSwitch module is part of the kernel. Open VSwitch can also operate, at a cost of performance, entirely in userspace, without assistance from a kernel module. To start experimenting with OpenFlow on a small scale test bed, Open VSwitch is a good option. Each of the regular Linux box should be configured as an Open VSwitch and a Linux virtualization software such as KVM could be used in conjunction with Open VSwitch to bring up multiple virtual machine (VM) hosts on the linux boxes configured as Open VSwitches. An external OpenFlow controller could be configured

on each of the Open VSwitches using the ovs-vsctl utility. Open VSwitch provides a good way to experiment with server virtualizations using OpenFlow.

4.5.3 HP Procurve

The Procurve series of switches from Hewlett-Packard are OpenFlow enabled. The HP Procurve switches are designed to be next-generation Layer 2, 3 intelligent switches. They are designed to meet the adaptive intelligence, versatility, and operational excellence to meet current and future networking demands. The HP Procurve switch series is OpenFlow enable with many additional features.

The OpenFlow module is included in the HP switch software for the HP Switch 8200zl, 6600, 6200zl, 5400zl, and 3500/3500yl products. The current OpenFlow module has been implemented in switch software revision K.15.06.5008 for those switches. Software revision K.15.06.5008 implements OpenFlow protocol version 1.0. An external OpenFlow controller is configured with the switch in ordered to be operated in OpenFlow mode.

4.5.4 Ciena Coredirector

The CoreDirector CI Switch can deliver a wide range of optical capacities, along with Ethernet switching capabilities. The switch supports SONET as well as SDH interfaces, specifically, OC-3/12/STM-1/4, OC-48/STM-16, OC-192/STM-64 optical interfaces, STM-1e electrical interfaces and Gigabit Ethernet interfaces. They provide nonblocking, bidirectional switching capacity that can be configured to switch and groom traffic from any input port to any output port down to the STS-1/VC-3 level.

For the purpose of our research, OpenFlow enabled firmware has been deployed on the Ciena Coredirector. As this is an hybrid optical switch, the above two mentioned

applications could not be tested on this platform. The firmware is an experimental unsupported version. Appendex A gives more technical details about the procedure of an OpenFlow firmware update on the Ciena Coredirector at UNL.

Chapter 5

Experimental Results

In this section we present the results obtained by running OpenFlow applications described in Chapter 4 on multiple network platforms. The applications have been tested for compatibility of the same OpenFlow application on multiple network platforms. The platforms used in the project are Mininet, Open VSwitch and HP Procurve 5046 Zl switch. We measured the round trip time of the packets to determine the performance.

5.1 Mininet

The Mininet network emulation platform was used to bring up the networks shown in Figures 5.1 and 5.3. A pre-packaged Ubuntu Virtual machine (VM) with modified kernel to support OpenFlow binaries, some tools for measurements and many more modifications to support emulation of large network, is available to experiment with OpenFlow. The Mininet VM was downloaded and run on a PC which the operating system Ubuntu 11.10 running on it. Intel Core duo processor with 4GB RAM was used in the PC.

5.1.1 Mininet Network Configuration 1

Figure 5.1 shows a network where in a single OpenFlow switch connected to 4 host machines is brought up on the Mininet VM. The OpenFlow switch is connected to a Beacon OpenFlow controller which runs the controller applications developed. The hosts labeled A and B are programmed to be work group wg1. And the hosts labeled C and D constitute the work group wg2. So, A cannot communicate with C initially.

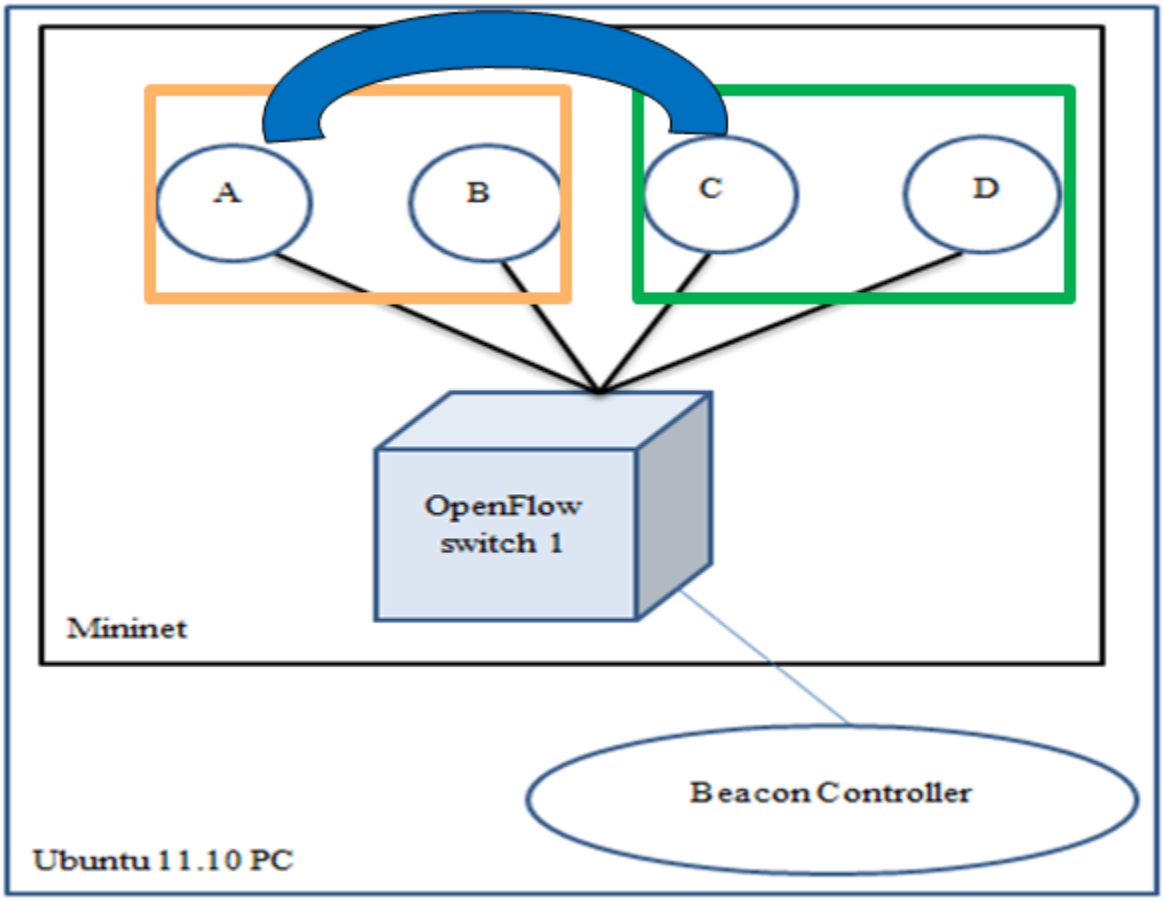
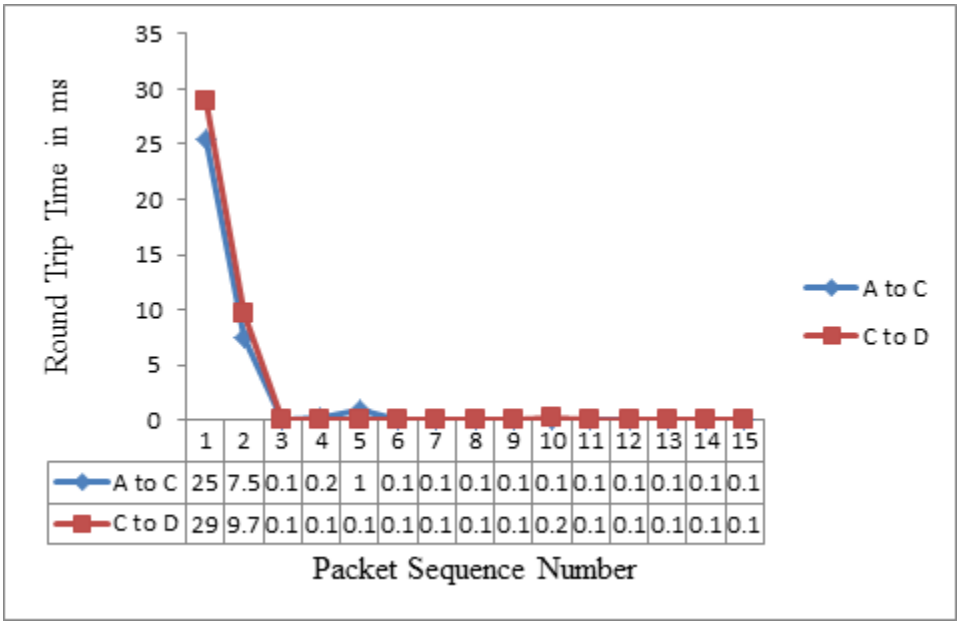


Figure 5.1: Minimet Configuration 1

The OpenFlow controller Application OFWhiteListing described in Chapter 4 is used to generate the following result. The host A is enabled to communicate with C for duration of two hours. Initially, the hosts A and C belong to two different

work groups. The graph in Figure 5.2 shows the round trip time of packets set from host A to host C and host C to host D. The communication between A to C is inter workgroup, whereas, the communication between C to D is within the same group. Despite that we observe that the round tip time of the packets stabilizes in both cases at around 0.1ms, after the first packet taking an average of 27ms. The X axis shows the sequence number of the packet. The Y-axis shows the round trip time of the packet in milliseconds.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A to C	25	7.5	0.1	0.2	1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
C to D	29	9.7	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.1	0.1

Figure 5.2: Mininet network configuration 1 running Application OFWhiteListing

.

5.1.2 Mininet Network Configuration 2

The second configuration used to test the Application OFWhiteListing on Mininet is shown in Figure 5.3 below. In this case two switches with four hosts attached to each of them are brought up using Mininet. The initial configuration goes as follows:

hosts A, B belong to work group wg1; hosts C,D belong to work group wg2; hosts E,F,G,H belong to work group wg3.

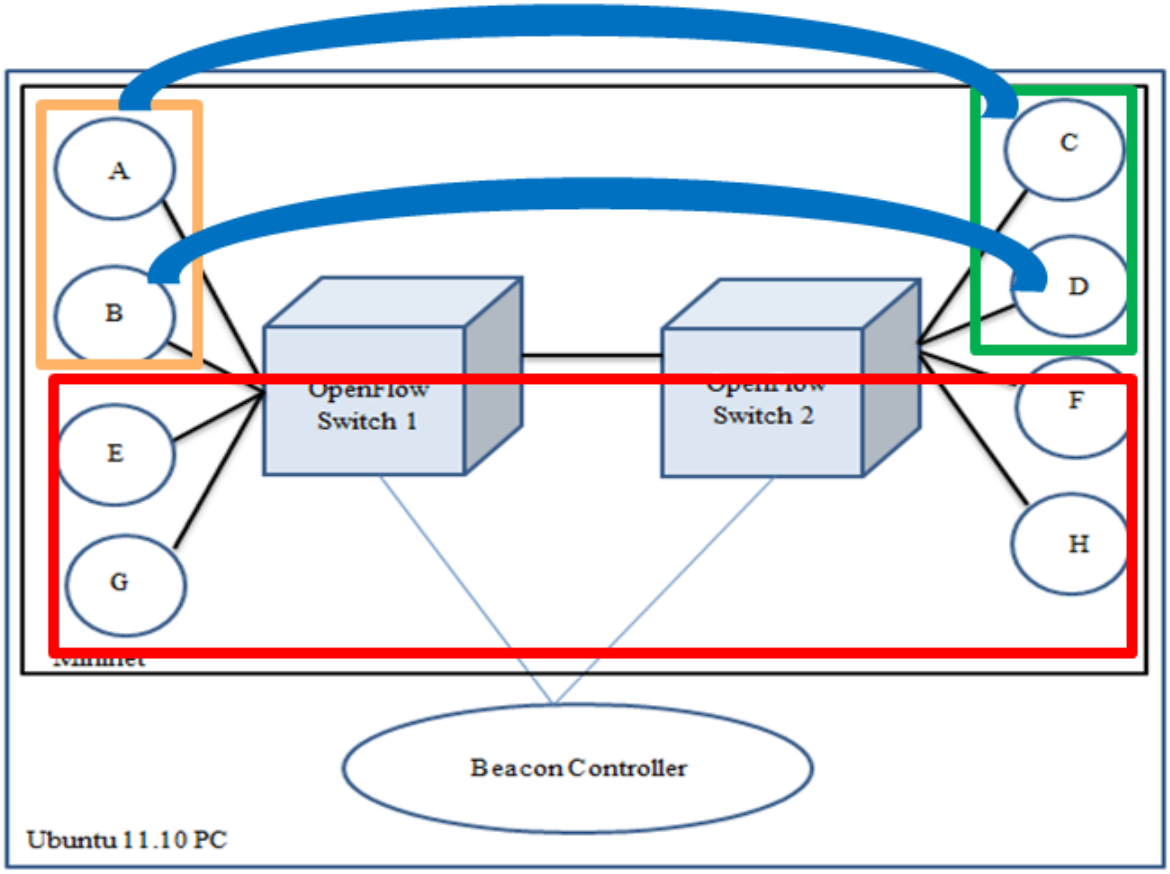
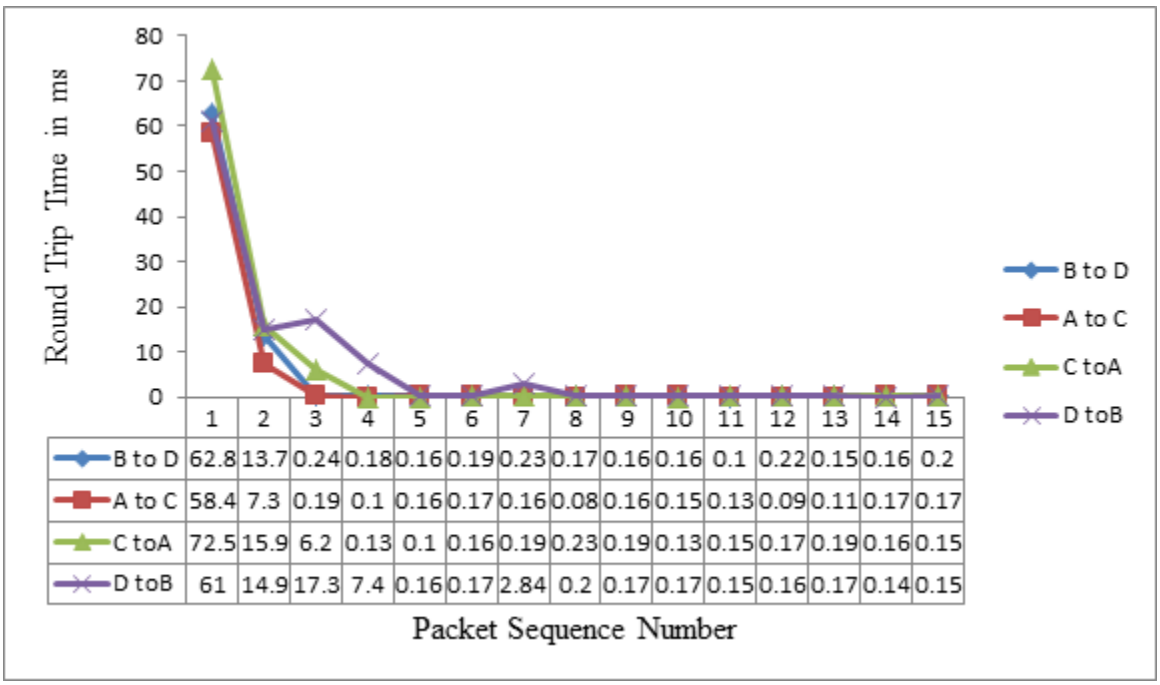


Figure 5.3: Minimet Configuration 2

Again, the OpenFlow controller Application OFWhiteListing described in the Chapter 4 is used to generate the following result. The host B is enabled to communicate with D for duration of two hours. Initially, the hosts B and D belong to two different work groups. The graph in the Figure 5.4 shows the round trip time of the packets flowing in the network between several hosts on the network. The OFWhiteListing application handles the inter and intra workgroup communications at the same speed. We observe that the round trip time of the packets stabilizes in all cases at around 0.2ms, after the first packet taking an average of 64ms. A point to be

noted in the following graph is that, the round trip time between the hosts decrease asymptotically. The reason being, the multiple packets had to be processed at the controller before the controller could actually enter the flows into the switches in the network. But eventually the flows get instantiated in the switches and the round trip time stabilizes.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
B to D	62.8	13.7	0.24	0.18	0.16	0.19	0.23	0.17	0.16	0.16	0.1	0.22	0.15	0.16	0.2
A to C	58.4	7.3	0.19	0.1	0.16	0.17	0.16	0.08	0.16	0.15	0.13	0.09	0.11	0.17	0.17
C toA	72.5	15.9	6.2	0.13	0.1	0.16	0.19	0.23	0.19	0.13	0.15	0.17	0.19	0.16	0.15
D toB	61	14.9	17.3	7.4	0.16	0.17	2.84	0.2	0.17	0.17	0.15	0.16	0.17	0.14	0.15

Figure 5.4: Mininet network configuration 2 running Application OFWhiteListing

.

5.2 Open VSwitch

5.2.1 Open VSwitch Network Configuration 1

The second network platform used was the Open VSwitch (OVS), which is an OpenFlow software switch. Two different networks have been configured using OVS. The following Figure 5.5 shows the network configuration where a single OVS switch is

being used. A Linux PC running Ubuntu 11.10 operating system has been used. The OVS module was built and inserted in to the kernel of the operating system. This enables the PC to become an OVS switch. Kernel-based Virtual Machine (KVM) virtualization software has been used in conjunction with the OVS on the Ubuntu PC. Using KVM, 4 VM hosts as shown in the figure 5.5 below were brought up. They are connected to the OVS switch on the PC through tap devices that are not shown in the figure below. A Beacon OpenFlow controller also runs on the PC and is connected to the OVS switch as an out of band connection.

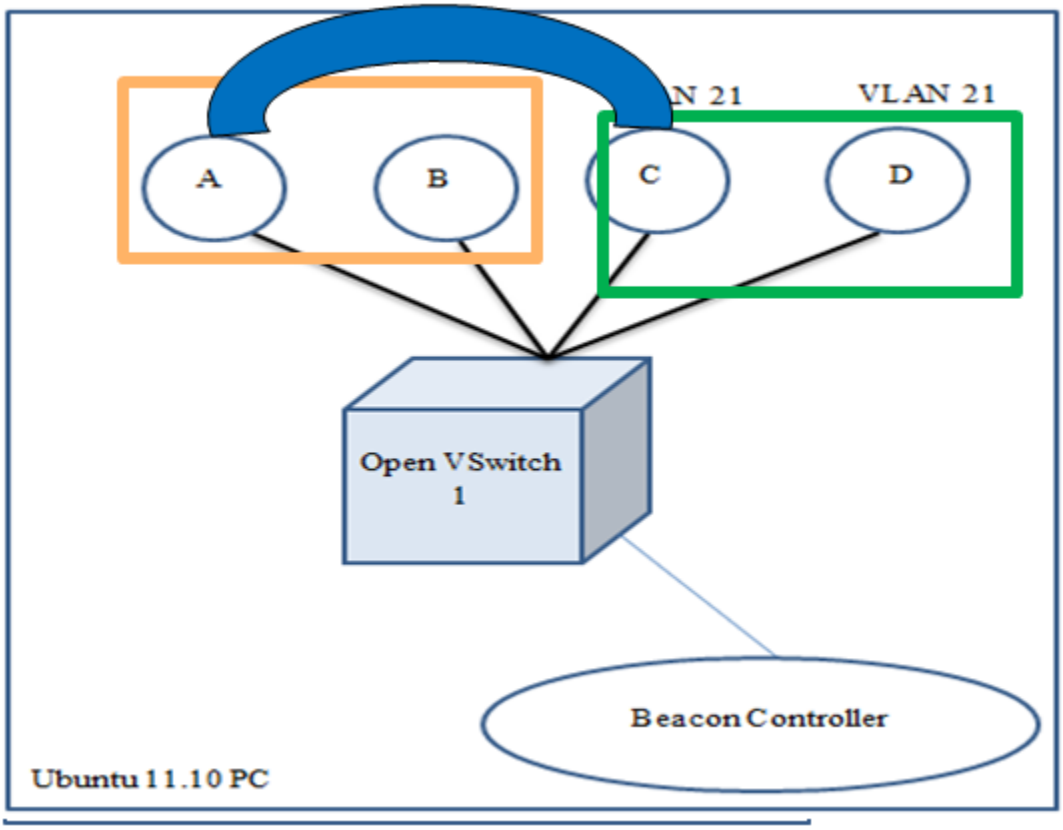


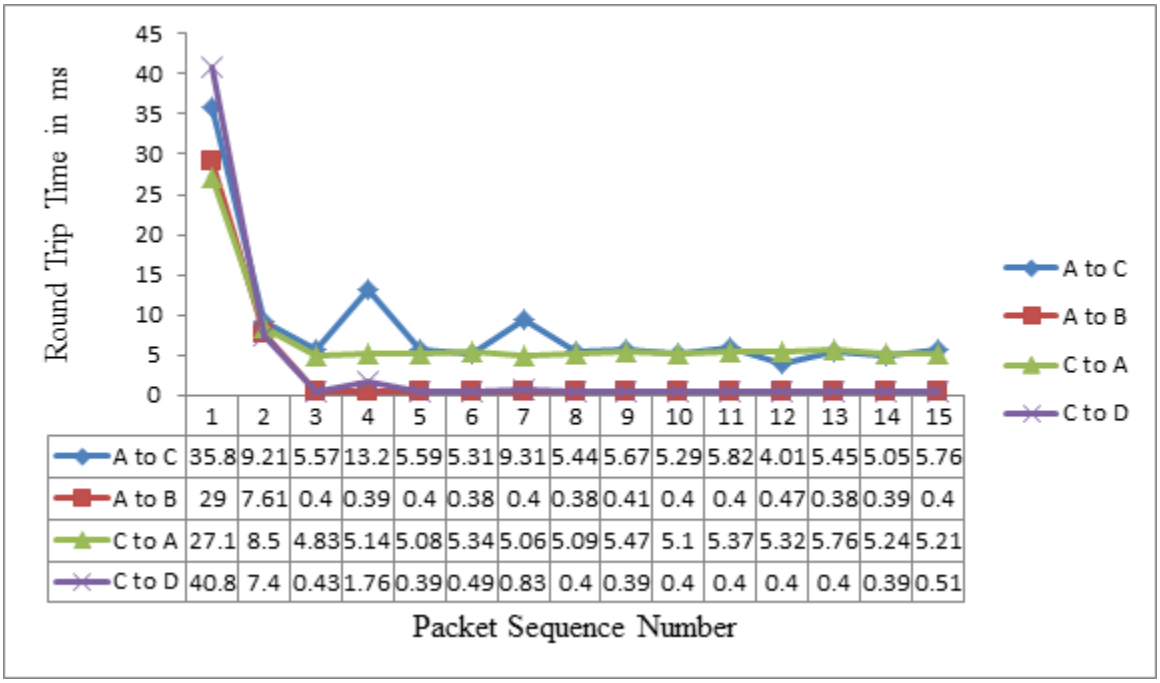
Figure 5.5: Open VSwitch configuration 1

.

The OpenFlow controller Application OFModifyVLAN described in Chapter 4 is run to control the network. Initially hosts C and D belong to VLAN 21. When the

controller application is instructed to allow host B communicate with host belonging to the VLAN 21, the following graphs are generated. The results in the graphs shown in the Figure 5.6 indicate the round trip time of the packets flowing between different hosts in the network. It can be observed that the inter workgroup communication between the Hosts A and C takes much longer than the inter workgroup communication between hosts B and D. After the first few packets, the round trip time of packets between A and C takes about 4.9ms while the communication between B and D takes about 0.5ms. The round trip time between the hosts C to D is observed to be highest. It can be attributed to the delay at the controller while processing the packet. Also, the round trip time of the communication between the hosts A to C spikes occasionally. As the spike is not as high as it is for the first packet, it may be attributed to the occasional delay in processing the packet at the switches. This delay in process of the packets at the switch is most probably because of the packet header modification actions.

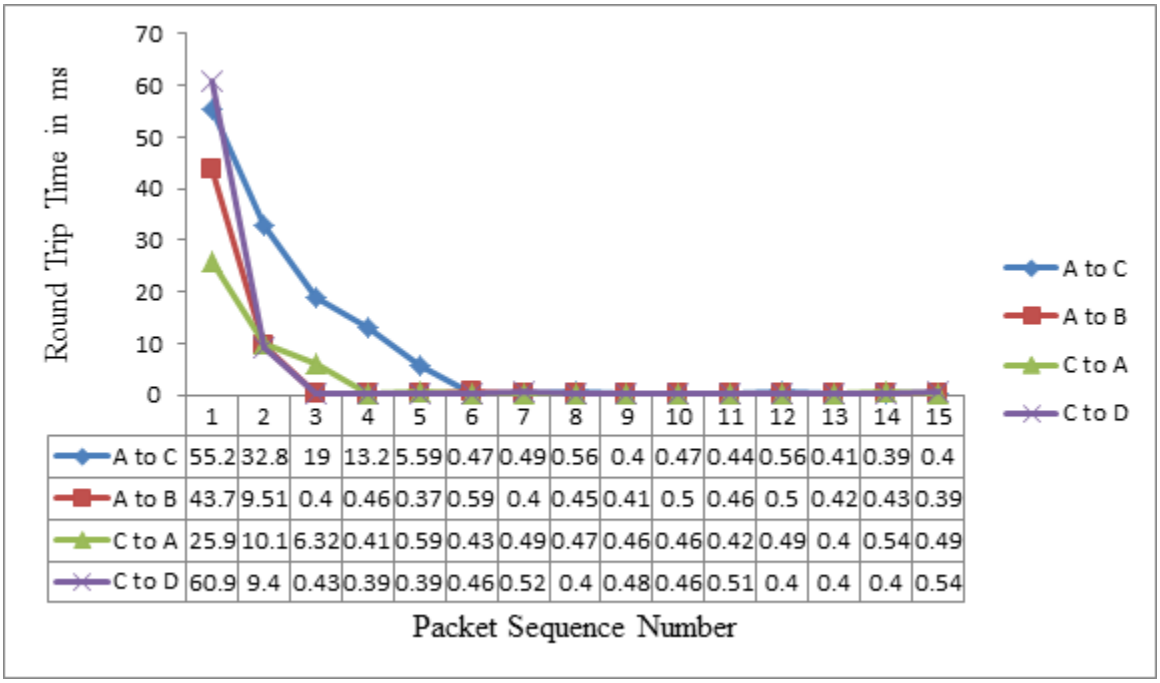
The following graph (Figure 5.7) shows the results obtained by running the OpenFlow controller Application OFWhiteListing on the same configuration as Figure 5.5. The only difference in the configuration being, that the VLANs are no longer used. Instead, the network is in an initial configuration where hosts A and B are in one group and hosts C and D are in another group. And B tries to communicate with C and D respectively. In the following graph, the round trip time between the hosts A and C decrease more gradually than the rest of the flows. The reason being, the multiple packets had to be processed at the controller before the controller could actually enter the flows into the switches in the network. But eventually the flows get instantiated in the switches and the round trip time decreases from 55ms to around 0.5ms.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A to C	35.8	9.21	5.57	13.2	5.59	5.31	9.31	5.44	5.67	5.29	5.82	4.01	5.45	5.05	5.76
A to B	29	7.61	0.4	0.39	0.4	0.38	0.4	0.38	0.41	0.4	0.4	0.47	0.38	0.39	0.4
C to A	27.1	8.5	4.83	5.14	5.08	5.34	5.06	5.09	5.47	5.1	5.37	5.32	5.76	5.24	5.21
C to D	40.8	7.4	0.43	1.76	0.39	0.49	0.83	0.4	0.39	0.4	0.4	0.4	0.4	0.39	0.51

Figure 5.6: Open VSwitch configuration 1 running Application OFModifyVLAN

.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A to C	55.2	32.8	19	13.2	5.59	0.47	0.49	0.56	0.4	0.47	0.44	0.56	0.41	0.39	0.4
A to B	43.7	9.51	0.4	0.46	0.37	0.59	0.4	0.45	0.41	0.5	0.46	0.5	0.42	0.43	0.39
C to A	25.9	10.1	6.32	0.41	0.59	0.43	0.49	0.47	0.46	0.46	0.42	0.49	0.4	0.54	0.49
C to D	60.9	9.4	0.43	0.39	0.39	0.46	0.52	0.4	0.48	0.46	0.51	0.4	0.4	0.4	0.54

Figure 5.7: Open VSwitch configuration 1 running Application OFWhiteListing

.

5.2.2 Open VSwitch Network Configuration 2

The second network configured using the OVS switch software is the one shown in the Figure 5.8 below. In this configuration, two Ubuntu PCs have been installed with OVS. KVM is used to bring up 4 hosts on each of the OVS switches. The two OVS switches are connected through an Ethernet interface eth0 by an RJ45 Ethernet cross over cable. A Beacon OpenFlow controller runs on one of the PC running OVS switch 1. So, it is reachable from both the switches.

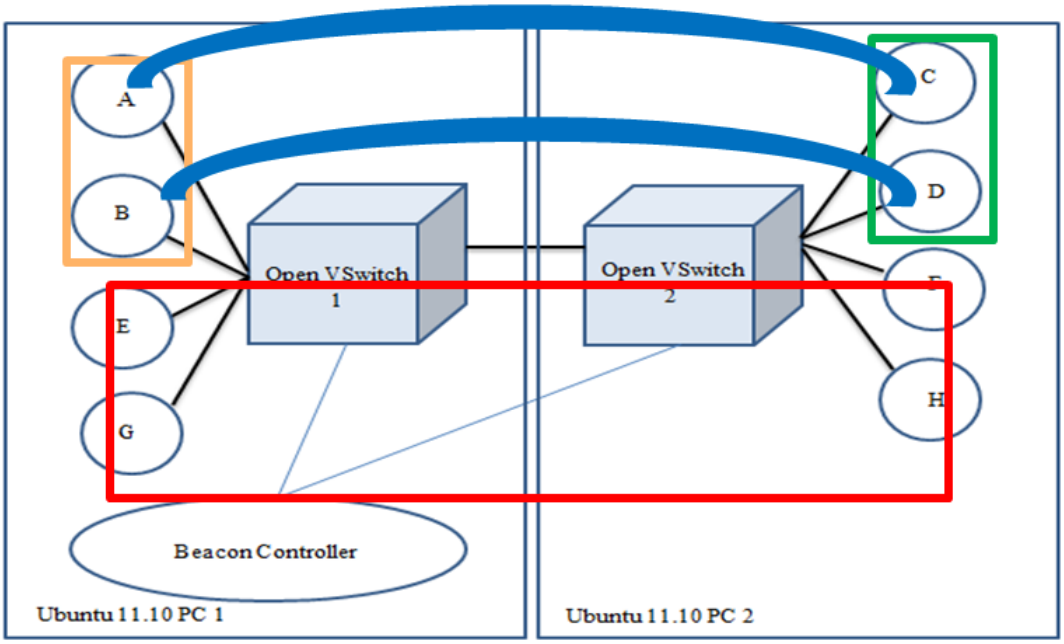
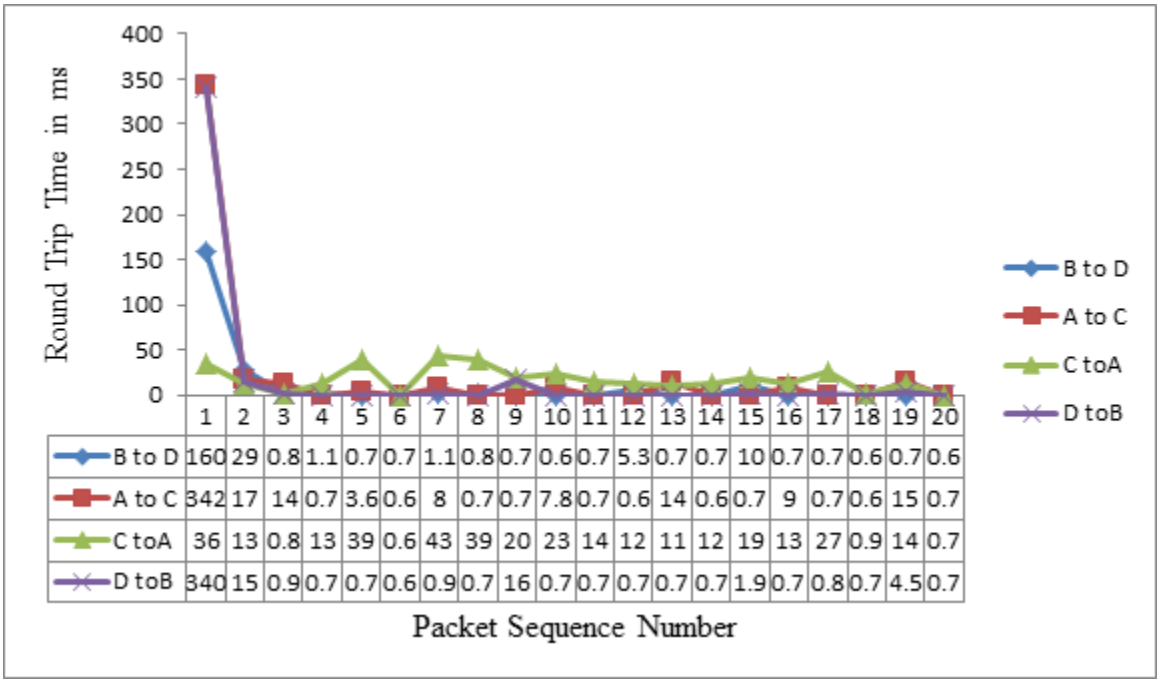


Figure 5.8: Open VSwitch configuration 2

The initial configuration goes as follows: hosts A, B belong to work group wg1; hosts C, D belong to work group wg2; hosts E, F, G, H belong to work group wg3. The controller is instructed to enable the communication link between B and D, although they belong to two different work groups. The graph in the following Figure 5.9 shows the round trip time of the packets flowing between different hosts. The round trip time of the first packet from A to C is around 342ms. Whereas, the time

taken by the first packet from C to A is 36ms. It is significantly lower than the time taken in the opposite direction. The reason behind is that during the experiment, the communication between A to C was initiated first and so, some of the flow entries are already inserted. So, when the communication between C to A was started, some of the acknowledgement packets did not have to go to the controller. Hence the significantly reduced round trip time of the first packet.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
B to D	160	29	0.8	1.1	0.7	0.7	1.1	0.8	0.7	0.6	0.7	5.3	0.7	0.7	10	0.7	0.7	0.6	0.7	0.6
A to C	342	17	14	0.7	3.6	0.6	8	0.7	0.7	7.8	0.7	0.6	14	0.6	0.7	9	0.7	0.6	15	0.7
C toA	36	13	0.8	13	39	0.6	43	39	20	23	14	12	11	12	19	13	27	0.9	14	0.7
D toB	340	15	0.9	0.7	0.7	0.6	0.9	0.7	16	0.7	0.7	0.7	0.7	0.7	1.9	0.7	0.8	0.7	4.5	0.7

Figure 5.9: Open VSwitch configuration 2 running Application OFWhiteListing

.

5.2.3 Open VSwitch controller benchmarking

Finally, in the Figure 5.10 below we present the OpenFlow controller benchmarking in the above two OVS network configuration running either of the two OpenFlow controller applications developed. The X axis indicates the number of packets of length 1500 bytes each being sent to the controller. The Y axis indicates the time

taken to process all the packets sent in milliseconds.

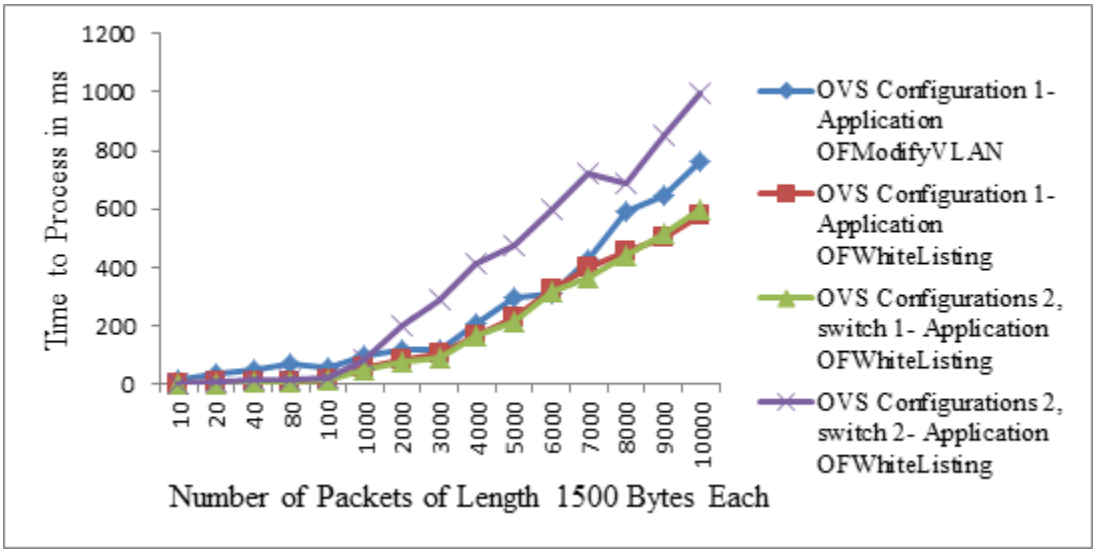


Figure 5.10: OVS network OpenFlow controller benchmarks

.

5.3 HP Procurve 5046Zl

Figure 5.11 shows two hosts connected to the HP Procurve 5046Zl at UNL running firmware supporting OpenFlow 1.0.0. Beacon controller is running on host A and is connected to switch as an out-of-band connection. Initially there is no communication link between the two hosts. By using the controller application OFWhiteListing, the link is setup between A and B to enable communication. We have recorded the round trip time of the packets in the following graph (Figure 5.12). The initial packets take around 70ms while the stabilized time is around 0.65ms.

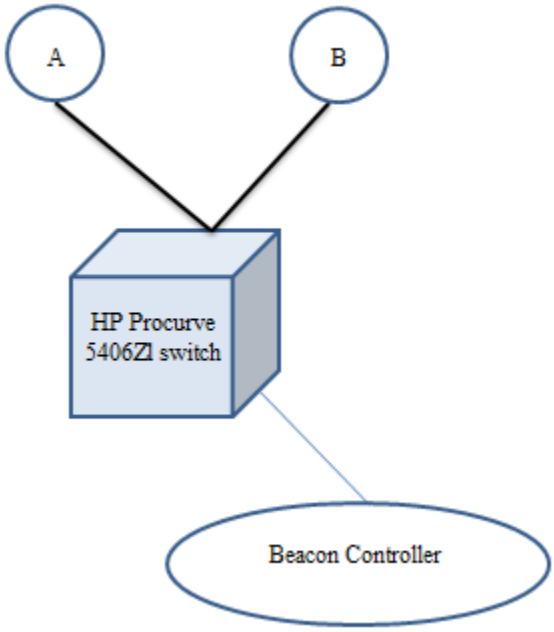
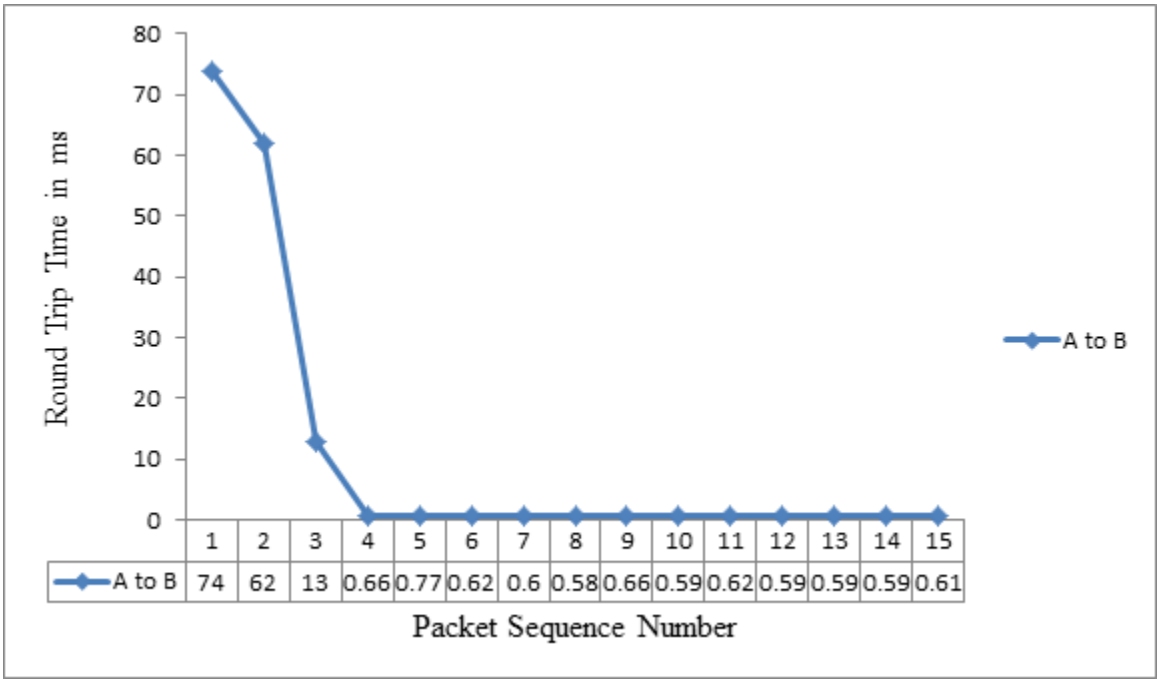


Figure 5.11: HP Switch Configuration

.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A to B	74	62	13	0.66	0.77	0.62	0.6	0.58	0.66	0.59	0.62	0.59	0.59	0.59	0.61

Figure 5.12: HP Configuration configuration 1 running Application OFWhiteListing

.

5.4 Comparison of the measurements

The above experiments have been performed multiple times to obtain the averages and do the comparison of their performances with respect to the round trip time (RTT) of the packets. Three instances of each of the experiments have been performed to obtain the averages.

The following graph (Figure 5.13) shows the comparison of the RTT of the first packet which goes through the controller. Controller application OFWhiteListing is being used. The average is obtained by using the RTT of each communication link (Ex: communication link A to C) in all the three experimental instances. Note that HP has been tested under only one configuration.

It can be observer that, as the complexity of the network grows from configuration 1 to configuration 2, the average RTT increases too. The Open VSwitch configuration 2 has the highest RTT. While Mininet configuration 1 shows the fastest RTT of the packets. And under the same configuration, Miniet is faster than Open VSwitch.

The next graph (Figure 5.14) shows the comparison of the RTT of the later packets which go through the flow tables inserted into the switches rather than the controller. Controller application OFWhiteListing is being used. The average is obtained by using the RTT about 7 packets of each communication link (Ex: communication link A to C) in all the three experimental instances. Note that HP has been tested under only one configuration.

It can be observer that, as the complexity of the network grows from configuration 1 to configuration 2, the average RTT increases too. The Open VSwitch configuration 2 has the highest RTT. While Mininet configuration 1 shows the fastest RTT of the packets. And under the same configuration, Miniet is faster than Open VSwitch.

The third comparison (Figure 5.15) is between the two controller applications

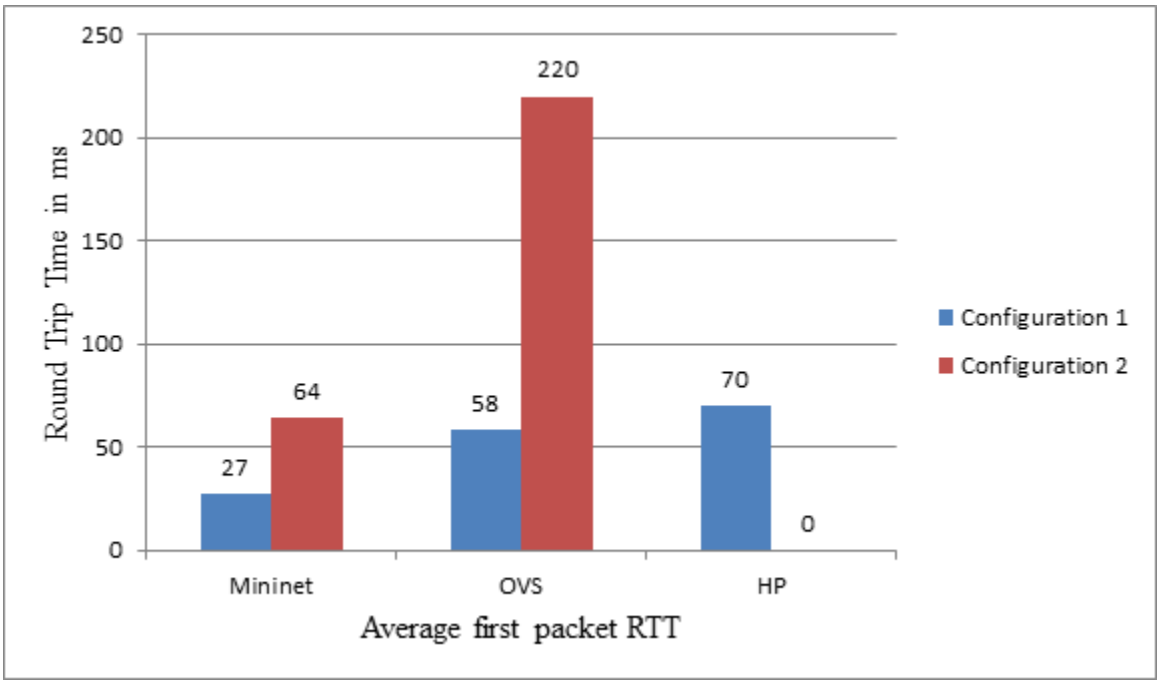


Figure 5.13: Comparison of round trip time of the 1st packet on the 3 network platforms

.

developed. The first packet average RTT as well as average time of 7 packets going through the flow table in the case of Open VSwitch in configuration 1 have been shown in the following graph. The OFModifyVLAN takes lesser time to set up, but performs lower than OFWhiteListing if the later packets RTT is observed. It takes about 2.9ms versus 0.5ms.

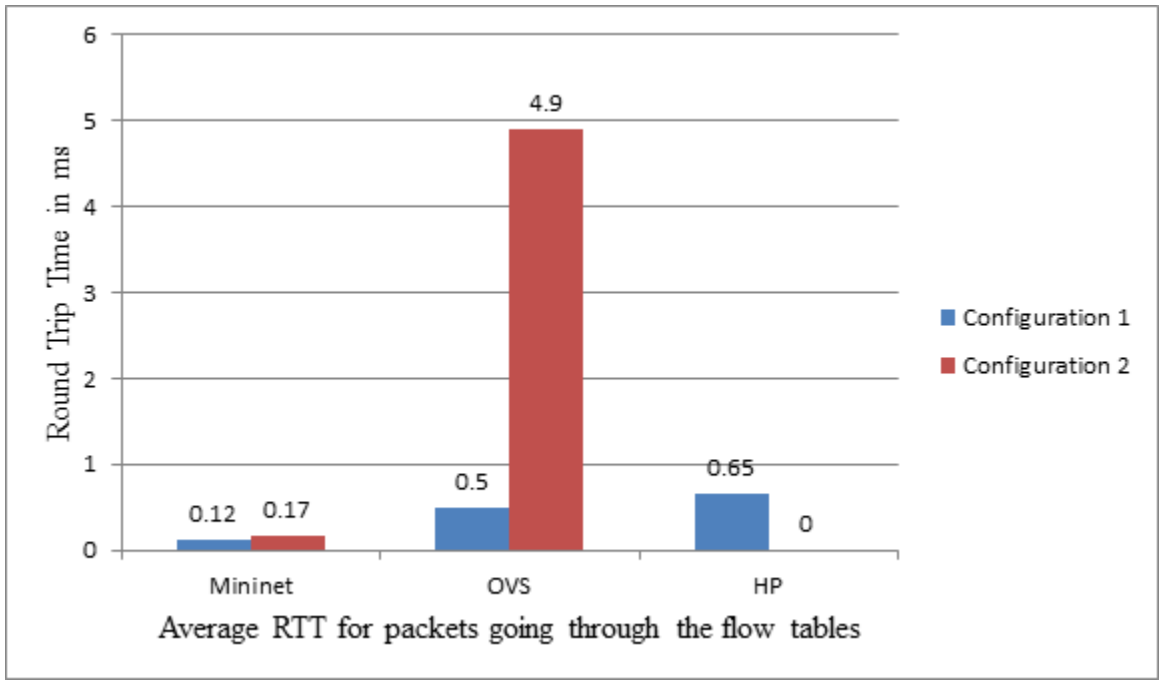


Figure 5.14: Comparison of round trip time of the packet going through flow tables on the 3 network platforms

.

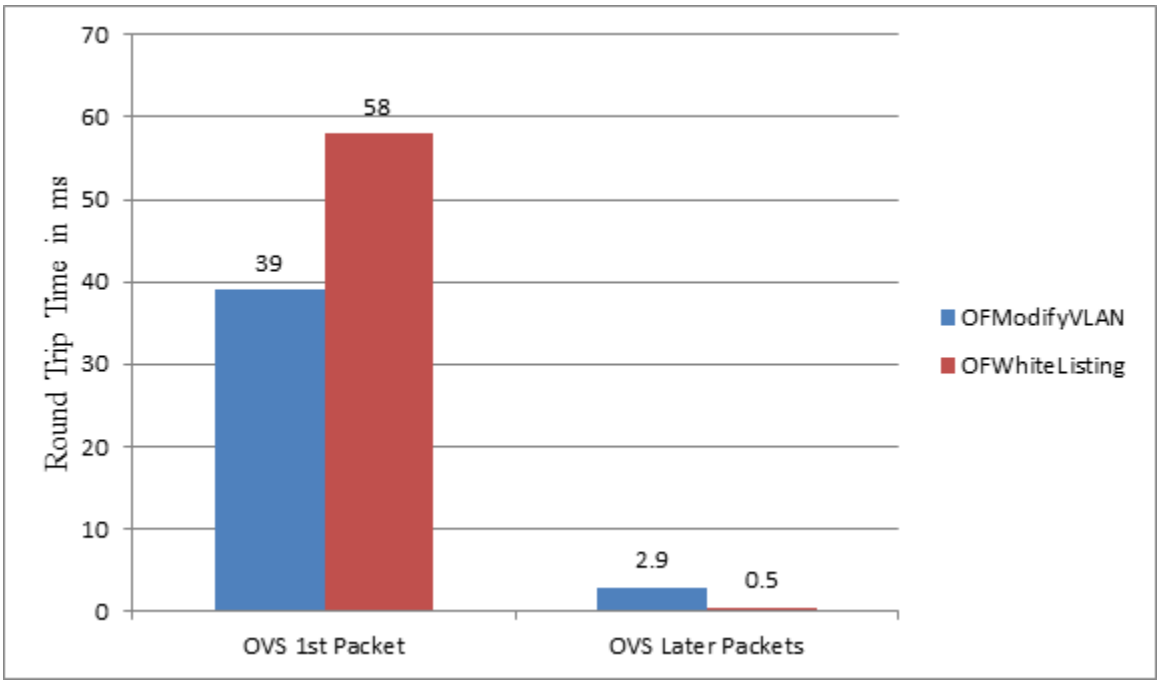


Figure 5.15: Comparision of RTT of packets under 2 OpenFlow applications on OVS

.

Chapter 6

Conclusions and Future work

6.1 Conclusions

In this project, we have studied Software Defined Networking through OpenFlow technology. We have proposed and implemented solution to the tedious traffic isolation problem in a network using an the OpenFlow contoller. Through a centralized OpenFlow controller, a user could pass instructions to change the configurations of the network as dynamically as desired.

We have discussed the multiple platforms used to set up OpenFlow networks. And the configurations used to set up the described OpenFlow network on each of the platforms has been discussed as well. From the results, it is clear that the initial packets that go through the network take longer time as they go through the controller. But the packets following are processed and transmitted at a much faster rate. The reason behind this decrease in round trip time is that when the first packet goes through the network there is no OpenFlow flow entries setup. So, the first packet has to go to the controller where it will be processed and the appropriate flows are setup in the network switches. So, the following packets of the same flow do not go to the

controller. They pass straight through the switches. Hence the decrease in the round trip time. Next, by observing the results from the OVS network configuration 1, we can compare the performance of the two OpenFlow controller applications developed. Under the same physical set up and user requirements, we see that OpenFlow controller OFWhiteListing performs better than that OpenFlow controller application OFModifyVLAN. This is because of the actions that are performed on the packets in each of the application is different. While the application OFModifyVLAN uses modifying and deleting VLAN ID in the packets, application OFWhiteListing uses action to forward a packet if there is a match. This indicates that modification of packet header fields is a more expensive operation than the forwarding and dropping operations. Also, using the results we can compare the performance of the platforms used, as the same application processes the packets at different speeds.

The two OpenFlow controller applications present distinct ways in which OpenFlow technology could be used to solve some of the network management problems we have. The performances of the OpenFlow controller under these multiple OpenFlow enabled network platforms have been compared.We conclude that it is possible for a single OpenFlow controller application to control a network comprising of different types of networking elements. Also, we conclude that given the current hardware capabilities, certain OpenFlow actions such as forwarding packet to a port, are performed faster than others like the actions that modify the header fields in the packet, while processing the packets. The reason is lack of hardware support for handling modification to packet headers.

6.2 Future work

The OpenFlow controller applications developed as part of the project could be extended to include additional feature. Along with the management of traffic in a network, features such as access control lists (ACL), firewalls, etc. could be included into the OpenFlow controller. This could eliminate the need for multiple network management middle boxes which carry out the above mentioned tasks. Another direction of enhancement of the application could be to use the OpenFlow counters which are part of the flow table, in the application. The switches and the OpenFlow controller continuously exchange messages. Based on the counter values received by the controller from the switches, the network configuration could be made to automatically change through the controller.

The OpenFlow specification is continuously evolving to include more features. OpenFlow specifications 1.1.0 and 1.2 are drafted, although the deployments have not yet started. Given the changes in the future specifications, the applications may have to be modified to accommodate these specification changes.

Another area of future work is the execution of the current OpenFlow controller applications on more extensive real networks. One option is to run the application on network testbeds like ESNet [1] and ORBIT[19] which are partly OpenFlow enabled. At UNL, two hardware switches, HP Procurve and Ciena Coredirector have been OpenFlow enabled. A network could be set up across these switches and the applications could be tested further for compatibility and performance. Another option is to connect to OpenFlow switches on other OpenFlow enabled networks through the HP Procurve or the Ciena Coredirector OpenFlow switches currently present on the UNL campus.

Appendix A

OpenFlow Enabled Firmware Update on CoreDirector CI Switch

A.1 Overview

The CoreDirector CI Switch can deliver a wide range of optical capacities, along with Ethernet switching capabilities. The switch supports SONET as well as SDH interfaces, specifically, OC-3/12/STM-1/4, OC-48/STM-16, OC-192/STM-64 optical interfaces, STM-1e electrical interfaces and Gigabit Ethernet interfaces. They provide non-blocking, bidirectional switching capacity that can be configured to switch and groom traffic from any input port to any output port down to the STS-1/VC-3 level. The OpenFlow standard is designed to realize the concept of Software defined networks. OpenFlow is a specification that enables programmable networks at campus level.

Establishing Dynamic Circuit Network (DCN) is currently implemented in Internet2 by using the software components On-Demand Secure Service and Advance Reservation System (OSCARS) and Dynamic Resource Allocation via GMPLS Op-

tical Networks (DRAGON) across various domains and technologies.

OSCARS is a networking service deployed in the DoE ESnet to create dynamic, deterministic and secure circuits across the ESnet network. MPLS and RSVP are the key protocols used to create advance reservations of andwidth using the software components developed as part of OSCARS project. DRAGON was a NSF funded project to dynamically provision network resources across various domains and across heterogeneous networking technologies. GMPLS is the key protocol used to create circuits spanning across both optical and Ethernet domain and hence DRAGON creates a Layer 1 virtual circuit. By making the CoreDirector CI switch, which is capable of Optical Switching, OpenFlow enabled, a circuit switch can be controlled by OpenFlow.

A.2 Required Tools

The Ciena Core Director at UNL is now being upgraded with OpenFlow enabled firmware.

The procedure includes: loading the software on the File Transfer Protocol (FTP) server, and upgrading the CoreDirector Network Element (NE) software.

- A Laptop Personal Computer (PC) running Windows 7 has been used. Laptop or Personal Computer (PC) running Windows NT® , Windows 2000® , or Windows XP® can also be used. Following software should be available on the Laptop or Personal Computer (PC) being used:
  - Ciena CoreDirector Node Manager Software that corresponds to the currently installed CoreDirector software release. The Core Director at UNL was running Version 5.2.6.

  - Ciena CoreDirector Node Manager Software that corresponds to the CoreDirector software upgrade release. The Core Director at UNL is upgraded to OpenFlow capable Version 6.1.1.
  - HyperTerminal application or equivalent. Note: Windows 7 no more has HyperTerminal as an in built feature. It has to be downloaded in case the PC runs Windows 7.
  - FTP server application or equivalent (such as wftpd32.exe or equivalent). wftp FTP software has been used. The wftp FTP software is available at `http://www.texis.com`.

- Cables to physically connect from the Laptop or the PC to the Core Director node
  - 9-pin to 25-pin RS-232 serial cable (DB-9F to DB-25M straight cable) for connection to the Core Director node.

Note: Most of the newly manufactured computers do not have the 9-pin RS-232 port. So, we must use an USB to DB-9M adapter. Using this adapter we connected to the DB-9F to DB-25M straight cable. The DB-9F to DB-25M straight cable was in turn connected to the Core Director node.

A.3 User Prerequisites

- The user must be able to log on to the CoreDirector Node Manager software.
- The user must have Administrator privileges account to access the CoreDirector Switch software.

- The user must be a registered system user with a valid user name and password to log in to the Command Line Interface (CLI).
- The entire upgrade is done by logging on to the telnet connection session.

A.4 Overview of the Software Update on the CoreDirector

1. Load Software on FTP Server
    - Copy the CoreDirector software upgrade file to the Software Release directory of the FTP server on the laptop computer.
    - Start the FTP server application
2. Copy and Save Current Installation Settings
    - Log on through hyper terminal, with the super user username and password (Figure A.1)
    - From the CoreDirector CM CLI Menu (Figure A.2), type the option number to Display the current install settings and press Enter. This would save the current installation settings.
3. Take back Up of the CoreDirector Database
4. Upgrade CoreDirector NE Software
    - From the node's CLI (command line interface) Menu, type the option number to Enter Upgrade mode and press Enter. The Upgrade Menu is displayed.

- From the Upgrade Menu, type the option number to Download and process a new load and press Enter. The Transfer Software Load menu is displayed.
- From the Transfer Software Load Menu, type the option number to Specify the IP Address of the FTP site and press Enter. The FTP site may be a PC or laptop. The IP address of the laptop being used is specified as the IP address in our case.
- After giving the IP address of the FTP server, we load the file on to the switch, Unarchive the file and validate the load.
- Return to upgrade menu.
- From the Upgrade Menu, type the option number to List available software versions and press Enter.
- Type the option number to Select version to upgrade to and press Enter.
- Type the version for upgrade and press Enter.
- Type the option number to Start upgrade and press Enter. Once the upgrade is done, Type the option number to Switch to the upgrade and press Enter.

The upgrade is complete.

A.5 Conclusion

Now, the Ciena Coredirector CI switch is OpenFlow enabled. The capability of the switch to switch and groom traffic down to a granularity of STS-1/VC-3 level can be controlled by an OpenFlow controller. After the firmware update, the OpenFlow Configuration option in the menu is present through command line interface (Figure A.3)

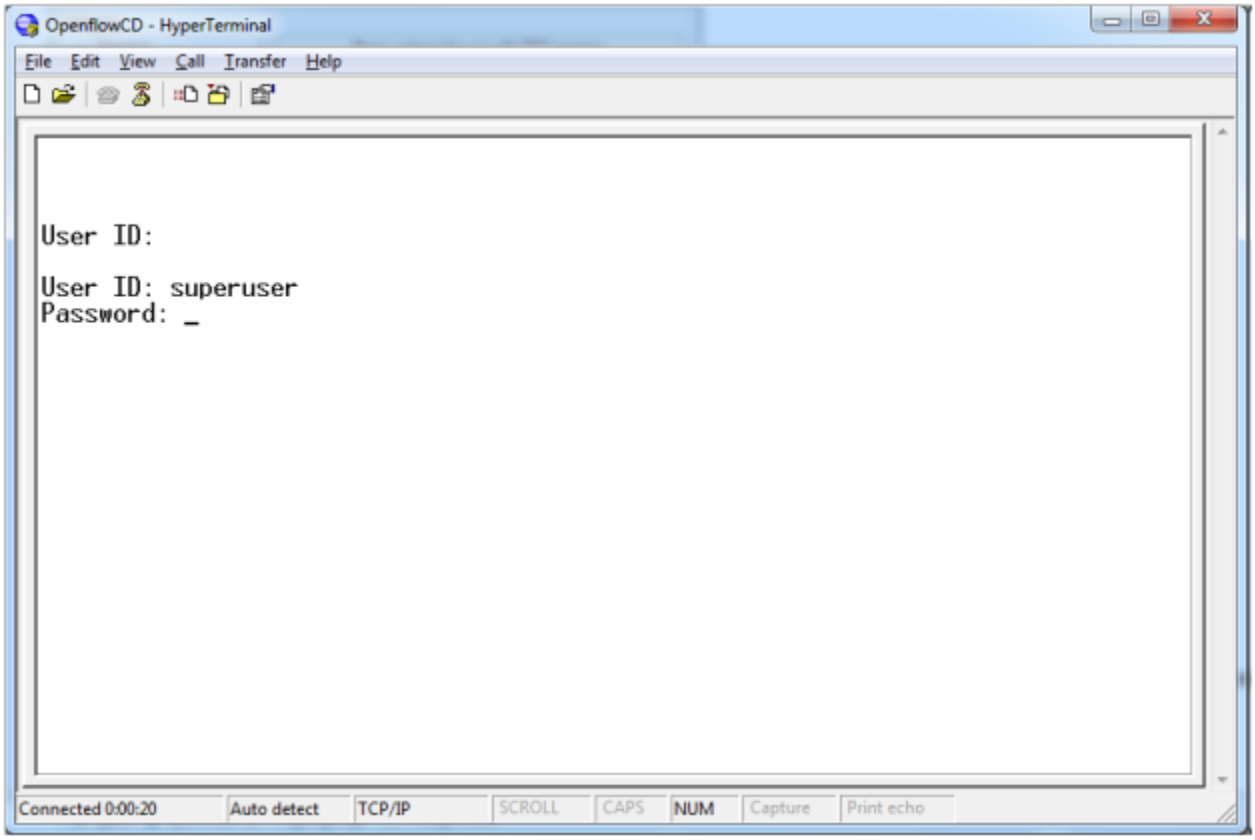


Figure A.1: CoreDirector Login Screen

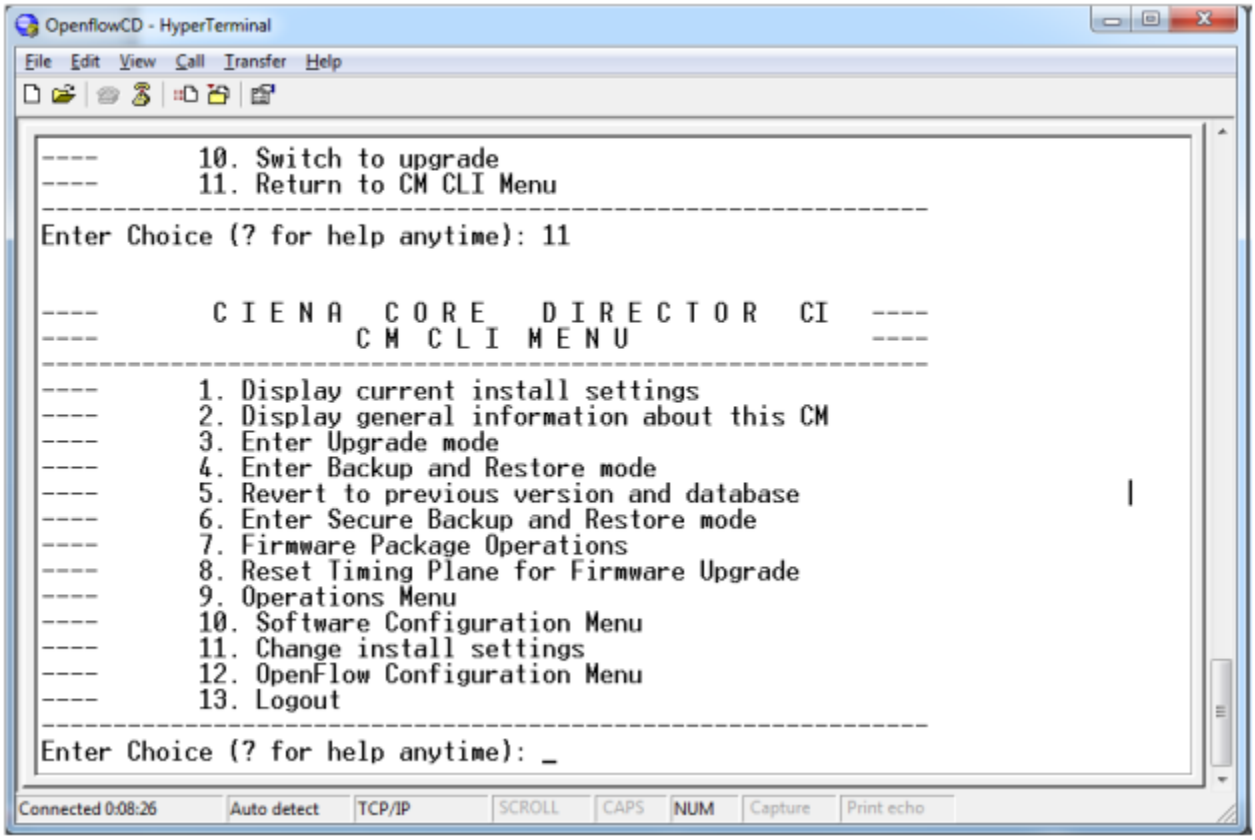


Figure A.2: CoreDirector Menu Before Upgrade

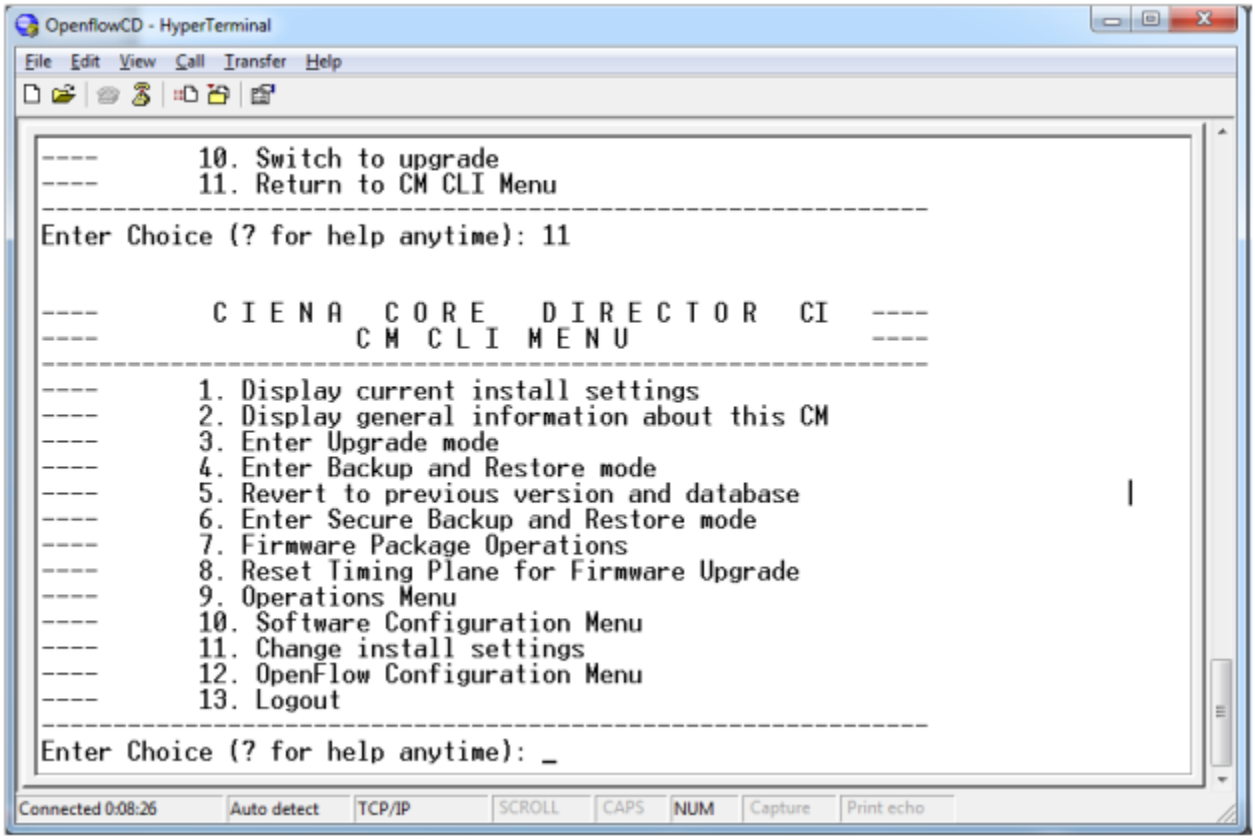


Figure A.3: CoreDirector Menu After Upgrade

Appendix B

Source Code

B.1 OFModifyVLAN Source Code

```
public Command receive(IOFSwitch sw, OFMessage msg) {
        OFPacketIn pi = (OFPacketIn) msg;
        LongShortHopscotchHashMap macTable = macTables.get(sw);
        if (macTable == null) {
            macTable = new LongShortHopscotchHashMap();
            macTables.put(sw, macTable);
        }

        // Build the Match
        OFMatch match = new OFMatch();
        match.loadFromPacket(pi.getPacketData(), pi.getInPort());

        int packetIPsrc = match.getNetworkSource();
        int packetIPdst = match.getNetworkDestination();
```

```
// if the src is not multicast, learn it
if ((dlSrc[0] & 0x1) == 0 && dlSrcLong != 0) {
    if (!macTable.contains(dlSrcLong) ||
            macTable.get(dlSrcLong) != pi.getInPort()) {
        macTable.put(dlSrcLong, pi.getInPort());
    }
}
if(intipSRC == packetIPsrc && intipDST == packetIPdst){
.
.
.

 log.info("the ...dlSrcHostLong is {}", dlSrcHostLong);
 OFActionVirtualLanIdentifier action1 = new OFActionVirtualLanIdentifier()
 action1.setVirtualLanIdentifier(id);

  OFActionOutput action2 = new OFActionOutput()
  .setPort((short) OFPort.OFPP_NORMAL.getValue());


   List<OFAction> actions = new ArrayList<OFAction>();
   actions.add(action1);
   actions.add(action2);

 // build flow mod
```

```
    OFFlowMod fm = (OFFlowMod) sw.getInputStream().getMessageFactory()
            .getMessage(OFType.FLOW_MOD);
    fm.setBufferId(bufferId)
        .setIdleTimeout((short) 0)
        .setOutPort((short) OFPort.OFPP_NONE.getValue())
        .setMatch(match)
        .setActions(actions)
        .setLength(U16.t(OFFlowMod.MINIMUM_LENGTH+OFActionOutput.MINIMUM_L
    try {
        sw.getOutputStream().write(fm);
    } catch (IOException e) {
        log.error("Failure writing FlowMod", e);
    }

// Send a packet out
        // build packet out
        OFPacketOut po = new OFPacketOut()
            .setBufferId(bufferId)
            .setInPort(pi.getInPort())
            .setActions(actions)
            .setActionsLength((short) (OFActionVirtualLanIdentifier.MINIMU

        // set data if it is included in the packetin
        if (bufferId == 0xffffffff) {
            byte[] packetData = pi.getPacketData();
            po.setLength(U16.t(OFPacketOut.MINIMUM_LENGTH
```

```
                        + po.getActionsLength() + packetData.length));
                po.setPacketData(packetData);
            } else {
                po.setLength(U16.t(OFPacketOut.MINIMUM_LENGTH
                        + po.getActionsLength()));
            }

            try {
                sw.getOutputStream().write(po);
            } catch (IOException e) {
                log.error("Failure writing PacketOut", e);
            }
.
.
.
}
```

B.2 OFWhiteListing Source Code

```
public Command receive(IOFSwitch sw, OFMessage msg){
Map<LinkTuple, Long> topName = topology.getLinks();
List<Device> devmng = deviceManager.getDevices();

          // Build the Match
        OFMatch match = new OFMatch();
```

```
         match.loadFromPacket(pi.getPacketData(), pi.getInPort());
int sourceGroupID = 0;
.
.
.

        int destinationGroupID = 0;
        boolean flagSrc = false;
        boolean flagDst = false;

        for (ArrayList<IPv4> currentWG : deviceLists) {

            int groupID = listOfGroups.get(currentWG);
            for (IPv4 currentIP : currentWG){
                if(packetIPsrc == currentIP.getSourceAddress()){
                    flagSrc = true;
                    break;
                }
            }
            if(flagSrc == true){
                sourceGroupID = groupID;
                break;
            }
        }

        for (ArrayList<IPv4> currentWG : deviceLists) {
```

```
    int groupID = listOfGroups.get(currentWG);
    for (IPv4 currentIP : currentWG){
        if(packetIPdst == currentIP.getSourceAddress()){
            flagDst = true;
            break;
        }
    }
    if(flagDst == true){
        destinationGroupID = groupID;
        break;
    }
}


if(sourceGroupID == destinationGroupID){
    for(IOFSwitch currentSw: beaconProvider.getSwitches().values()){
        match.setInputPort(pi.getInPort());

        // build action

        OFActionOutput action = new OFActionOutput()
        .setPort((short) OFPort.OFPP_NORMAL.getValue());

        // build flow mod
        OFFlowMod fm = (OFFlowMod)currentSw.getInputStream().getMessageFac
                .getMessage(OFType.FLOW_MOD);
```

```
fm.setBufferId(bufferId)
    .setIdleTimeout((short) 0)
    .setOutPort((short) OFPort.OFPP_NONE.getValue())
    .setMatch(match)
    .setActions(Collections.singletonList((OFAction)action))
    .setLength(U16.t(OFFlowMod.MINIMUM_LENGTH+OFActionOutput.MINIM
try {
    currentSw.getOutputStream().write(fm);
} catch (IOException e) {
    log.error("Failure writing FlowMod", e);
}

// build packet out
OFPacketOut po = new OFPacketOut()
    .setBufferId(bufferId)
    .setInPort(pi.getInPort())
    .setActions(Collections.singletonList((OFAction)action))
    .setActionsLength((short) OFActionOutput.MINIMUM_LENGTH);

// set data if it is included in the packetin
if (bufferId == 0xffffffff) {
    byte[] packetData = pi.getPacketData();
    po.setLength(U16.t(OFPacketOut.MINIMUM_LENGTH
            + po.getActionsLength() + packetData.length));
    po.setPacketData(packetData);
} else {
```

```
                    po.setLength(U16.t(OFPacketOut.MINIMUM_LENGTH
                            + po.getActionsLength()));
                }

                try {
                    currentSw.getOutputStream().write(po);
                } catch (IOException e) {
                    log.error("Failure writing PacketOut", e);
                }

 .
 .

    }
}
```

Bibliography

[1] Advanced Networking Initiative (ANI), http://www.es.net/RandD/advanced-networking-initiative/.

[2] Clemson OpenFlow Agregate, http://groups.geni.net/geni/wiki/GeniAggregate/ClemsonOpenFlow.

[3] Floodlight, http://floodlight.openflowhub.org/.

[4] GENI, Exploring Networks of the future, http://www.geni.net/.

[5] Georgia Tech OpenFlow Agregate, http://groups.geni.net/geni/wiki/GeniAggregate/GeorgiaTechOpenFlow.

[6] Indiana OpenFlow Agregate, http://groups.geni.net/geni/wiki/GeniAggregate/IndianaOpenFlow.

[7] Internet2, www.internet2.edu.

[8] IP8800 Openflow Networking, http://support.necam.com/pflow/legacy/ip8800/.

[9] Java Spring Framework, http://www.springsource.org.

[10] KSU Lab OpenFlow Aggregate, http://groups.geni.net/geni/wiki/GeniAggregate/KansasStateOpenFlow.

[11] Maestro Platform, http://code.google.com/p/maestro-platform/.

[12] MRI-R2 Consortium: Development of Dynamic Network System (DYNES), http://www.internet2.edu/ion/dynes.html.

[13] National LambdaRail, http://www.nlr.net.

[14] NOX, http://www.noxrepo.org.

[15] Ofelia, http://www.fp7-ofelia.eu/news-and-events/press-releases/ofelia-openflow-facility-now-open-for-experiments/.

[16] Open Networking Foundation, https://www.opennetworking.org/.

[17] Open Networking Summit 2012 Program, http://opennetsummit.org/.

[18] Open VSwitch, http://openvswitch.org.

[19] OpenFlow Experimentation in ORBIT, http://www.orbit-lab.org/wiki/Documentation/OpenFlow.

[20] OSGi Framework, http://www.osgi.org.

[21] PhD Thesis: A UNIFIED CONTROL ARCHITECTURE FOR PACKET AND CIRCUIT NETWORK CONVERGENCE, http://www.openflow.org/wk/index.php/PACC_Thesis.

[22] Rutgers OpenFlow Agregate, http://groups.geni.net/geni/wiki/Geni-Aggregate/RutgersOpenFlow.

[23] Unniversity of Washington OpenFlow Agregate, http://groups.geni.net/geni/wiki/GeniAggregate/WashingtonOpenFlow.

[24] Winsconsin OpenFlow Agregate, http://groups.geni.net/geni/wiki/Geni-Aggregate/WisconsinOpenFlow.

[25] OpenFlow Switch Specification, Version 1.0.0 (Wire Protocol 0x01) , http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf, 2009.

[26] OpenFlow Current Deployments, http://www.openflow.org/wp/current-deployments/, 2011.

[27] OpenFlow Stanford Deployment, http://www.openflow.org/wp/stanford-deployment/, 2011.

[28] OpenFlow Switch Specification, Version 1.1.0 Implemented (Wire Protocol 0x02), http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf, 2011.

[29] OpenFlow Switch Specification, Version 1.2 (Wire Protocol 0x03), https://www.opennetworking.org/images/stories/downloads/openflow/openflow-spec-v1.2.pdf, 2011.

[30] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.*, 2007.

[31] H.E. Egilmez, B. Gorkemli, A.M. Tekalp, and S. Civanlar. Scalable video streaming over OpenFlow networks: An optimization framework for QoS routing. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, 2011.

[32] D. Erickson. Beacon Home, https://openflow.stanford.edu/display/Beacon/Home/, 2011.

[33] N. Foster, R. Harrison, M.l J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: a network programming language. In *Proceedings of*

the 16th ACM SIGPLAN international conference on Functional programming, 2011.

[34] GENI. GENI OpenFlow Backbone Deployment at Internet2, http://groups.geni.net/geni/wiki/OFI2.

[35] N. Gude, P. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 2008.

[36] L. Jianying, J. Pettit, M. Casado, J. Lockwood, and N. McKeown. Prototyping Fast, Simple, Secure Switches for Ethane. In *High-Performance Interconnects, 2007. HOTI 2007. 15th Annual IEEE Symposium on*, 2007.

[37] National LambdaRail. Testbed Networks: Provided by NLR, www.nlr.net/testbeds.php.

[38] B. Lantz, B. Heller, and N. McKeown. A Network on a Laptop: Rapid Prototyping for Software-Defined Networks . In *Proceedings of the ACM HOTNETS 2010 conference*, 2010.

[39] A. Lara, A. Kolasani, and B. Ramamurthy. Network innovation using software defined networking and openflow. *IEEE Communications Surveys and Tutorials*, 2012.

[40] Steven Levy. Going With the Flow: Google′s Secret Switch to the Next Wave of Networking, http://www.wired.com/wiredenterprise/2012/04/going-with-the-flow-google/all/1.

[41] L. Lu, Y. Xiao, and H. Du. OpenFlow control for cooperating AQM scheme. In *Signal Processing (ICSP), 2010 IEEE 10th International Conference on*, 2010.

[42] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 2008.

[43] M. R. Nascimento, C. E. Rothenberg, M. Salvador, and M. F. Magalhães. QuagFlow: partnering Quagga with OpenFlow. In *Proceedings of the ACM SIGCOMM 2010 conference*.

[44] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena, and Maur. F. Magalhães. Virtual routers as a service: the RouteFlow approach leveraging software-defined networks. In *Proceedings of the 6th International Conference on Future Internet Technologies*, 2011.

[45] B. Pfaff, J. Pettit, K.A.T. Koponen, M. Casado, and S. Shenker. Extending networking into the virtualization layer. In *Proceedings of the ACM SIGCOMM HotNets*, 2009.

[46] M. Reitblatt, N. Foster, J. Rexford, and D. Walker. Consistent updates for software-defined networks: change you can believe in! In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, 2011.

[47] R. Sherwood, M. Chanl, A. Covington, and G. Gibb et al. Carving research slices out of your production networks with OpenFlow. *SIGCOMM Comput. Commun. Rev.*, 2010.

[48] Y. Yamasaki, Y. Miyamoto, J. Yamato, H. Goto, and H. Sone. Flexible Access Management System for Campus VLAN Based on OpenFlow. In *Applications and the Internet (SAINT), 2011 IEEE/IPSJ 11th International Symposium on*, 2011.

[49] K. K. Yap, T. Y. Huang, B. Dodson, M. S. Lam, and N. McKeown. Towards software-friendly networks. In *Proceedings of the first ACM asia-pacific workshop on systems*, 2010.

[50] C. Zheng, L. C. Alan, and T. S. E. Ng. Maestro: A System for Scalable OpenFlow Control. *Rice University Technical Report TR10-08*, 2010.