Department Of Computer Science & Information Systems

A Report on

Investigations into Co-existence of Controllers in Software Defined Networking

Submitted By

M Sharat Chandra (2014A7PS108P)

Under the supervision of

Prof. Dr. Rahul Banerjee

In the partial fulfilment of the requirement of the course

CS F266: Study Oriented Project

Birla Institute of Technology & Science

Pilani

Acknowledgement

I would like to express our special gratitude to Professor Dr. Rahul Banerjee who gave me the opportunity to do a project focussed on solving challenging problems in the field of Software-Defined Networking. I would also like to thank Pranjal Ranjan sir who had supported us throughout this project. Their mentorship and motivation made me accomplish the goals of this project. I gained deep understanding and lots of knowledge in the field of Software-Defined Networking. More importantly, I learnt and got insights about how to approach research in the right way. This research project has greatly helped me in understanding the real world applications of SDN which will help solve the challenges that exist in this field.

Table of Contents

1. Introduction

Software-defined networking (SDN) is growing up to be a standout amongst the most encouraging solutions for future Networking. It decouples the control plane from the data plane and provides programmability for network application development by breaking the vertical integration which existed for years. In a general SDN scenario, the network is divided into many subdomains each under a different controller. The goal of this project is to investigate into the coexistence of various controllers in a hybrid scenario where both open-source and proprietary SDN components exist.

2. Background

2.1 Software-Defined Networking

"Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow® protocol is a foundational element for building SDN solutions." [4]

2.1.1 The SDN Architecture

Every traditional network device has two planes. The first plane is the forwarding plane which forwards the data; therefore, it is also called as the data plane. The second one is the control plane, which performs the tasks which need intelligence in the network such as routing and topology discovery. Figure 2 given below depicts the idea of SDN which decouples the two aforementioned planes and makes the network intelligent, programmable and responsive one that can be controlled by a centralized controller.

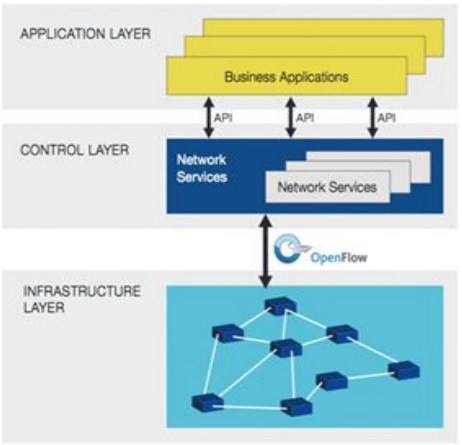

Figure 2: Basic Architecture of SDN network

Figure 2 illustrates the three layers present in the SDN architecture.

- **Infrastructure layer** comprises the forwarding network elements. Hence, it is also called the data plane or the forwarding plane. The data plane mainly performs data forwarding, as well as information monitoring and collection of statistics.
- **Control layer** is responsible for programming and managing the forwarding/data plane. It is also called the control plane. The control plane consists of software controllers which communicate with the forwarding plane through the standardized interfaces known as **southbound interfaces**.
- **Application layer** mainly consists of network applications which can facilitate in the introduction of new network features such as security, novel forwarding schemes, manageability and also assist in the configuration of the network. The controllers provide an abstracted and global view of the network to the application layer. The application layer uses that information to program the network in any way desired. The interface between the application layer and the control layer is known as the **northbound interface**.

2.1.2 Benefits of SDN

Software Defined Networking will change the way that system specialists and designers construct and work with their networks to accomplish their requirements. With the introduction of SDN, the manageability and programmability of networks has increased. Also, the networks have become open-source and non-proprietary. SDN provides more control of the networks, permits to tailor and to enhance their networks to lessen the general cost of maintaining the network. Some of the main SDN benefits can be summarized below [6].

- **Fast Service Deployment**
- **Automated Configuration**
- **Network Virtualization**
- **Reducing the Operational Expense**
- **Network Management Simplicity**

3. Problem Statement and Objectives

3.1 Problem Statement

“Investigations into co-existence of open source and proprietary controllers in Software-Defined Networking (SDN)”

3.2 Objectives

- Develop an understanding of SDN through literature survey
- Build a comprehensive knowledge of controllers in SDN
- Investigations into various existing methodologies for co-existence of controllers
- Development of framework to support co-existing controllers

4. Methodologies for co-existence of controllers

SDN domains [12] are built by system and network administrators which vary in scale for adaptable requirements. An SDN controller needs to provide powerful network service capabilities to applications in such a scenario. Also for defining domains and interconnections just the simple connections between SDN components won’t be sufficient; consideration of various aspects such as the connections of network topologies, which of the neighbouring controllers to talk to and how can their addresses be fetched.

After performing a thorough literature survey, two approaches have been identified to suit the current scenario at hand. The following subsections detail those two approaches.

4.1 Vertical Interfacing Approach

In vertical interfacing approach all the controllers communicate via a master-slave architecture. The master controller is a custom software which communicates to various controllers using the Northbound APIs. This controls the network by instructing the slave controllers to install flows in the switches that they are connected to.

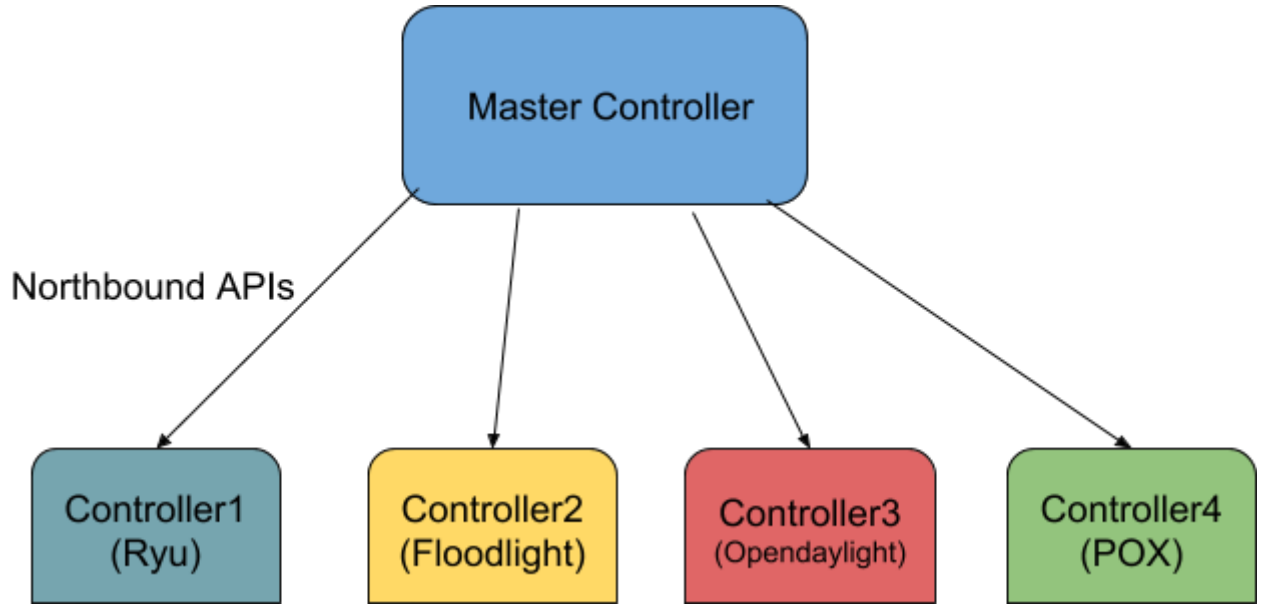

Figure 4: Overview of vertical approach

4.1.1 Master Controller

The master controller connects to all the slave controllers using their Northbound interfaces. This controller generates a global view of the network and has access to install flows on various slave controllers. It also acts as a monitoring framework by gathering network data. The data gathered via requests made to the REST APIs can be used to develop custom routing algorithms based on live traffic situation. Thus, the three main jobs of this controller become:

- Provide a global view of the network
- Manage traffic by installing flows using routing algorithms
- Monitor the live traffic

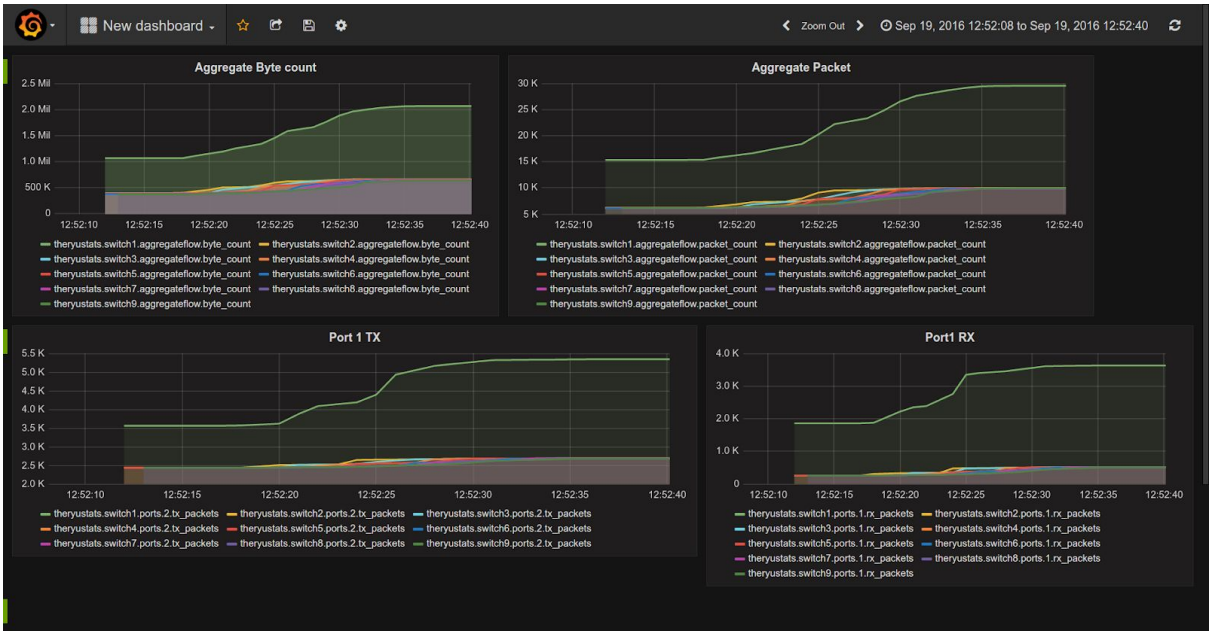

Figure 5: Screenshot of the monitoring system

4.1.2 Slaves (SDN Controllers)

The SDN controllers which are connected to switches share their information with the master controller via various APIs. They allow the installation of flow rules in the switches under their control by providing the master Northbound Interfaces. Various controllers have different NBIs and the master has a module to decode and send the messages for each individual controller.

4.2 Horizontal Approach

In horizontal approach, all the controllers talk to each other via a protocol. The protocol SDNi [12] is proposed for the interface between Software Defined Networking (SDN) spaces to trade data between the area SDN Controllers.

4.2.1 SDN Domains

An SDN domain is a part of the system dictated by the system administrator. It has a SDN controller to control the space, it can specifically speak with other SDN-competent device. The SDN controller keeps up a global perspective of topology secured by different NOS by collecting network topology views from those domains under consideration.

While getting an application ask for certain system assets, the SDN controller ought to choose if the demand can be fulfilled. On the off chance that the demand can be fulfilled, the controller ought to teach the devices in its area to do as such.

4.2.2 SDNi

There exists a need to propose a protocol for interfacing SDN domains. SDNi is one such protocol. It is in charge of organizing practices between SDN controllers and trade control and application data over various SDN domains. SDNi acts like an "east-west" protocol amongst SDN controllers. All things considered, SDNi ought to be a convention actualized by the NOS and how the applications/SDN controllers that keep running in the NOS utilize this protocol (i.e. what is the API to utilize it).

The tasks which should be intrinsically possible through SDNi protocol are to:

- Facilitate flow setup initiated by applications having data on Quality of Sevice, Routing requirements etc. and so forth over numerous SDN domains.
- Trade reachability data to encourage inter-SDN routing. This will permit a solitary stream to navigate different SDNs and have every controller select the most fitting path when numerous such ways are accessible.

4.2.3 SDNi Protocol

The function of SDNi protocol is to perform the information exchange amongst the SDN domains that are under the control of a solitary network administrator or many network administrators working in collaboration. One approach to the implementation of SDNi is to utilize expansion of BGP and SIP over SCTP protocols to exchange information.

5. Testbed and Experiments

To unleash the power of programmability on networks, the theoretical knowledge needs to be implemented on an experimental research test-bed for understanding SDN components. We propose “**SoftNet Testbed**”, a network testbed for Software-defined networking research and development.
This test bed was built in two phases. Phase I of the testbed consists of regular desktop systems along with a legacy layer 2 switch (Juniper) and instances of Open vSwitch installed on every system which would enable them to simulate a data center rack. In Phase II, we used hardware switches (with native SDN support) and high performance servers.

Capacity	**Phase I**	**Phase II**
CPU Type	Intel Core i7™	Xeon™
Processor	1	2
Cores	8	12
CPU Frequency (GHz)	3.40	2.5
Memory (GB)	32	64
DISK (GB)	1000	8000

NIC (Mbit/s)	1000	1000 (x 4)
Kernel	3.10.0+2	4.4.0
OS	XEN 6.5	Ubuntu 16.04

Table 1: Hardware Descriptions of Testbed

5.1 Testbed Phase I

The preliminary setup is focused on getting quick understanding of SDN components. The specifications are described in Table 1. We have used desktop CPUs and traditional layer 2 network switch to start-off with. The setup contains 3 PCs with Xenserver 6.5 OS installed. In addition, we have used 2 storage pools for iso files and vm files using the nfs-kernel-server in Ubuntu 16.04. Monitoring and management of the VMs installed is done by an additional Windows machine containing XenCenter. We have set up the SDN controller in a VM which controls the Open vSwitches in each of the XenServers. Each XenServer can be imagined as a virtual datacenter rack, with Open vSwitch (OVS) acting as a top of the rack (TOR) switch.

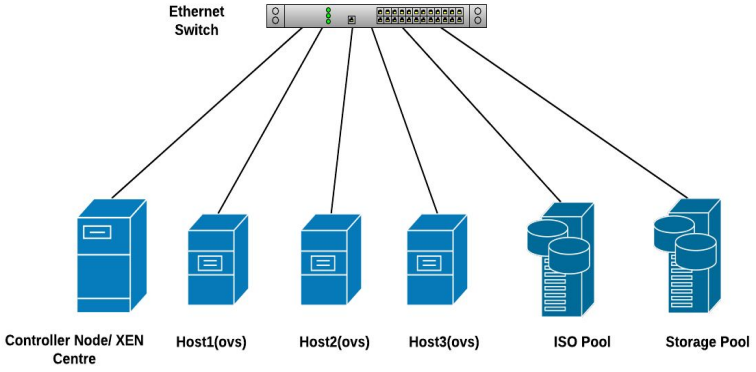

Figure 6: Physical Setup - Phase I

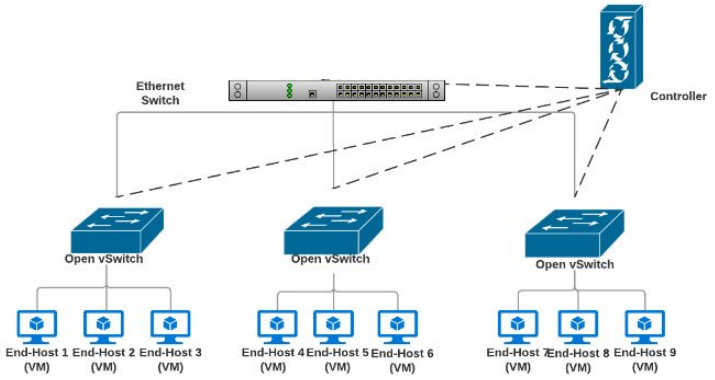

Figure 7: Logical Representation - Phase I

5.2 Testbed Phase II

Our final SDN testbed is set up on the SDET network rack in the SDN Lab. We used three HP 3500yl procurve switches. Also, we have used 3 server class machines with Ubuntu 16.04 installed on each of them. These servers are configured in RAID 6 configuration. Each HP 3500-yl switch is configured in Openflow-SDN mode. These switches are logically connected to a RYU controller and a DHCP server which are set up on desktop systems [Dell Optiplex 7040]. One of the switches acts as the aggregation switch while the other two act as TOR switches. The TOR switches are connected to three server-class machines. Each server contains multiple vms. The figure below shows the topology and connections between the switches. The Phase II of our testbed is pictorially represented in figures below.

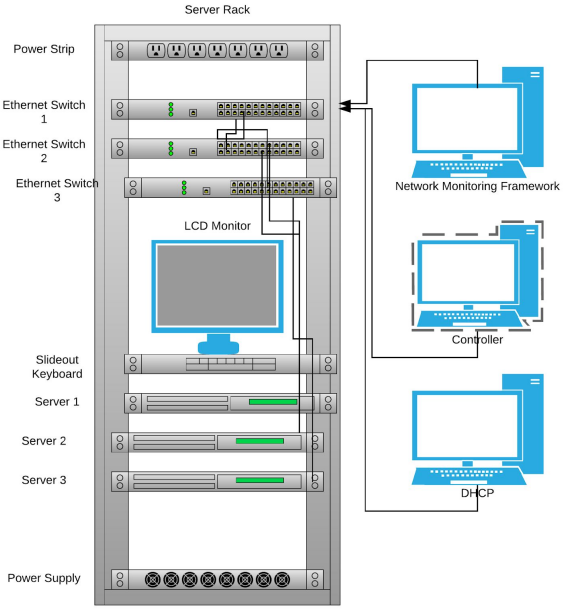

Figure 8: Physical Setup - Phase II

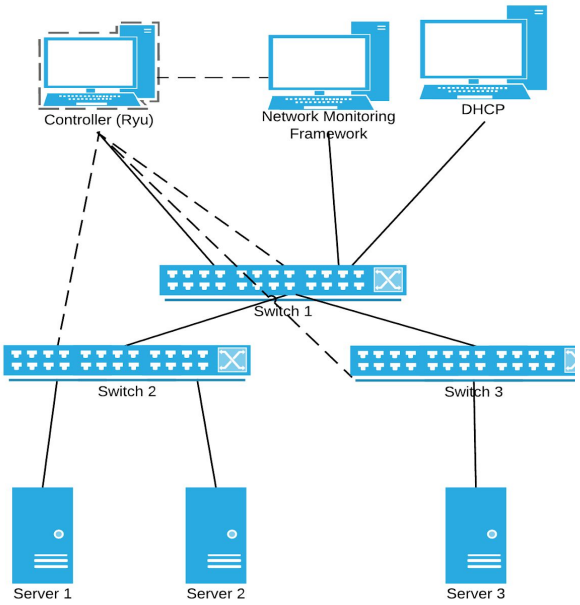

Figure 9: Logical Representation Phase II

5.3 Experiments Performed

We designed some experimental scenarios to test on our testbed.

5.3.1 VM Migration

Live Migration of Virtual Machines can be performed in between the servers via Xen and KVM Management tools. Using this capability, several experiments can be conducted. One such experiment is a comparative analysis of live migration in SDN and Legacy network scenario. Live migration of Virtual Machine is a frequent activity in Data Centres. Thus, there exists a need to prioritize such traffic in between servers. Using the programmability offered by SDN, such flow rules can be installed which would prioritize the migration. The aim of these tests was to replicate the abovementioned scenario on a small scale and study the results achieved in SDN and non SDN modes. As expected the results were in favor of SDN as it took considerably less amount of time to migrate the same VM in SDN scenario compared to a legacy one where prioritized migration was not possible.

5.3.2 TCP Congestion

Transmission Control Protocol is a protocol in Transmission Layer responsible for reliable end to end communication between hosts [13]. The main features of TCP are flow control, error detection, reliability and congestion control. Multiple TCP variants have been introduced each of which differ in Congestion Control Mechanism. In our testbed we loaded various TCP variants pre-installed on Linux kernel and tested them by generating traffic through multiple sources.

We primarily used two TCP flavours- Cubic and Westwood. We carried out the tests in both non SDN as well as SDN environment.

In Cubic, window size is dependent on time rather than acknowledgement receipt in westwood. Thus Cubic performed much better in both environments. Also congestion is achieved much earlier in sdn environment due to extra control packets generated by controller which adds to over increasing traffic.

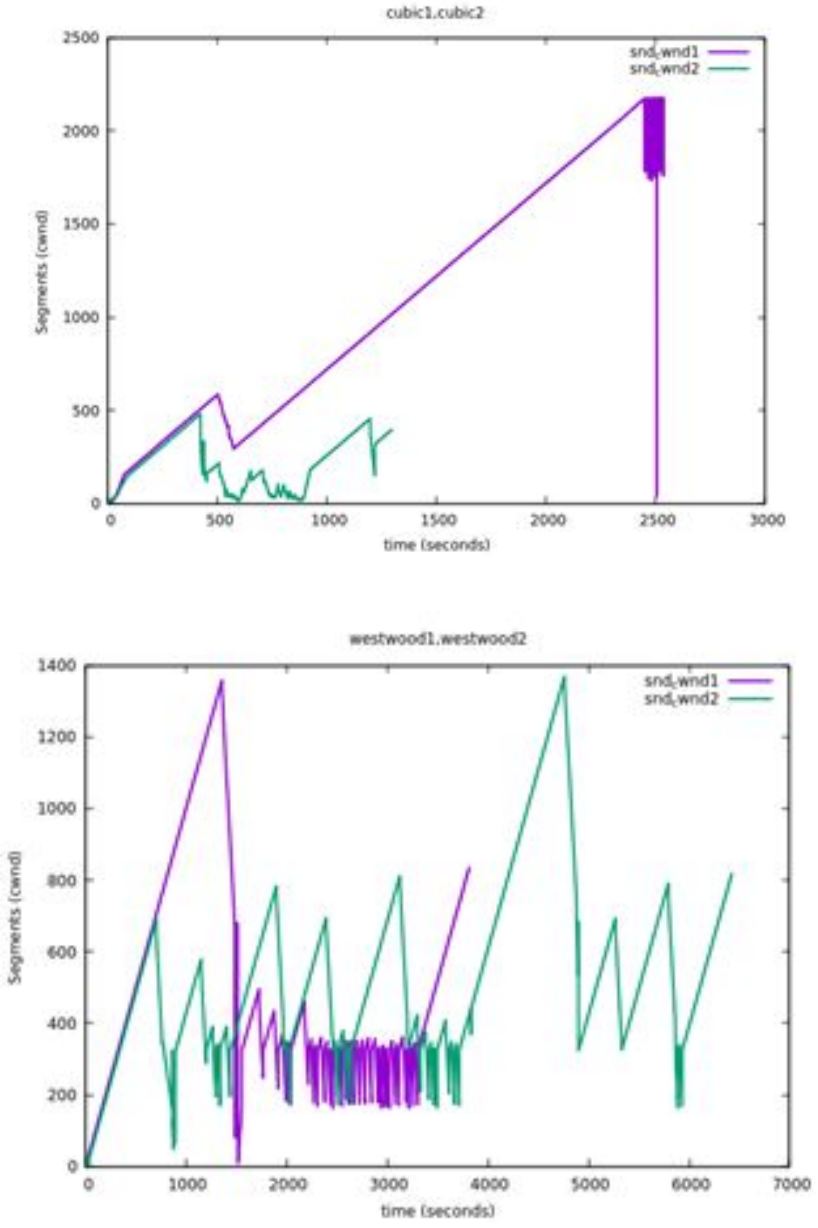

Figure 10: Results of TCP congestion test

5.3.3 Iperf Network Performance test

The HP switches were evaluated using the open-source iperf tool. The results obtained in SDN and Non-SDN mode are given as follows.

```
Connecting to host 10.0.0.12, port 5201
[  4] local 10.0.0.15 port 35969 connected to 10.0.0.12 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   112 MBytes   942 Mbits/sec    0    236 KBytes
[  4]   1.00-2.00   sec   111 MBytes   935 Mbits/sec    0    236 KBytes
[  4]   2.00-3.00   sec   111 MBytes   935 Mbits/sec    0    247 KBytes
[  4]   3.00-4.00   sec   111 MBytes   934 Mbits/sec    0    247 KBytes
[  4]   4.00-4.97   sec   108 MBytes   937 Mbits/sec    0    247 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-4.97   sec   555 MBytes   936 Mbits/sec    0             sender
[  4]   0.00-4.97   sec  0.00 Bytes  0.00 bits/sec                  receiver
```

Figure 11: Results of Iperf test in Non-SDN

```
Connecting to host 10.0.0.31, port 5201
[  4] local 10.0.0.32 port 33562 connected to 10.0.0.31 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   346 KBytes  2.84 Mbits/sec   24   5.66 KBytes
[  4]   1.00-2.00   sec   127 KBytes  1.04 Mbits/sec   21   5.66 KBytes
[  4]   2.00-3.00   sec   127 KBytes  1.04 Mbits/sec   17   5.66 KBytes
[  4]   3.00-4.00   sec  0.00 Bytes  0.00 bits/sec   15   7.07 KBytes
[  4]   4.00-5.00   sec   127 KBytes  1.04 Mbits/sec   11   5.66 KBytes
[  4]   5.00-6.00   sec   127 KBytes  1.04 Mbits/sec   19   5.66 KBytes
[  4]   6.00-7.00   sec  0.00 Bytes  0.00 bits/sec   20   5.66 KBytes
[  4]   7.00-8.00   sec   127 KBytes  1.04 Mbits/sec   15   5.66 KBytes
[  4]   8.00-9.00   sec  0.00 Bytes  0.00 bits/sec   20   4.24 KBytes
[  4]   9.00-10.00  sec   127 KBytes  1.04 Mbits/sec   13   2.83 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  1.08 MBytes   909 Kbits/sec  175             sender
[  4]   0.00-10.00  sec   935 KBytes   766 Kbits/sec                  receiver

iperf Done.
```

Figure 12: Results of Iperf test in Non-SDN

6. Summary: Tasks Accomplished and Challenges faced

The tasks accomplished as a part of this project are described as follows (Some of these tasks were done as a team with Ameesha, Vidhi and Kapil in the SDN Lab):

- Literature survey regarding the approaches to co-operability of different controllers
- Analysis of scenarios where SDN is useful, discussion of various challenges in SDN
- Performance Analysis of Native vs Virtual Machine on KVM: System and Network
- Survey of various test beds in SDN and virtualized workload management
- Setting up a monitoring frame work capable of receiving all the stats of all the ports
- Setting up the Master-Slave architecture for co-existence of controllers
- Performance evaluation of our testbed using publicly available datasets
- Setting up of final Test Bed on Network rack

Throughout this process of learning and experimenting, the challenges which were faced both at an individual level as well as along with the team members are described as follows:

- Incompatibility of libvirt and Open vSwitch
- Incomplete topology monitoring due to legacy hardware switch
- Link Layer Discovery Protocol (LLDP) message inconsistency in SDN and legacy networks
- Very low bandwidth in HP switches when configured in SDN mode due to the OpenFlow agent
- HP Switches provide an indirect way, by separating controller interface and openflow VLAN, to configure in SDN mode
- Incompatibilities between OVS and HP Switch

7. References

[1] Ankit Rao, Shrikant Auti," High Availability and Load Balancing in SDN Controllers", International Journal of Trend in Research and Development, vol. 3, issue 2, ISSN: 2394-9333, 2016, pp.310-314.

[2] Cui Chen-xiao1 and Xu Ya-bin1, "Research on Load Balance Method in SDN" International Journal of Grid and Distributed Computing, vol. 9, no. 1, 2016, pp.25-36

[3] Diego Kreutz, "Software-Defined Networking: A Comprehensive Survey", pp 1-60, 2014

[4] Definition of Software-Defined Networking ONF. https://www.opennetworking.org/sdn-resources/sdn-definition Accessed: 21/11/2016

[5] 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., 38(2):69–74, March 2008.

[6] N. Feamster, J. Rexford and E. Zegura "The Road to SDN", ACM, December 30, 2013.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G.Parulkar, L.Petterson, J.Rexford, S.Shenker and J.Turner, "OpenFlow: Enabling Innovation in Campus Networks" ACM SIGCOMM April, 2008

[8] Niels L. M. van Adrichem," OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks", IEEE 2014

[9] OFELIA Control Framework (OCF): a set of software tools for testbed management. http://github.com/fp7-ofelia/ocf. Accessed: 15/10/2016

[10] OpenFlow Switch Specification: Version 1.3.0. https://www. opennetworking.org /images/stories/downloads/sdn-resources/specifications/OpenFlow/openflow-spec-v1.1.0 .pdf. Accessed: 19/09/2016.

[11] Open Networking Foundation. https://www.opennetworking.org/. Accessed: 19/08/2016.

[12] SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains. https://tools.ietf.org/html/draft-yin-sdn-sdni-00. Accessed: 23/11/2016

[13] Sanjeev Patel, Kritika Rani: Comparative performance analysis of TCP-based congestion control algorithms. IJCNDS 17(1): 61-75 (2016)

[14] Ryu: Component-based Software Defined Networking Framework. https://osrg.github.io/ryu/. Accessed: 23/11/2016