Performance Analysis of Software Defined Network Controller Architecture – A Simulation Based Survey

Madhukrishna Priyadarsini, Dr. Padmalochan Bera, Rohan Bhampal
School of Electrical Sciences
Indian Institute of Technology Bhubaneswar, India
Email:{*mp18,plb and rb13*}*@iitbbs.ac.in*

Abstract—Software Defined Networking (SDN) has proclaimed significant attention from network researcherers and industries in recent years. Software defined network is an effort to design, build, and manage networks that separates the networks control (brains) and data (muscle) planes. It enables the network control to become directly programmable and the underlying architecture to be abstracted for applications and network services. SDN platform provides various advantages such as programmability, potential for task virtualization and schedulability and easy management of the network. However, it introduces new challenges towards scalability and performances, security hardening, cross-layer communication protocols, etc. It is important to understand and analyze the performances and limitations of SDN for implementation and deployment in live network environments and applications. This paper reports a number of technologies, models and tools to evaluate the performance metrics of SDN controllers along with simulation results. It also states the working procedure of various controllers like NOX, NOX-MT, POX, Rosemary, FloodLight, OpenDayLight, Beacon, Maestro. This study will help in designing efficient architecture and control algorithms for SDN controllers.

Keywords: SDN, Performance, Controller, Parameters, iPerf, Cbench, NOX, POX, Beacon.

1 Introduction

The recent trends of digitization of data in large scale calls significant chnages or disruption in the communication technologies and network platforms towards scalable, configurable, performance-centric, pay-per-use and demand driven application execution environment. The traditional network, computing infrastructure, and protocol stack may not be suitable to provide adequate solutions to such growing and heterogeneous demands. This leads towards a divergent approach in network systems architecture, called Software-Defined Networking (SDN) [1]. Software Defined Networking is a layered network framework that provides unprecedented programmability, automation, and network control by ramifying the control plane and the data plane of the network [2]. In SDN architecture, network intelligence and states are logically centralized, and the underlying network infrastructure is abstracted for network applications. Network architectures in which the control plane is separated from the data plane have been gaining popularity with scope of reserach and developments. One of the major features of this approach is that it provides a more structured software environment for developing network-wide abstractions while potentially simplifying the data plane.

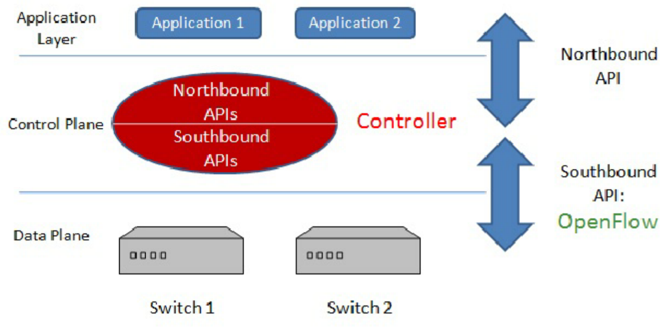

Fig. 1. Software Defined Network Architecture

SDN offers various prevalences, such as centralized and decentralized control of multiple cross-vendor network elements, mainly data plane platforms with a common API abstraction layer for all SDN-enabled equipment. It dewindles the complexity of network configuration and operation that is achieved by automation of high level configuration and forwarding behaviour of network elements [7]. SDN acquiesces easy deployment of new protocols and network-services which leads towards high operation abstraction. SDN infrastructure can adjust to the specific user application running on it through control plane, which abundantly improves the user experience.

TABLE 1
Comparison Between SDN and Conventional Networking

	SDN	Conventional Networks
Feature	Decoupled data, control plane and programmability	A n/w protocol per problem, complex network control
Configuration	Automated configuration with centralized structure	Manual configuration with errors
Performance	Dynamically controlled, globally available with cross layer information	Static configuration with limited information
Innovation	Easy s/w up-gradation ,sufficient testing environment,quick deployment	Difficult h/w implementation,limited testing environments

However, SDN, has its disadvantages: the added flexibility and functionality require additional overhead on the equipment, and as a result there are performance forfeiture in terms of processing speed and throughput. This is not suggesting that the overall performance is necessarily decreasing; many network services and tasks that were executed by the end-nodes or by the control layers of the network systems can be executed by the SDN-enabled equipment in a simpler and quicker way.

Benefits of SDN over conventional Networking is described in table 1.

SDN performance is evaluated considering number of network parameters like:
.Throughput
.Latency
.Jitter
.Bandwidth
.Workload

In this paper we present study of number of parameters that affect the performance of SDN controller, how variation of those parameters influence the performance. Implementation of set of tools that are used to measure performance of SDN controller with simulation results. Study of various controllers with their working procedures and how those controllers enhance the performance are also discussed.

The rest of the paper is organized as follows. Section 2 presents the motivation and objectives. We described the parameters, tools, technologies, models that are used for enhancement of performance of SDN Controllers in Section 3. Section 4 presents the study of different SDN controllers like NOX, NOX-MT, POX, Rosemary, FloodLight, Beacon, Maestro, OpenDayLight. Section 5 is Disscussion along with simulation results.We conclude in Section 6 with further proceedings.

2 Motivation and Objectives

Today, a large number of heterogeneous applications are being executed in the backbone networks of any organizations or publicly accessible networks. On the other hand, the requirements of the organizations is becoming heterogeneous and stringent in terms of cost and QoS. In such scenarios, increase of traffic and varying requirements trivially cause degradation in bandwidth of network, response time, throughput and work load of SDN controllers. In order to maintain and enhance the performance of a network, the analysis of these parameters of SDN Controllers in heterogeneous and live networks with varying network inputs is necessary. The analysis will help in desigining efficient architecture and control algorithms for SDN controllers. This motivates the present study as reported in this document.

The main obbjectives of this work are as follows:

1) To analyze the performance of SDN controllers and the network devices and to extract a comprehensive report with recommendations towards maintaining performance parameters.
2) To improve the existing network function execution and scheduling algorithms of controllers.
3) To design new algorithms for enhancement of SDN Controllers performance.
4) To design a architectural blueprint of an efficient SDN controller with network function virtualization and emulation of the same using apprpriate software package or in-house designed tools.

3 Methodology

In this section, we first describe various important concepts used for enhancing the performance of the SDN controllers. The commonly used concepts are:
1) Number and location of the controller
2) Logical realization of the control plane this can be either centralized or distributed.

In first case logically controllers are placed according to load and also number of controllers are increased. If we consider a case study: suppose for a particular geographical location load on the controller is increased, then at that location more number of controllers should be introduced. As a result the co-relation between the application, control, data layers are maintained, response time, processing time, bandwidth are also synchronized.

In second case there is a centralized and distributed view of the control plane. For a particular load it gives accurate response time, processing time and also the bandwidth is maintained. control plane is logically distributed. Lets assume a scenario in a large cluster: number of nodes are more, so load on a single controller is also very huge. To manage this situation distributed control plane concept is introduced.

The above mentioned concepts are having following deficits
1) Protocol of communication between controllers: Inorder to utilize data center resources efficiently, SDN controllers need to communicate among themselves which is frequently observed in multi-SDN cotoller environment.
Inter-SDN Controller Communication can be implemented using the vertical or the horizontal approach. In vertical approach the master controller has global view of network and also orchestrates every individual network controller in each domain.In the horizontal approach, the SDN controllers establish peer-to-peer communication. Each controller sends request for communication or information to its peers, in own as well as other domains in the network. IN these two approaches number of controller increases. If number of controller increases then communication between themselves is one major issue and overhead increases in the network.
2) Synchronization cost: In muti-SDN controller architecture the controller which is logically centralised must be physically distributed among multiple controllers for performance, scalability and fault-tolerance. In order to provide a consistent network view with control applications, the network states must be synchronized among different controllers. Synchronization between controllers and their response time will cost a lot.

Various tools are used to calculate throughput, latency, response time, jitter in SDN. As these parameters influence the performance greatly, so using tools we can able to figure out performance of the Software Defined Network. Some tools are mentioned with their implementation prospective.

1) OFLOPS:(OpenFLow Operations Per Second) OFLOPS is a standalone controller that benchmarks various aspects of an OpenFlow switch. It quantifies a switch's performance by implementing a modular framework. It permits development of tests for openflow switches such as CPU utilization, packet counters etc. It determines the hindrance between the switch and the remote control application.
2) Cbench:(controller benchmarker) Cbench is the commonly used tool for SDN controller benchmarking. It tests OpenFlow controllers for new flows by generating packet-in events. It emulates a bunch of open-flow switches that are connected to a controller and then computes the performance metrics like throughput, response time, latency of an SDN controller. In data plane it provides simple 3-stage pipelining concept (hashing, filtering, counting). In control plane it provides the measurement library that automatically configures the pipeline and allocates resources for different measurement task.

TABLE 2
SDN Controllers

Controller	Language	Created By
NOX	C++	Niciria Networks
NOX-MT	C++	Niciria Networks
POX	Python	Murphy McCauley
Maestro	Java	Stanford University
Beacon	Java	Rice University
FloodLight	Java	Big Switch Networks
Rosemary	Python+Java	SRI International
OpenDayLight	Java	Cisco and OpendayLight

3) iPerf: IPerf is used for measuring TCP and UDP bandwidth performance. By harmonizing various parameters and characteristics of the TCP/UDP protocol, the user is able to perform a number of tests that provide an acumen on the network's bandwidth availability, delay, jitter and data loss [20].IPerf is a command line program which accepts a number of different options, making it very easy to use. It enables the deployment of various networking services, network protocols, and architectures.

4 Study of Controllers

Here we considered 7 controllers with their working procedures, architectures and implementations. In table 2 all the controllers with their implementation language are stated.
1) NOX: It was initially developed side-by-side with OpenFlow and was the first OpenFlow controller. It uses the concept of single threading and virtualization [4]. NOX is the basic level controller for performance enhancement. NOXs network view includes the switch-level topology; the locations of users, hosts, middleboxes, and other network elements, and the services (e.g., HTTP or NFS) being offered. It also includes all bindings between names and addresses, but does not include the current state of network traffic [13].There exists multithreaded successor of NOX which is known as NOX-MT. NOX-MT uses well-known optimization techniques (e.g., I/O batching) to improve the baseline performance.
2) NOX-MT: NOX-MT, a marginally modified multi-threaded successor of NOX, to show that with simple tease NOXs throughput and response time is significantly improved. The techniques used to optimize NOX are utterly well-known including: I/O batching to minimize the overhead of I/O, porting the I/O handling harness to Boost Asynchronous I/O (ASIO) library (which simplifies multi-threaded operation), and used a fast multiprocessor-aware malloc implementation that scales well in a multi-core machine [4]. It does not address many of NOXs performance deficiencies, including: heavy use of dynamic memory allocation and redundant memory copies on a per request basis. Addressing these issues would significantly improve NOXs performance.

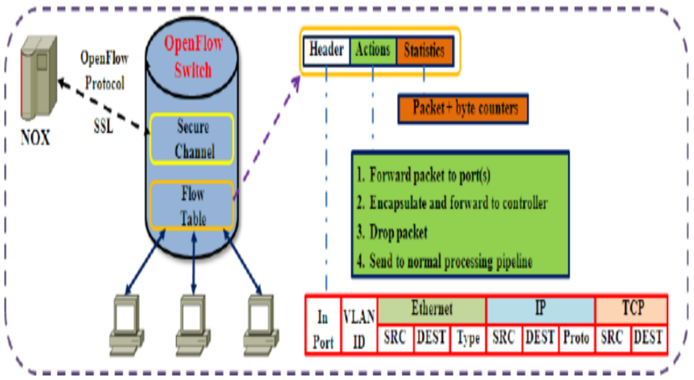

Fig. 2. NOX Controller Architecture

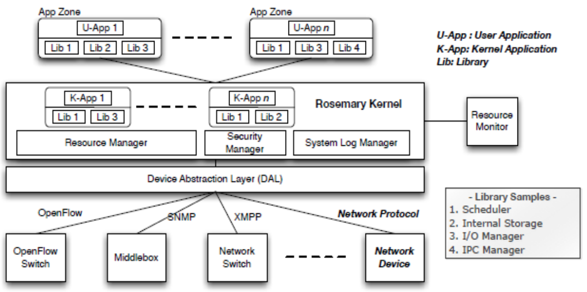

Fig. 4. Rosemary Controller Architecture

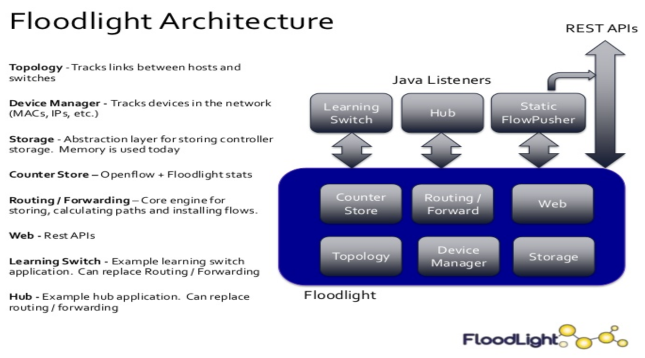
Floodlight Architecture
Topology - Tracks links between hosts and switches
Device Manager - Tracks devices in the network (MACs, IPs, etc.)
Storage - Abstraction layer for storing controller storage. Memory is used today
Counter Store – Openflow + Floodlight stats
Routing / Forwarding – Core engine for storing, calculating paths and installing flows.
Web - Rest APIs
Learning Switch - Example learning switch application. Can replace Routing / Forwarding
Hub - Example hub application. Can replace routing / forwarding
REST APIs
Java Listeners
Learning Switch
Hub
Static FlowPusher
Counter Store
Routing / Forward
Web
Topology
Device Manager
Storage
Floodlight
FloodLight

Fig. 3. FloodLight Controller Architecture

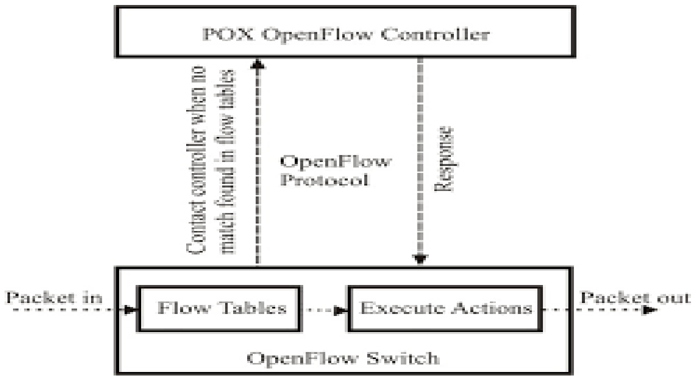

Fig. 5. POX Controller Architecture

3) Floodlight: The Floodlight is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller. It is supported by a community of developers including a number of engineers fromBig Switch Networks. Floodlight can handle mixed OpenFlow and non-OpenFlow networks [19]. Floodlight is designed to work with the thriving number of switches, routers, virtual switches, and access points that support the OpenFlow standard.It peaks that the link between switch and controller is of primary importance for the gross performance of the network. If there is some problem in the link then directly it affects the performance of the controller. Also it cites that high latency is another cause of degradation of network performance. High throughput with high latency causes bad performance [14].

4) Rosemary: Rosemary distinguishes itself by its brew of process containment, resource utilization monitoring, and an application permission structure, all designed to prevent common failures of network applications from halting operation of the SDN Stack [5].Rosemary can sustain more than 10 million flow requests per second.

It consists of four main components: (i) data abstraction layer (DAL), (ii) Rosemary kernel, (iii) System libraries, and (iv) A resource monitor. DAL encapsulates underlying hardware devices (i.e., network devices) and forwards their requests to the upper layer (i.e., Rosemary kernel). A key objective of DAL is to mobilize requests from diverse network devices. Rosemary kernel provides basic necessary services for network applications, such as resource control, security management, and system logging.

5) POX: POX is an OpenFlow networking software platform written in Python. POX started life as an OpenFlow controller, but now functions as an OpenFlow switch, and also be useful for writing networking software.POX provides OpenFlow interface and reusable components for path selection, topology discovery, etc [12]. POX, which enables rapid development and prototyping, is becoming more commonly used than NOX.

6) Maestro: Maestro, which keeps a simple single threaded programming model for application programmers of the system, yet enables and manages parallelism as a service to application programmers. It exploits parallelism in every corner together with additional throughput optimization techniques to scale the throughput of the system.

The programming framework of Maestro provides interfaces for: Introducing new customized control functions by adding modularized control components. Maintaining network state on behalf of the control components. Composing control components by specifying the execution sequencing and the shared network state of the components. It provides the control com-

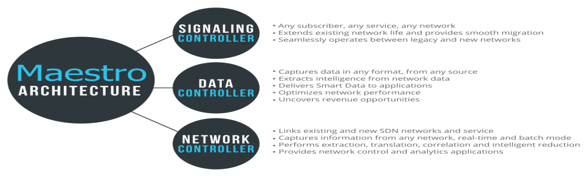
Maestro
ARCHITECTURE
SIGNALING CONTROLLER
• Any subscriber, any service, any network
• Extends existing network life and provides smooth migration
• Seamlessly operates between legacy and new networks
DATA CONTROLLER
• Captures data in any format, from any source
• Extracts intelligence from network data
• Delivers Smart Data to applications
• Optimizes network performance
• Uncovers revenue opportunities
NETWORK CONTROLLER
• Links existing and new SDN networks and service
• Captures information from any network, real-time and batch mode
• Performs extraction, translation, correlation and intelligent reduction
• Provides network control and analytics applications

Fig. 6. Maestro Controller Architecture

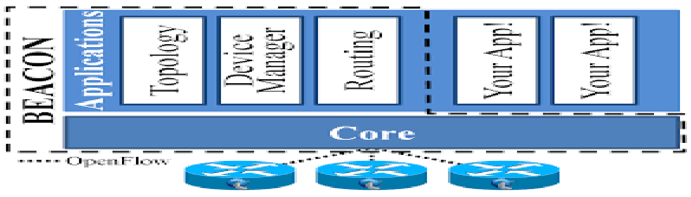

Fig. 7. Beacon Controller Architecture

ponents for realizing either a learning switch network, or a routed network using OpenFlow switches.

7) Beacon: Beacon is a fast, cross-platform, Java-based OpenFlow controller that supports both event-based and threaded operation [9]. Beacon runs on many platforms, from high end multi-core Linux servers to Android phones. Beacon architecture includes event handling, reading openflow messages, writing openflow messages to enhance performance [15].

5 Discussion

In this section we describe the variation of throughput, latency, jitter using iperf and cbench tool. By varying different parameters we found variation in the performance in client as well as server. Both throughput and latency are tested in TCP and UDP mode using iPerf. In UDP mode jitter is calculated. We also focuses on NOX, POX and Beacon controller implementation. Simulation results are provided. NOX and POX performances are compared as well as their performance with Beacon is stated. Beacon is the modern controller which is used now-a-days and it provides much higher througput than NOX and POX.

Cbench is a controller benchmarking tool used to calculate latency. It also calculates the switches performance that are connected to the controller. Cbench output is provided with switch's performance.

Graphical results for each simulation are provided.

iPerf execution output:

All the iperf execution simulation results are given in fig.8-11 by considering the performance parameter bandwidth and jitter.

Fig.12 shows the openflow configuration of controllers and switches. Using this configuration we

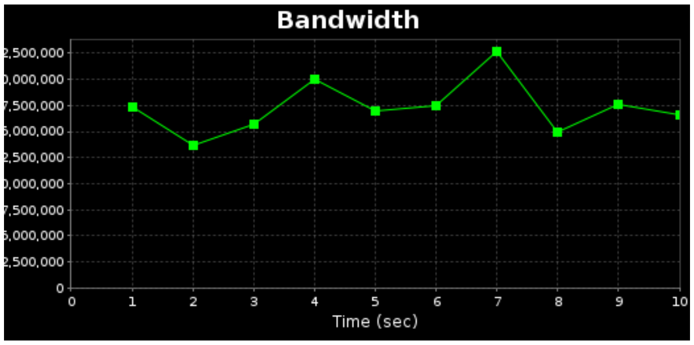

Fig. 8. Bandwidth of client in TCP mode.Client connecting to 172.17.16.80, TCP port 5001, TCP window size: 2564 KByte, Time:10 sec, Bandwidth:17291407 Kbits/sec.

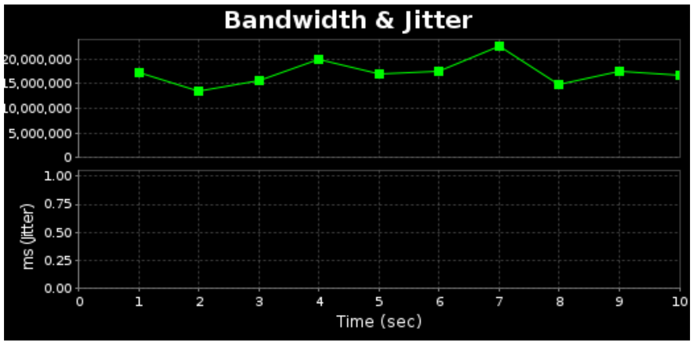

Fig. 9. Bandwidth of server in TCP mode.Server listening on TCP port 5001, TCP window size: 85.3 Kbyte, Time:10 sec, Bandwidth:17255580 Kbits/sec.

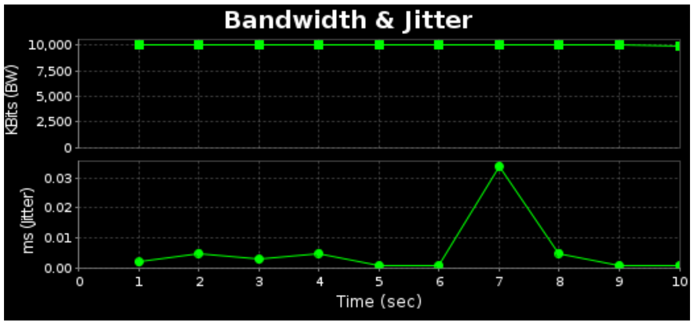

Fig. 10. Banwidth and Jitter of server in UDP mode.Server listening on UDP port 5001, Receiving 1470 byte datagrams, UDP buffer size:160 Kbyte,Time:10 sec, Bandwidth:9998 Kbits/sec, Jitter:0.001 ms, Lost/Total Datagrams:0/8502, 1 datagram received out-of-order

found the NOX and POX controller performance.
Here we have studied performance and behaviour of POX controller like hub behavior with tcpdump, Benchmark hub controller, Sending Openflow messages with POX, Parsing Packets with the POX packet libraries.
NOX controller is the basic controller used in SDN. Its performance and behaviour is tested here. Results

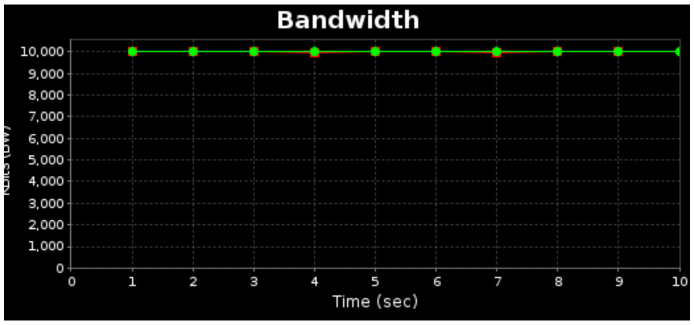

Fig. 11. Bandwidth of client in dual testing mode. Client connecting to 172.17.16.80, UDP port 5001 Sending 1470 byte datagrams UDP buffer size: 160 Kbyte, Time:10 sec, Bandwidth:10004 Kbits/sec, Sent:8503 packets, 1 datagram received out-of-order. Server report::Bandwidth:10004 Kbits/sec, 1 datagram received out-of-order.

of NOX and POX controller performance based on response time are given.

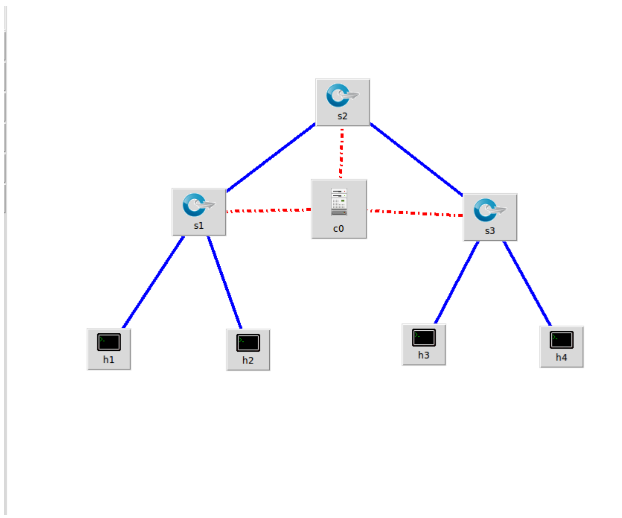

Fig. 12. Openflow configuration using controller and switches

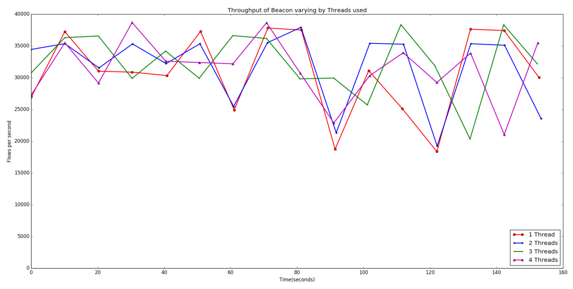

Fig. 13. Throughput of Beacon controller with 4 threads

From figure 15 and 16 of NOX and POX response time it is cleared that POX is providing more responses per second than NOX. In case of throughput, NOX

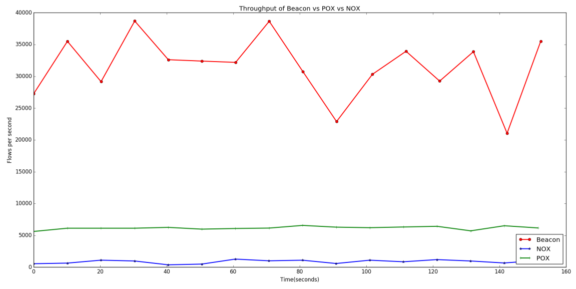

Fig. 14. Throughput Comparison of NOX, POX and Beacon controller

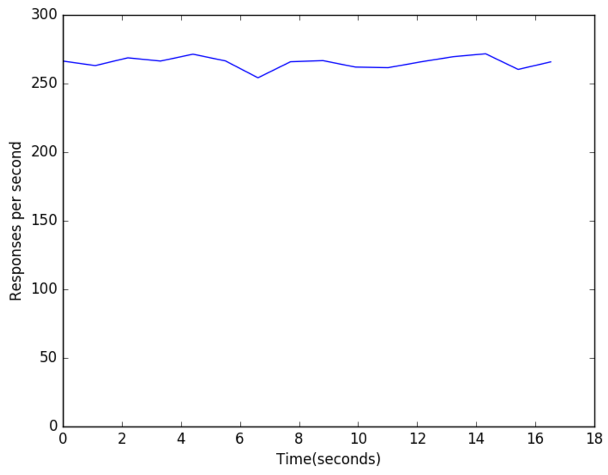

Fig. 15. Response Time in POX Controller

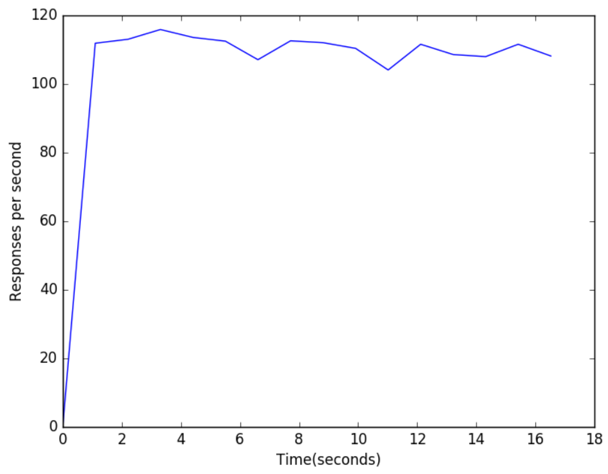

Fig. 16. Response Time in NOX Controller

initially gives higher throughput than POX but with increment of time its performance falls down as load increases. From all the simulations we observe that POX is the advanced version of NOX which provides better performance. Beacon provides higher response time and throughput rather than NOX and POX as its working principle depends upon multiple threads. NOX and POX works efficiently in homogeneous traffic while for heterogeneous traffic Beacon is the best.

6 Conclusion

Performance enhancement is the measure issue in SDN which has greater impact on the SDN security as well as its architecture. All the tools and controller's performance are tested and the results are compared. During high traffic the load is more and performance is less as compared in less load. So to enhance the performance, controller's should designed in such a way that in high load they should maintain their performance. This leads to our further work that is to design better controller's algorithm which will enhance the performance of SDN controller.

References

[1] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, Haiyong Xie, "A Survey on Software-Defined Networking", *IEEE Communication Surveys and Tutorials, Vol. 17, No. 1, First Quarter 2015.*

[2] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, On Controller Performance in Software-defined Networks, *in Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, Berkeley, CA, USA, 2012, pp. 1010.*

[3] Fouad Benamrane, Mouad Ben mamoun, and Redouane Benaini, "Performances of OpenFlow-Based Software- Defined Networks: An overview", *Journal of Networks, Vol. 10, No. 6, June 2015.*

[4] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, NOX: towards an operating system for networks, *ACM SIGCOMM Comput. Commun. Rev., vol. 38, no. 3, pp. 105110, 2008.*

[5] Seungwon Shin et al., "Rosemary: A Robust, Secure, and High-Performance Network Operating System", *CCS14, November 3, 2014, Arizona, USA.*

[6] Zuhran Khan Khattak, Muhammad Awais and Adnan Iqbal, "Performance Evaluation of OpenDaylight SDN Controller", *IEEE Transaction, 2014.*

[7] Yimeng Zhao, Luigi Iannone and Michel Riguidel, "On the Performance of SDN Controllers: A Reality Check", *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), 2015.*

[8] Marcial P. Fernandez, "Evaluating OpenFlow Controller Paradigms", *ICN 2013 : The Twelfth International Conference on Networks.*

[9] David Erickson, "The Beacon OpenFlow Controller", *HotSDN13, August 16, 2013, Hong Kong, China.*

[10] Alexander Gelberger, Niv Yemini, Ran Giladi, "Performance Analysis of Software-Defined Networking (SDN)", *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems.*

[11] Wolfgang Braun and Michael Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices", *Future Internet 2014, 6, 302-336; doi:10.3390/fi6020302.*

[12] "About Pox, http://www.noxrepo.org/pox/about-pox/.

[13] About Nox, http://www.noxrepo.org/nox/about-nox/.

[14] "Floodlight project, http://www.projectfloodlight.org /floodlight/.

[15] "What is Beacon? https://openflow.stanford.edu/display/ Beacon/Home.

[16] OpenDayLight project, http://www.opendaylight.org/.

[17] Open Network Operating System, http://onosproject.org/.

[18] Michael Jarschel, Frank Lehrieder, Zsolt Magyari, and Rastin Pries. A Flexible OpenFlow-Controller Benchmark, *In Software Defined Networking (EWSDN), 2012 European Workshop on, pp. 48-53. IEEE, 2012.*

[19] Syed Abdullah Shah, Jannet Faiz, Maham Farooq, Aamir Shafi, and Syed Akbar Mehdi, "An architectural evaluation of SDN controllers, *In Communications (ICC), 2013 IEEE International Conference on, pp. 3504-3508. IEEE, 2013.*

[20] "iperf", https://iperf.fr/.