Performance Analysis of Software Defined Network Controller Architecture – A survey

Madhukrishna Priyadarsini, IIT Bhubaneswar, Dr. Padmalochan Bera, IIT Bhubaneswar, Rohan Bhampal, IIT Bhubaneswar

Abstract—Software Defined Networking (SDN) has gained significant attention from network researcher community and industries in recent years. Software defined network is an approach to design, build, and manage networks that separates the network's control (brains) and data (muscle) planes enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services.The 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, Rosemary, NOX-MT, FloodLight. This study will help in designing efficient architecture and control algorithms for SDN controllers.

Keywords: SDN, Performance, Controller, Parameters, iPerf, Cbench.

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 triggered the emergence of a different approach to network systems architecture, called Software-Defined Networking (SDN) [1]. Software Defined Networking is a layered network architecture that provides unprecedented programmability, automation, and network control by decoupling 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 decoupled 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.

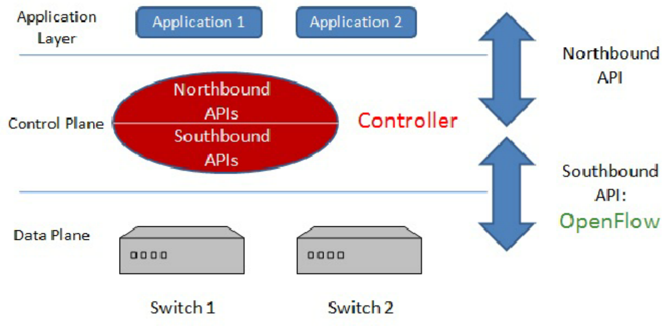


Fig. 1. Software Defined Network Architecture

SDN offers many advantages, 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 also reduces the complexity of network configuration and operation that is achieved by automation high level configuration is translated into specific forwarding behaviour of network elements [7]. SDN allows easy deployment of new protocols and network-services as a result of high operation abstraction. Increased control granularity in SDN allows a per flow definition with a high granularity policy level. SDN infrastructure can adjust to the specific user application running on it via the control plane, which greatly improves the user experience.

Some key concepts of SDN system architecture are described below:

Business applications:

This refers to applications that are directly consumable by end users. Possibilities include video conferencing, supply chain management and customer relationship management.

Network and security services:

This refers to functionality that enables business applications to perform efficiently and securely. Possibilities include a wide range of security capabilities such as

firewalls, IDS/IPS and DDoS protection.
Pure SDN switch:
In a pure SDN switch, all of the control functions of a traditional switch (i.e., routing protocols that are used to build forwarding information bases) are run in the central controller. The functionality in the switch is restricted entirely to the data plane.
Hybrid switch:
In a hybrid switch, SDN technologies and traditional switching protocols run simultaneously. A network manager can configure the SDN controller to discover and control certain traffic flows while traditional, distributed networking protocols continue to direct the rest of the traffic on the network.
Hybrid network:
A hybrid network is a network in which traditional switches and SDN switches, whether they are pure SDN switches or hybrid switches, operate in the same environment.
Northbound API:
Relative to Figure 1, the northbound API is the API that enables communications between the control layer and the business application layer. There is currently not a standards-based northbound API [27].
Southbound API:
Relative to Figure 1, the southbound API is the API that enables communications between the control layer and the infrastructure layer. Protocols that can enable this communications include OpenFlow, the extensible messaging and presence protocol (XMPP) and the network configuration protocol [21].
Part of the confusion that surrounds SDN is that many vendors don't buy in totally to the ONF definition of SDN. For example, while some vendors are viewing OpenFlow as a foundational element of their SDN solutions, other vendors are taking a wait and see approach to OpenFlow. Another source of confusion is disagreement relative to what constitutes the infrastructure layer. To the ONF, the infrastructure layer is a broad range of physical and virtual switches and routers. One of the current approaches to implementing network virtualization relies on an architecture that looks similar to the one shown in Figure 1, but which only includes virtual switches and routers.

Software Defined Networking, however, has its disadvantages: the added flexibility and functionality require additional overhead on the equipment, and as a result there are performance penalties in terms of processing speed and throughput. This is not to say that the overall performance is necessarily decreasing; many network services and tasks that were executed by the end-nodes or by the control or management layers of the network systems can be executed by the SDN-enabled equipment in a simpler and quicker way, thereby improving the overall performance of the networking tasks. Irrespective of these things there are few more need of SDN performance [27], [35]. Some of them are described below:
1) Optimize Network Device Virtualization.
2) Traffic Engineering/ Bandwith Management.
3) Capacity Optimization.
4) Load Balancing.
5) High Utilization.
6) Fast Failure Handling.

SDN have the following emerged almost likely set of opportunities

- Support the dynamic movement, replication and allocation of virtual resources.
- Ease the administrative burden of the configuration and provisioning of functionality such as QoS and security.
- More easily deploy and scale network functionality.
- Perform traffic engineering with an end-to-end view of the network.
- Better utilize network resources.
- Reduce OPEX.
- Have network functionality evolve more rapidly based on a software development lifecycle.
- Enable applications to dynamically request services from the network.
- Implement more effective security functionality.
- Reduce complexity.

The performance of SDN networks highly depends on the control layer, which in turn, is constrained by the scalability of centralized controllers. Indeed, all the transactions in the control plane are involved with controllers. Switching devices need to request controllers for packet forwarding rules reactively when the first packet of each flow arrives [27]. Rule update and network status collection also involve in frequent communication between controllers and switching devices. In this aspect, bandwidth consumption and latency of frequent communication affect control layer scalability significantly.

The result of SDN perfomance can be used in various applications like

- QoS management
- Link flow usage
- Anomaly detection
- Traffic matrix estimation
- Traffic engineering

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 represents SDN Performance Issues, Benefits, Challenges . Section 3 presents the motivation and objectives. We described the parameters, tools, technologies, models that are used for enhancement of performance of SDN Controllers in Section 4. Section 5 presents the study of different SDN controllers like NOX, NOX-MT, POX, Rosemary, FloodLight, Beacon, Maestro, OpenDayLight. Section 6 is Disscussion along with simulation results.We conclude in Section 7 with further proceedings.

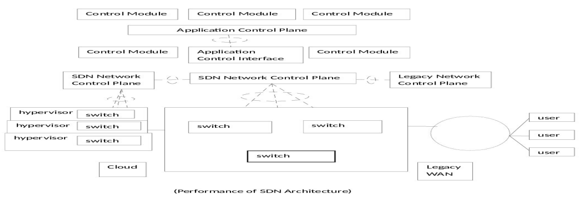

Control Module
Control Module
Control Module
Application Control Plane
Control Module
Application Control Interface
Control Module
SDN Network Control Plane
SDN Network Control Plane
Legacy Network Control Plane
hypervisor
switch
hypervisor
switch
hypervisor
switch
switch
switch
switch
user
user
user
Cloud
Legacy WAN
(Performance of SDN Architecture)

Fig. 2. Performance of SDN Architecture

2 SDN Performance: Issues, Benefits, Challenges

Being one of the most popular subjects in ICT domain SDN has not yet been reached on its exact destination. In fact, a lot of different issues have surfaced over the last couple of years, each of which has its own existence. In this section, we first present a generally accepted definition of SDN Performance, and then outline a set of key benefits and challenges in performance of SDN. Performance of SDN architecture is shown in fig.2.

2.1 Definition of SDN Performance:

In general performance means the action or process of performing a task or function.SDN performance indicates how the architecture helps in performing the tasks. SDN performance depends upon combinely 3 parameters.Those are
. Performance of Dataplane.
. Performance analysis of SDN Controller.
. Modelling and performance evaluation of SDN Architecture.

Performance of Data plane

Data plane is otherwise known as forwarding plane.Forwards traffic to the next hop along the path to the selected destination network according to control plane logic. Data plane packets go through the router. Data plane actually moving the packets based on what we learned. The data plane is the actual movement of the customers data packets over the transit path we learned in the control plane stage [28]. Performance of data plane depends upon analysis of throughput and processing delays of openflow enabled forwarding devices. Testbed are used to measure data plane performance of devices like routers, switches etc.

Performance of Control plane

Controllers in SDN are "brains" of the network. It is the application that acts as strategic control point in the SDN network, manage flow control to the switches/routers 'below' (via southbound APIs) and the applications and business logic 'above' (via northbound APIs) to deploy intelligent networks. Recently, as organizations deploy more SDN networks, the Controllers have been tasked with federating between SDN Controller domains, using common application interfaces, such as OpenFlow and open virtual switch database (OVSDB) [27]. Performance of SDN controller is measured by analysing softwares used within the controllers in realistic environment. And also some network parameters with hardwares are considered like throughput,latency, CPU and RAM. Performance analysis of SDN controller holds some features like

- Generated control message per switch.
- Topology emulation.
- Incoming data packets can be arbitarily distributed.

Modelling and Performance evaluation of SDN architecture

The OpenFlow concept of flow-based forwarding and separation of the control plane from the data plane provides a new flexibility in network innovation. While initially used solely in the research domain, OpenFlow is now finding its way into commercial applications. However, this creates new challenges, as questions of OpenFlow scalability and performance have not yet been answered. Based on measurements of switching times of current OpenFlow hardware, basic models are used for the forwarding speed and blocking probability of an OpenFlow switch combined with an OpenFlow controller. Models can be used to estimate the packet sojourn time [1] and the probability of lost packets in such a system and can give hints to developers and researchers on questions how an OpenFlow architecture will perform given certain parameters.

2.2 Benefits:

In comparison to conventional networking, SDN encourages innovation by providing a programmable network platform to implement, experiment, and deploy new ideas, new applications, and new revenue earning services conveniently and flexibly [20]. High configurability of SDN offers clear separation among virtual networks permitting experimentation on a real environment. Progressive deployment of new ideas can be performed through a seamless transition from an experimental phase to an operational phase. Benefits of SDN over conventional Networking is described in table 1.

2.3 Challenges:

SDN performance is evaluated considering number of network parameters like:
.Throughput
.Latency
.Jitter
.Bandwidth
.Workload
.Response Time
In a complicated SDN scenario performance evaluation by considering these parameters is very difficult. Let us take an example: In a distributed view, protocol of communication between controllers is a tiddy job. Synchronization cost of communication between controllers is also a challenge for SDN. During these two above mentioned scenarios all the network parameters varies, and evaluation

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

of these network parameters is a complicated task.
SDN's abstraction and separation of the network and control layer adds complexity for network monitoring and visibility tools [35]. Virtualization means that IT must monitor not only the physical network, but also the virtual network and hypervisor traffic. Integration of SDN networks with legacy physical networks will require tools to model and measure performance and latency, as well as provide comprehensive network mapping that reflects both environments [36].

3 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.

In this section five motivating examples are given which shows that SDN not only save time but also boost network's performance.
1) Connecting Branches: With the increased use of proprietary products and software there is also an increased network security risk with each device and location. Every time a port is added it meant analyzing, inspecting, and securing, which for hundreds of ports meant weeks of work. This is especially cumbersome for companies with branches spread out around the world. SDN allows to install remote access points in every branch locations and in employee's residences without any on-site configuring.
2) Wifi calling: While we might not be minute conscious when making phone calls (though many of us still remember the days of only calling someone after 9pm because it was free) there are times when our cell phone signal may be slow (or non-existent), especially in old buildings with thick concrete construction.Avoiding these dead zones becomes easy using wireless LAN for Wi-Fi calling. Using a mobility controller in data center allows us to program thousands of access points with ease.We can configure the settings to add Wi-Fi calling to the top of our priority list as well. Dead signal zones become a thing of the past (provided we have wifi.)
3) Deployment of Mobile Business Application: Skype for Business has gained a lot of traction recently, allowing colleagues to communicate using chat, voice, video, screen sharing, and file transferring in one application.It syncs with Microsoft Exchange making 4-digit extensions a thing of the past. Skype for Business shows extreme promise in enterprise communications. SDN creates ease in this otherwise cumbersome process. Imagine trying to configure every switch to talk to the Skype server, yikes.After the initial identification of the traffic on network, it will push out the settings to the other access points as to what apps to prioritize, including which device and the individual behind the device.
4) Safeguard Network: Network access controls should be improving the intelligence and safety of our network.For example, if our firewall detects a threat or infected mobile device on the network, it can adapt and adjust the settings of that specific device to protect the network and other connected devices from a potential problem.Preventing a major problem with thousands of devices by restricting the responsible device immediately saves time and money, as well as saving you from a massive headache.
5) Dynamic Interconnection: The IoT (internet of things) offers almost endless array of new business opportunities through the implementation of intelligent networking.With so much potential to increase productivity, decrease operational costs and uncover new ways to differentiate ourself from the competition, leveraging the IoT should be at the top of every enterprise mobility strategy.However, being that the internet of things is still relatively new there are some unknowns when it comes to network security and just how they will perform or affect other areas of your network.With network software controls and SDN, we can get applications to not only work as they were designed but guarantee that they're secure too.Today's tech industry is all about increasing efficiency, productivity, and security, with the least amount of headaches for those implementing these new solutions. To be successful we have to be prepared, that's why we always say, "Plan twice, deploy once".

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.

4 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 [27]
2) Logical realization of the control plane – this can be either centralized or distributed [27].

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 plane, control plane and data plane 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. Let's 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: To utilize data center resources efficiently, SDN controllers need to communicate. In a multi-SDN controller environment, each controller needs to be connected to the neighboring controller.
Approaches to Inter-SDN Controller Communication Inter–SDN controller communication can be implemented using the vertical or the horizontal approach [39].
Vertical Approach:
In the vertical approach, there is a master controller over the individual network controllers [42]. The master controller has a global view of the network across all connected SDN domains and can orchestrate the configuration in each domain.
Horizontal Approach:
In the horizontal approach, the SDN controllers establish peer-to-peer communication. Each controller can request for information or connections from its peers, that is, SDN controllers from other domains in the network. This is also called the SDN east-west interface [42]. IN these two approaches number of controller increases. If number of controller increases then communication between themselves is one issue and also overhead increases

2) Synchronization cost: The logically centralised control plane in Software-Defined Networks (SDN) must be physically distributed among multiple controllers for reasons of performance, scalability and fault-tolerance. However, this means that the network state must be synchronised among the different controllers to provide control applications with a consistent network view.Synchronization between controllers and their response time will cost a lot.

Two analytical models are used for analysis of performance issues.
1) Queuing Theory:
Queueing theory studies workloads in a variety of network traffic scenarios and offers an assortment of tools allowing for characterizing network performance. Queueing theoretic modelling techniques contribute to accurate performance evaluation of SDN-based production networks that are exposed to new user demands continually. In networking, the limitations of scale and bottlenecks of a particular network can be determined with an alytical models (without simulation or physical deployment). It allows to evaluate probability of dropping packets and packet delays in the system. This theory helps to improve the performance by considering the packet delays in the network [43].
2) Network Calculus Formalism:
Network calculus is a theory dealing with queueing type problems encountered in computer networks, with particular focus on quality of service guarantee analysis. It allows to analyse delay and queue size boundaries of SDN switches and controllers. As a result controllers and switches performance is improved and so as SDN performance. An essential idea of network calculus is to use min-plus algebra and max-plus algebra [44] to transform non-linear queueing systems into linear systems that are analytically tract able.

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) It is a standalone controller that benchmarks various aspects of an OpenFlow switch. Oflops implements a modular framework for adding and running implementation-agnostic tests to quantify an switch's performance. It permits development of tests for open-flow switches such as CPU utilization, packet counters etc [24]. It determines the bottleneck between the switch and the remote control application.
2) Cbench:(controller benchmarker) Cbench [29] is the most commonly used tool for SDN controller benchmarking. It is a program for testing OpenFlow controllers by generating packet-in events for new flows. Emulates a bunch of open-flow switches connect to a controller and then computes the performance metrics like throughput, response time, latency of an SDN controller [25]. In data plane it provides simple 3-stage pipeline (hashing, filtering, counting). In control plane it provides the measure-

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

ment library that automatically configures the pipeline and it allocates resources for different measurement task [41].
3) iPerf: IPerf is a popular network tool that was developed for measuring TCP and UDP bandwidth performance. By tuning various parameters and characteristics of the TCP/UDP protocol, the user is able to perform a number of tests that provide an insight on the network's bandwidth availability, delay, jitter and data loss [45].IPerf is a command line program that accepts a number of different options, making it very easy to use.Here are iPerf's main features:
.Measures TCP bandwidth
.Reports on maximum segment size / maximum transmission unit .Support for TCP Window size
.Multi-threaded for multiple simultaneous connections
.Creates specific UDP bandwidth streams
.Measures packet loss
.Measures delay jitter
.Runs as a service or daemon
.Runs under Windows, Linux OSX or Solaris
4) ProGFE(Programmable Generic Forwarding Element): It is a flexible and reconfigurable network platform, which enables the deployment of various networking services, network protocols, and architectures. This platform is con- figured on the fly, via an XML-based API. ProGFEs opera- tion is based on IETFs ForCES (Forwarding and Control Element Separation)definition of Logical Functional Blocks (LFBs), where each LFB defines the way the ProGFE operates.It enables the deployment of various networking services, network protocols, and architectures.

5 Study of Controllers

Here we considered 8 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 has been the basis for research projects in the early exploration of SDN. It uses the concept of single threading and virtualization [4]. NOX is the basic level controller for performance enhancement. NOX's 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. The view includes all bindings between names and addresses, but does not include the current state of network traffic [13].

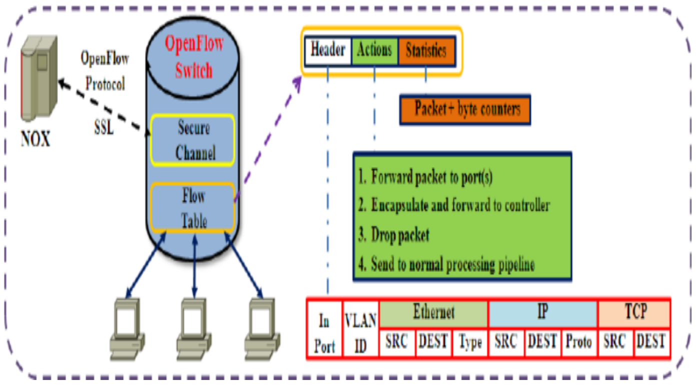
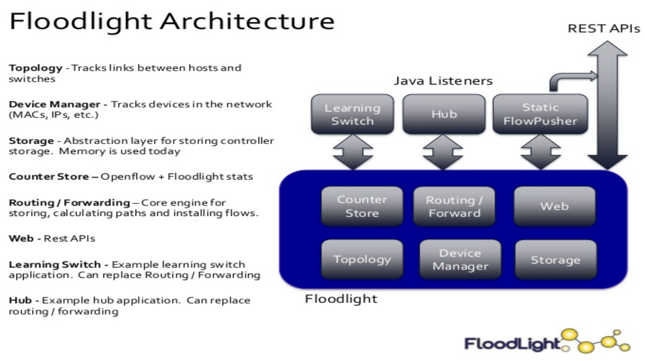


Fig. 3. NOX Controller Architecture

This choice of observation granularity provides adequate information for many network manage- ment tasks and changes slowly enough that it can be scalably maintained in large networks [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 slightly modified multi-threaded successor of NOX, to show that with simple tweaks NOX's throughput and response time is significantly improved. The techniques used to optimize NOX are quite 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 also used a fast multiprocessor-aware malloc implementa- tion that scales well in a multi-core machine. Despite these modifications, NOX-MT is far from perfect [4]. It does not address many of NOX's performance deficiencies, including but not limited to: heavy use of dynamic memory allocation and redundant memory copies on a per request basis, and using locking were robust wait free alternatives exist. Addressing these issues would significantly improve NOX's performance.
3) Floodlight: The Floodlight Open SDN Controller is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller. It is supported by a community of developers including a number of engineers from Big Switch Networks.. Floodlight can handle mixed OpenFlow and non-OpenFlow networks [19]. Floodlight is designed to work with the growing number of switches, routers, virtual switches, and access points that support the OpenFlow standard.It highlights that the link between switch and controller is of primary importance for the whole 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].
Feature Highlights Offers a module loading system that make it simple to extend and enhance. Easy to set up with minimal dependencies Supports a broad range of virtual- and physical- OpenFlow switches Can handle mixed OpenFlow and non-OpenFlow networks – it can manage multiple "islands" of OpenFlow hardware switches De-



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

Fig. 4. FloodLight Controller Architecture

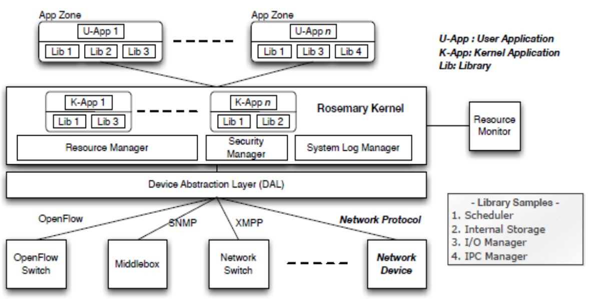


Fig. 5. Rosemary Controller Architecture

signed to be high-performance – is multithreaded from the ground up Support for OpenStack (link) cloud orchestration platform

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

ROSEMARY consists of four main components: (i) a data abstraction layer (DAL), (ii) the 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 marshal requests from diverse network devices.

The ROSEMARY kernel provides basic necessary services for network applications, such as resource control, security management, and system logging. They are basic services to operate network applications, and we design this component as thinly as possible to expose a clear view of the data plane to an application. ROSEMARY provides diverse libraries for applications, and each application can choose necessary libraries for its operations. When it selects libraries, each application is required to solicit permission from ROSEMARY kernel. The resource monitor (RM) is used to track the respective resource utilization of running applications and terminate misbehaving applications.

5) POX: Pox is a networking software platform written in Python. POX started life as an OpenFlow controller, but can now also function as an OpenFlow switch, and can be useful for writing networking software in general.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.

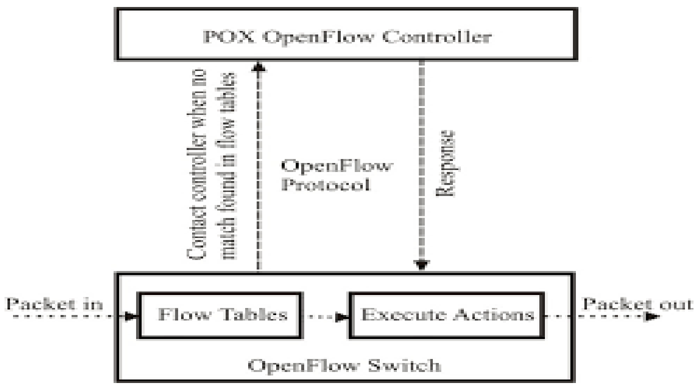


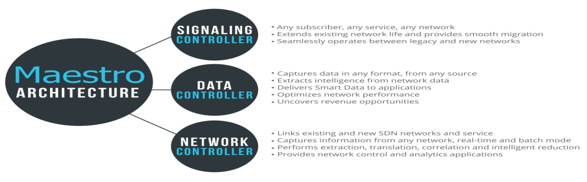
Fig. 6. POX Controller Architecture

The programming framework of Maestro provides interfaces for: Introducing new customized control functions by adding modularized control components [30]. 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. Maestro currently provides the control components for realizing either a learning switch network, or a routed network using OpenFlow switches. Some components such as the command line console, etc.

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 [38].

Beacon architecture includes event handling, reading openflow messages, writing openflow messages to enhance performance [15].

8) OpenDayLight: OpenDayLight(ODL) is an open



- Any subscriber, any service, any network
- Extends existing network life and provides smooth migration
- Seamlessly operates between legacy and new networks

- Captures data in any format, from any source
- Extracts intelligence from network data
- Delivers Smart Data to applications
- Optimizes network performance
- Uncovers revenue opportunities

- 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. 7. Maestro Controller Architecture

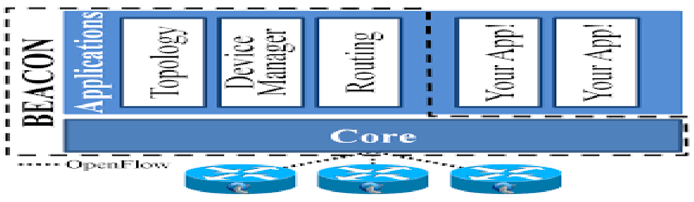


Fig. 8. Beacon Controller Architecture

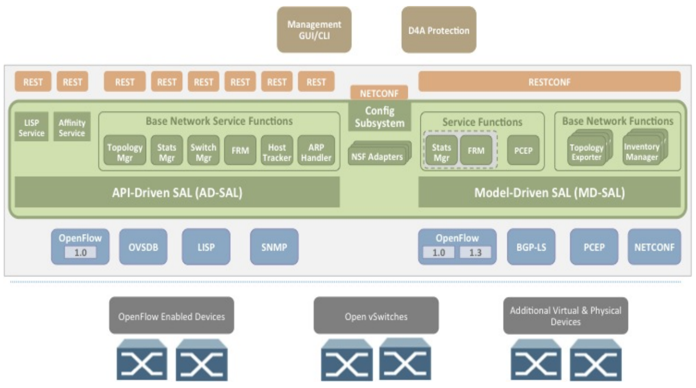


Fig. 9. OpenDayLight Controller Architecture

source SDN controller aimed at enhancing software-defined networking (SDN) by offering a community-led and industry-supported framework, which has been renamed the OpenDaylight Platform. It is open to anyone, including end users and customers, and it provides a shared platform for those with SDN goals to work together to find new solutions [6]. The OpenFlow protocol, considered the first SDN standard, defines the open communications protocol that allows the SDN Controller to work with the forwarding plane and make changes to the network. This gives businesses the ability to better adapt to their changing needs, and have greater control over their networks. The OpenDaylight Controller is able to deploy in a variety of production network environments. It can support a modular controller framework, but can provide support for other SDN standards and upcoming protocols. The OpenDaylight Controller exposes open northbound APIs, which are used by applications. These applications use the Controller to collect information about the network, run algorithms to conduct analytics, and then use the OpenDaylight Controller to create new rules throughout the network [37].

Various platform releases of OpenDayLight controllers are available starting from Hydrogen, Helium, Lithium, Beryllium (latest version).

6 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 also calculated. We also focuses on NOX and POX controller implementation. Simulation results are provided. NOX and POX performances are compared.

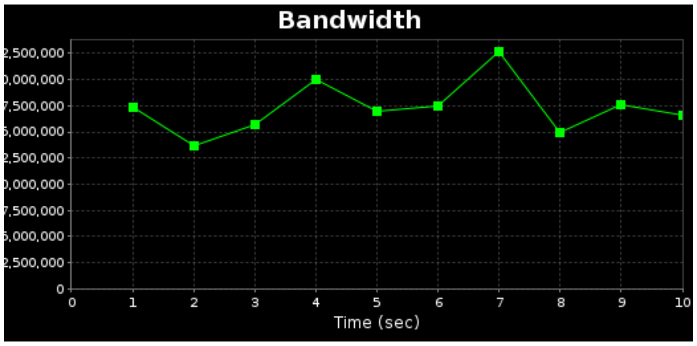


Fig. 10. Bandwidth of client in TCP mode.Client connecting to 172.17.16.80, TCP port 5001, TCP window size: 2564 KByte, Time:10 sec, Bandwidth:17291407 Kbits/sec.

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.9-13 by considering the performance parameter bandwidth and jitter.

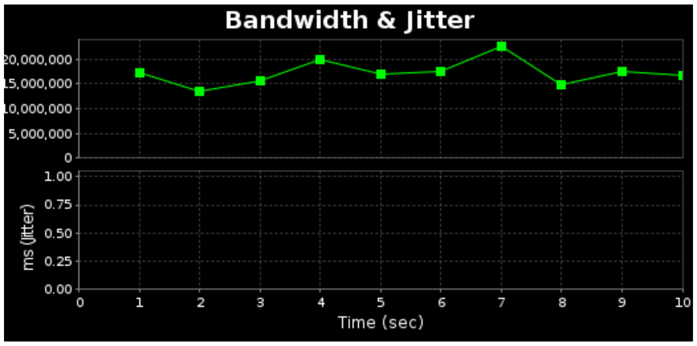


Fig. 11. Bandwidth of server in TCP mode.Server listening on TCP port 5001, TCP window size: 85.3 Kbyte, Time:10 sec, Bandwidth:17255580 Kbits/sec.

Cbench execution output:

When a fake switch polls on the socket it owns and receives no data. Cbench treats this as an error condition, spits the following to stderrand 'exit(1)'s.
cbench -p 54321
cbench: controller benchmarking tool
running in mode 'latency'
connecting to controller at localhost:54321
faking 16 switches offset 1 :: 16 tests each; 1000 ms per test with 100000 unique source MACs per switch
learning destination mac addresses before the test
starting test with 0 ms delay after features-reply ignoring first 1 "warmup" and last 0 "cooldown" loops connection delay of

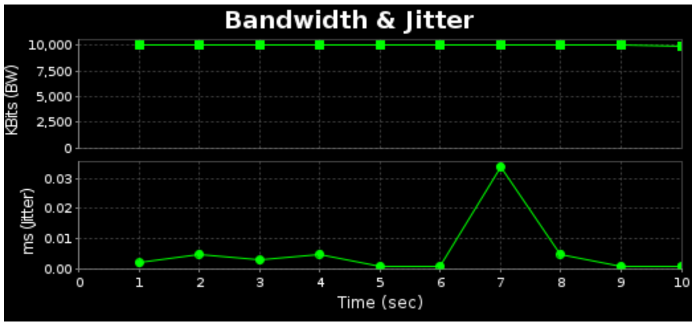


Fig. 12. Banwidth and Jitter of server in UDP mode.Server listening on UDP port 5001, Receiving 1470 byte datagrams, UDP buffer size:160 Kbyte,Time:10 sec, Bandwidth:9998 Kbits/sec, Jitter:0.001 ms, Lost/Total Datagrams:0/8502, 1 datagram received out-of-order

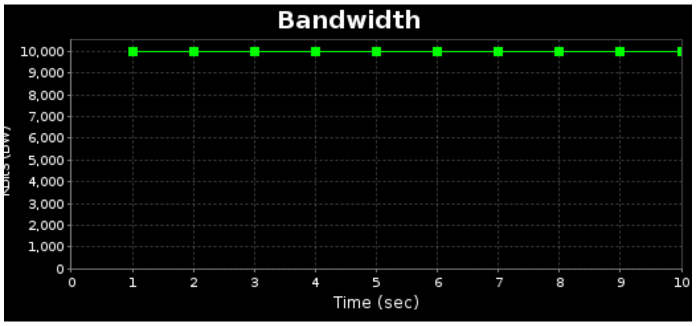


Fig. 13. Banwidth of client in UDP mode.Client connecting to 172.17.16.80, UDP port 5001 Sending 1470 byte datagrams, UDP buffer size: 160 Kbyte, Time:10 sec, Bandwidth:9998 Kbits/sec.

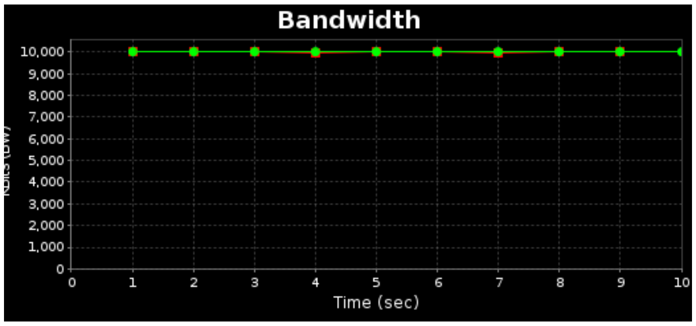


Fig. 14. Bandwidth of client in dual testing mode. Client connecting to 172.17.16.80, UDP port 5001 Sending 1470 byte datagrams UDP buffer size: 160 Kbyte, Time:10 sec, Bandwidth:10004 Kbits/sec, Sent:8503 packets, 1 datagram received out-of-order. Server report::Bandwidth:10004 Kbits/sec, 1 datagram received out-of-order.

```
0ms per 1 switch(es)
debugging info is off
controller msgbuf-read() = -1: msgbuf-read: Connection refused
... exiting
```

Output from simple controller in OpenFlow reference suite.

```
./cbench -p 54321
cbench: controller benchmarking tool
connecting to controller at localhost:54321
faking 16 switches :: 16 tests each; 1000 ms per test
starting test with 0 ms delay after features-reply
debugging info is off
16 switches: fmods/sec: 4661 4656 4655 4651 4649 4647 4645
4643 4640 4636 4634 4631 4625 4621 4617 4608 total =
74.218852 per ms
16 switches: fmods/sec: 4843 4843 4842 4842 4842 4842 4842
4842 4841 4841 4841 4841 4841 4841 4839 4838 total =
77.458909 per ms
16 switches: fmods/sec: 4649 4647 4647 4646 4646 4646 4646
4646 4646 4643 4642 4642 4642 4641 4641 4640 total =
74.308588 per ms
16 switches: fmods/sec: 4832 4832 4832 4832 4832 4831 4831
4831 4831 4831 4831 4831 4830 4829 4829 4828 total =
77.292227 per ms
16 switches: fmods/sec: 4784 4784 4784 4784 4784 4784 4784
4784 4784 4784 4784 4783 4783 4783 4783 4782 total =
76.537923 per ms
16 switches: fmods/sec: 4705 4703 4701 4698 4694 4690 4688
4685 4682 4681 4679 4677 4670 4668 4665 4661 total =
74.945651 per ms
16 switches: fmods/sec: 4813 4810 4805 4803 4802 4799 4797
4794 4791 4790 4786 4782 4781 4774 4770 4763 total =
76.659157 per ms
16 switches: fmods/sec: 4817 4813 4808 4805 4801 4797 4796
4796 4791 4790 4785 4779 4777 4771 4769 4765 total =
76.659463 per ms
16 switches: fmods/sec: 4676 4674 4669 4667 4666 4662 4660
4657 4654 4651 4649 4644 4641 4638 4634 4624 total =
74.465777 per ms
16 switches: fmods/sec: 4880 4878 4875 4872 4870 4867 4865
4863 4860 4857 4853 4851 4844 4844 4843 4841 total =
77.762378 per ms
16 switches: fmods/sec: 4795 4795 4794 4793 4793 4793 4793
4793 4793 4793 4793 4792 4792 4791 4791 4790 total =
76.683003 per ms
16 switches: fmods/sec: 4868 4868 4868 4867 4867 4867 4866
4866 4865 4865 4865 4864 4864 4864 4864 4863 total =
77.850144 per ms
16 switches: fmods/sec: 4780 4781 4781 4781 4780 4779 4779
4779 4779 4779 4779 4778 4778 4778 4777 4776 total =
76.463082 per ms
16 switches: fmods/sec: 4844 4844 4844 4844 4844 4844 4843
4842 4842 4842 4842 4842 4841 4841 4841 4840 total =
77.479845 per ms
16 switches: fmods/sec: 4847 4847 4847 4847 4848 4848 4848
4848 4847 4847 4847 4847 4847 4846 4846 4846 total =
77.552845 per ms
16 switches: fmods/sec: 4788 4788 4788 4787 4787 4787 4787
4786 4786 4786 4785 4784 4784 4784 4784 4784 total =
76.574311 per ms
RESULT: 16 switches 16 tests min/max/avg/stdev =
74218.85/77850.14/76432.01/1214.77 responses/s.
```

Fig.15 shows the openflow configuration of controllers and switches. Using this configuration we found the NOX and POX controller performance.
POX is a Python-based SDN controller platform geared towards research and education. 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 of NOX and POX controller performance based on response time are also given.

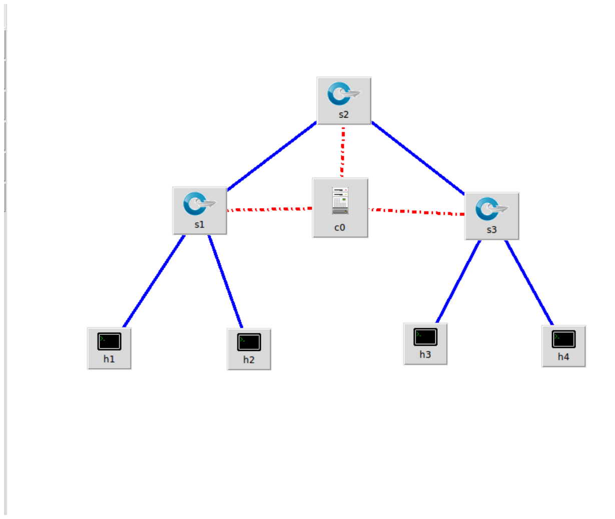


Fig. 15. Openflow configuration using controller and switches

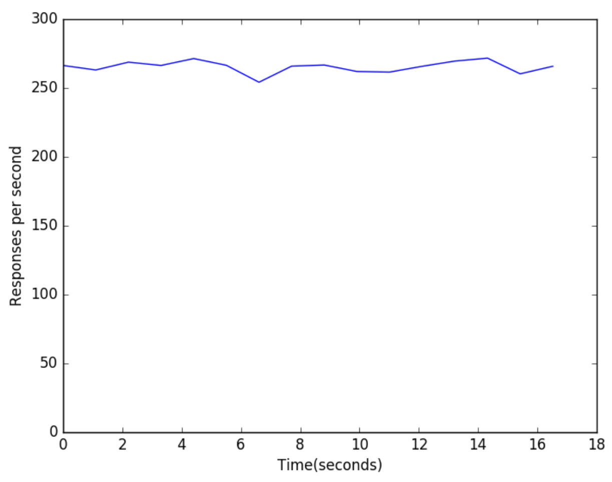


Fig. 16. Response Time in POX Controller

From figure 16 and 17 of NOX and POX response time it is cleared that POX is providing more responses per second than NOX. In case of throughput, NOX initially gives higher throughput than POX but with increment of time its performance falls down as load increases which is shown in figure 18. From all the simulations we observe that POX is the advanced version of NOX which provides better performance.

7 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

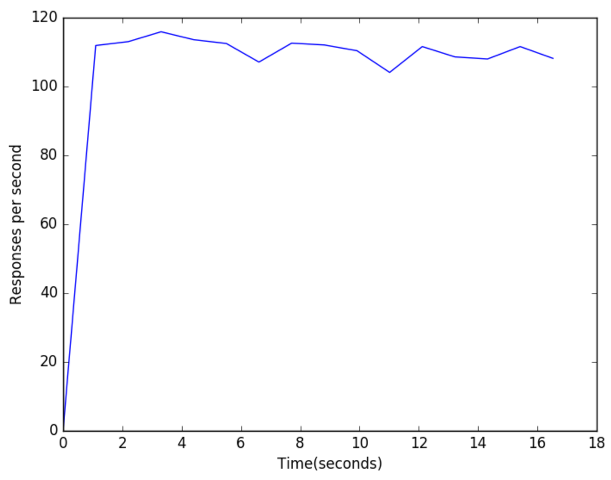


Fig. 17. Response Time in NOX Controller

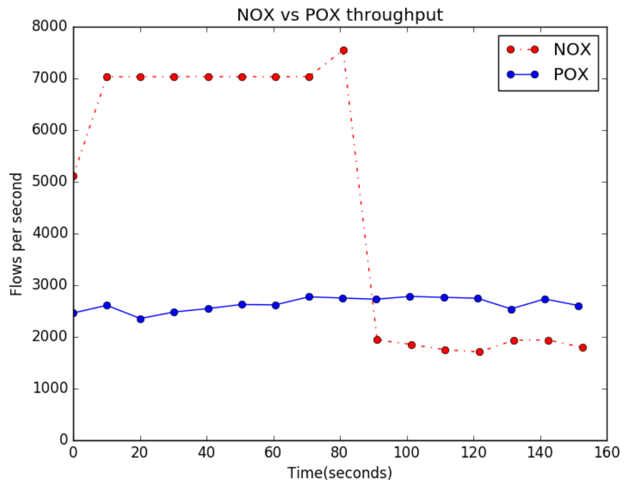


Fig. 18. Throughput Comparison of NOX and POX controller

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.

8 Acknowledgments

This report is done for the partial fulfillment of PhD degree in IIT Bhubaneswar. It provides basic understanding about performance of SDN controller, idea about various tools and their implementation.

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. 10–10.*

[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. 105–110, 2008*.

[5] Seungwon Shin et al., ”Rosemary: A Robust, Secure, and High-Performance Network Operating System”, *CCS’14, 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”, *HotSDN’13, 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] B. Heller, R. Sherwood, and N. McKeown, “The Controller Placement Problem”, *in Proceedings of the First Workshop on Hot Topics in Software Defined Networks, New York, NY, USA, 2012, pp. 7–12*.

[21] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, “Reliability-aware controller placement for Software Defined Networks”, *in 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), 2013, pp. 672–675*.

[22] K. Mahmood, A. Chilwan, O. N. Østerbø, and M. Jarschel, “On the Modeling of OpenFlow-based SDNs: The Single Node Case”, *ArXiv Prepr. ArXiv14114733, 2014*.

[23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: enabling innovation in campus networks”, *ACM SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, 2008*.

[24] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W.Moore, “OFLOPS: An Open Framework for Openflow Switch Evaluation,” *in Proceedings of the 13th International Conference on Passive and Active Measurement, Berlin, Heidelberg, 2012, pp. 85–95*.

[25] R. Sherwood and Y. KOK-KIONG, ”Cbench: an open-flow controller benchmarker”, *2010*.

[26] Y. HU, W. WANG, X. GONG, X. QUE, and S. CHENG, “On the placement of controllers in software defined networks”, *J.China Univ. Posts Telecommun., vol. 19, Supplement 2, pp. 92–171, Oct. 2012*.

[27] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, “Modeling and performance evaluation of an OpenFlow architecture”, *in Proc. ITC, 2011, pp. 1–7*.

[28] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. “Network innovation using openflow: A survey”, *IEEE Communications Surveys and Tutorials , Vol 16 No 1 (2013) pp 1-20*.

[29] “Cbench”, https://github.com/andi-bigswitch/oflops/tree/master/cbench.

[30] “Maestro platform”, http://code.google.com/p/maestro-platform/.

[31] S. H. Park, B. Lee, and et al., “A high-performance IO engine for SDN controllers”, *in 3rd European Workshop on Software Defined Networks (EWSDN), Sep. 2014, pp. 121–122*.

[32] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, “OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks”, *in NOMS, pp. 1–8, 2014*.

[33] ProgrammableFlow Controller. http://www.necam.com/SDN/ doc.cfm?t= PFlowController.

[34] C. Doerr, R. Gavrila, F. A. Kuipers, and P. Trimintzios, “Good practices in resilient internet interconnection,” *ENISA Report, Jun. 2012*.

[35] Dixit , A. Hao , F. Mukherjee , S. Lakshman , and Kompella, ”Towards an elastic distributed sdn controller”, *In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking(2013)*.

[36] Kreutz, D. Ramos, F. M., and Verissimo, ”Towards secure and dependable software-defined networks”, *In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (2013)*.

[37] OpenDaylight Performance Stress Tests Report. *SDN-NFV, 2015.*

[38] David Erickson, ”Using Network Knowledge to Improve Workload Performance in Virtualized Data Centers”, *PhD thesis, Stanford University, May 2013*.

[39] Andreas Voellmy et al., ”Scalable software defined network controllers”, *In SIGCOMM, 2012*.

[40] Volkan Yazıcı et al., ”Controlling a Software-Defined Network via Distributed Controllers”, *In NEM Summit, 2012*.

[41] R.Sherwood and K.-K.Yap. Cbench.http://www.openflow.org/ wk/index.php/Oflops/, *2012. [Online; accessed 10-April-2012]*

[42] Shie-Yuan Wang, Hung-Wei Chiu and Chih-Liang Chou, ”Comparisons of SDN OpenFlow Controllers over EstiNet: Ryu vs. NOX”. *The International Symposium on Advances in Software Defined Networks, April 19-24, 2015, Barcelona, Spain.*

[43] Jordan Ansell, Winston K.G. Seah, Bryan Ng and Stuart Marshall, ”Making Queueing Theory More Palatable to SDN/OpenFlow-based Network Practitioners”, *IEEE/IFIP NOMS 2016 Workshop: 8th International Workshop on Management of the Future Internet (ManFI)*

[44] Yuming Jiang, ”Network Calculus and Queueing Theory: Two Sides of One Coin”, *VALUETOOLS 2009, October 20-22, 2009 - Pisa, Italy.*

[45] ”iperf”, https://iperf.fr/.