*Proceedings of the 18th Mediterranean Electrotechnical Conference*
*MELECON 2016, Limassol, Cyprus, 18-20 April 2016*

# SDN Controllers: A Comparative Study

Ola Salman Imad H. Elhajj Ayman Kayssi Ali Chehab
Electrical and Computer Engineering Department
American University of Beirut
Beirut 1107 2020, Lebanon
{oms15, ie05, ayman, chehab}@mail.aub.edu

*Abstract*— **The control plane is an essential part of the SDN architecture, so it is very important to give proper attention to any proposal or design of an SDN controller. During the past few years, several controllers have been developed and several studies have been done to evaluate, compare and test the performance of these controllers. In this paper, new controllers are tested, such as ONOS and Libfluid-based controllers (raw, base), using Cbench, an OpenFlow testing tool. Even though the results show that MUL, Beacon, and Maestro (in latency mode) are the best performing controllers; however, the selection of the best-fitted controller should be based on several criteria, per user requirements.**

***Keywords— SDN; controller; Cbench; performance.***

## I. INTRODUCTION

Software Defined Network (SDN) is a new networking paradigm where the architecture moves from the traditional fully distributed model to a more centralized approach. This approach is also characterized by the separation of the data and control planes. The data plane includes the forwarding element (switches and routers) and the control plane includes the controller. The controller provides a high abstraction level of the forwarding elements management which is absent in today's networks. Therefore, the controller is a fundamental component of the SDN architecture that will contribute to the success or failure of SDN. Therefore, there is a need to assess and compare the different existing controllers in the market and research domains.

We are far from a controller (in some cases referred to as network operating system) which is hardware and language independent. Yanc, proposed in [1], can be considered as an example of such a system. However, today's controllers run as monolithic applications and they are highly tied to their programming languages (java, python, C, C++, etc.) Therefore, SDN specific languages such as Pyretic and Frenetic offering high-level abstraction languages were proposed to allow for application portability; however, they are in reality linked to a specific controller platform (POX).

Due to the importance of the controller within the SDN architecture and the diversity of architectures and implementations in the market and research fields, there is a need to assess and benchmark all these choices against different performance indicators.

From controllers processing few thousands of flows/s to ones processing few millions of flows/s, multiple approaches, languages, architectures, application program interfaces (APIs) and protocols have been implemented. In this paper, we will try to compare the most popular controllers and to test their performance using Cbench, which is an OpenFlow testing tool.

This paper is structured as follows: section II discusses the previous work related to SDN controller's comparison, section III describes the basic controller's modules, section IV presents the most important SDN controller's features, section V shows and analyzes the results of our controllers' performance test and finally we conclude in section VI.

## II. RELATED WORK

Several studies have been done in the aim of comparing SDN controllers. The work done in [2], one of the first providing a comparative study of the SDN controllers, considered a limited number of controllers (NOX-MT, Beacon and Maestro) focusing only on the controller performance. However, with time these controllers have been replaced by other controllers like POX, Ryu, FloodLight and OpenDaylight.

The study conducted in [3] provides a set of requirements that are the basis of the comparison between the controllers: TLS Support, virtualization, open source, interfaces, GUI, RESTful API, productivity, documentation, modularity, platform support, age, OpenFlow support, and OpenStack Neutron support. The comparison was done using a Multi-Criteria Decision Making (MCDM) method named Analytic Hierarchy Process (AHP) adapted by a monotonic interpolation/extrapolation mechanism which maps the values of the properties to a value in a pre-defined scale. By using the adapted AHP, five controllers (POX, Ryu, Trema, Floodlight, and OpenDaylight) have been compared, and "Ryu" was selected to be the best controller based on their requirements. However, the assumed scale is subjective and changing the scale would lead to a different conclusion.

An advanced study of SDN OpenFlow Controllers was performed in [4]. A comparative analysis of the effectiveness of the widely used SDN controllers: NOX, POX, Beacon, Floodlight, MuL, Maestro and Ryu is done. The authors used a tool named hcprobe. They concluded that the tested controllers present some security vulnerabilities, face difficulties to deal with average workload and that Beacon is the most performing controller based on the throughput test results. Due to the fact that new controllers are developed, this comparison must be extended to consider these controllers and to consider more controller features.

In [5] two operating modes are compared, proactive and reactive. The proactive mode has better performance than the reactive one due to the fact that the rules are loaded to the switch at the start not as in the reactive mode where the rules are loaded to the switch each time the switch receives a packet with no

978-1-5090-0058-6/16/$31.00 ©2016 IEEE

matching rule in its flow table. While this comparison sheds the light on an important factor in the performance comparison it is not sufficient to make a decision regarding the best featured controller.

Another study is done in [6] giving further considerations when designing a new controller. Two types of architectures are considered: static partitioning with static batching, and shared queue with adaptive batching. Beacon which uses static batching presented the highest performance in the experiment conducted with Maestro, NOX-MT and Floodlight. However, Maestro that uses the adaptive batching architecture presents the best latency records. So, the choice of the architecture depends on the required controller's behavior which is related to its application domain.

The fact that the controllers are software-based, the programming languages used to develop them are key. In [7], the choice of the programming language is demonstrated to have an impact on the portability and the performance of the controller. The authors claim that Java is the best choice given that it supports multithreading and it is a cross-platform language. Whereas Python presents issues with multithreading on the performance level and C, C++ have limitations with memory management and the .Net languages are dependent on the runtime platform (compatibility with Linux is not supported). Thus, they show that Beacon, which is java-based, has the best performance among several controllers (NOX, POX, Maestro, Floodlight, Ryu). However, the coexistence of these different languages till now shows that each language has certain characteristic that had not been replaced by the other languages.

Therefore, in [8] the issue of software aging is highlighted. Memory leak is the main issue studied and the comparison was done between two java based controllers (Floodlight and Beacon) to ensure the fairness of the study. The results showed that Beacon outperforms Floodlight with less memory consumption.

In this work, we will compare the most common open source controllers considering multiple criteria. This comparison is conducted with a number of controllers, considering the old and the most recent ones. Additionally, the comparison not only considers quantitative measures (throughput and latency), but also a wide set of other SDN-specific qualitative criteria. This comparison will facilitate the choice of the appropriate controller for a specific application domain.

## III. SDN Controller Basic Functions

As we mentioned before, SDN separates the data plane and the control plane. Essentially, the intelligence of the network is moved to the controller; all computations are done there and many applications and features can be added as needed. The basic modules are discussed in [9] where a lightweight carrier-grade controller is proposed. Link discovery module, topology module, storage module, strategy making module, flow table module and control data module are the basic SDN controller's modules. Essentially, two modules are responsible for providing the routing service: the topology manager and the link discovery modules. Collecting the physical link status information is the role of the link discovery module. There are two types of link discovery: link discovery between OpenFlow nodes (switches) and link discovery between an end host and an OpenFlow node. Essentially, the former uses the Link Layer Discovery Protocol (LLDP). Thus, the provided information by the link discovery module is used to build the neighbor database at the controller level. This database is managed by the topology manager module to build the global topology database which relies on the computation of the shortest (and alternate) path to any OpenFlow node or host. Any changes or link ruptures are trackable by the link discovery module. Therefore, the topology manager module has the role to maintain the topology information and to recalculate the routes in the network after any modification in the neighbor database.

## IV. SDN Controller Features

### *1) Cross platform compatibility*

Running cross-platform, allowing multithreading, being easy to learn, allowing fast memory access and good memory management are essential programming languages' characteristics. When choosing a certain controller, we have to take these factors into consideration because they affect the controller's performance and development speed. Python, C++, and Java are the most used languages for SDN controllers programming. In general, the Java coded controllers have the characteristic to run cross-platform and present good modularity, the C coded controllers provide high performance but lack high modularity, good memory management and good GUI and the Python coded ones lack real multi-threading handling.

### *2) Southbound Interfaces*

Southbound APIs allows control over the network. These APIs are used by the controller to dynamically make changes to forwarding rules installed in the data plane devices consisting of: switches, routers, etc. While OpenFlow is the most well-known of the SDN protocols for southbound APIs, it is not the only one available or in development. NETCONF (standardized by IETF), OF-Config (supported by the Open Network Foundation (ONF)), Opflex (supported by Cisco) and others are examples of southbound interfaces used for managing network devices. Additionally, some routing protocols such as IS-IS, OSPF, BGP are being also developed as southbound interfaces in some controllers in the aim to support hybrid networks (SDN and non-SDN ones) or to apply the traditional networking in a software-defined manner [10].

### *3) Northbound Interfaces*

The northbound APIs are used by the application layer to communicate with the controller. They are the most critical part in the SDN controller architecture. The most valuable benefit of SDN is derived from its ability to support and enable innovative applications. Because they are so critical, northbound APIs must support a wide variety of applications. These APIs allow also the connection with automated stacks such as OpenStack or CloudStack used for Cloud management. Recently, the ONF turned its focus to the SDN northbound API after working to standardize the southbound interface (OpenFlow). They have established a Northbound Working

Group that will write code, develop prototypes and look for standards creation [11]. Currently, the Representational State Transfer (REST) protocol seems to be the most used northbound interface and most of the controllers implement it.

*4) OpenFlow Support*

The OpenFlow protocol is a key enabler for software-defined networks. It was the first standardized southbound interface. It allows direct manipulation of the forwarding plane of OpenFlow switches [12]. When choosing an OpenFlow controller, we need to understand the OpenFlow functionality that the controller supports as well as the development roadmap to implement newer versions of OpenFlow, such as v1.3 or v1.4. One reason for needing to take this into consideration is that important functionality such as IPv6 support, for example, is not part of OpenFlow v1.0 but is part of the OpenFlow v1.3 standard.

*5) Network programmability*

Network programmability is the most important benefit of the SDN introduction to deal with the unprecedented management complexities in today's network with the explosion in the number of connected devices and the deployment of new services. Using the device-by-device paradigm to manage the high scale future networks will not be feasible. The old static way of managing network devices is time consuming, error prone and leads to inconsistencies. Software defined paradigm comes to hide these management difficulties introducing automation and dynamicity in the management process. Automated scripts can be run through command-line interfaces (CLIs) and applications can be deployed on top of the controller platform to perform predefined tasks and management functions. The controller support of network programmability relies essentially on its degree of integration of a wide number of northbound interfaces, a good graphical user interface and a CLI [13].

*6) Efficiency (Performance, Reliability, Scalability, and Security)*

The controller efficiency is an umbrella term used to refer to the different parameters – performance, scalability, reliability and security. Various metrics, such as number of interfaces a controller can handle, latency, throughput, etc. define what we call performance. Similarly, there are various metrics defining the scalability, reliability and security. Most of the work done to compare the controllers consider only the performance criteria. Additionally, the centralization of the control in the SDN scheme will present a serious challenge from the reliability and the performance perspectives. Thus, the distributed scheme, supported by some controllers, aims to cope with this issue [14].

*7) Partnership*

Being under good partnership oversight, an SDN controller will have chances to be maintained and enhanced for a long time [13]. The experience in the network and computer domains, and the economic capacity of the partner's organization are the main criteria biasing trust and use of products. Cisco, Linux Foundation, Intel, IBM, Juniper, etc. are examples of reputable organizations entering the SDN market and participating in controllers development.

Several surveys [15-21] have been done in the previous two years providing us with lists of the most commonly known controllers. Essentially, most of the listed features are taken into consideration when comparing the controllers.

The results are summarized in Table 1 below. We note that ONOS and OpenDaylight are the most featured controllers. These two Java coded controllers run cross-platforms and present high modularity using the OSGI container that allows loading bundles at runtime. Inheriting the power of Java/Javascript in the graphical user interfaces programming, they present good GUI feature. Being under the partnership of well-known network providers and research communities, they have a clear development plan and good documentation. Additionally, their support for the distributed scheme make them able to conduct a real SDN wide deployment.

The ONOS controller is principally designed for carrier networks. It gives them the ability to provide new SDN services along with their initial proprietary services. ONOS architecture is designed to maintain high speed and large scale networks. Its main distinguishing characteristic is its support for hybrid networks.

However, OpenDaylight was primarily datacenter focused but its latest release (Lithium) shows a capability to support different kinds of applications. Many southbound interfaces have been added (HTTP, COAP, PCEP, LACP, OpFlex, SNMP, etc.) and new modules have been implemented (IoT data broker (IoTDM), unified secure channel of USC, etc.). Thus, it is the first controller entering the IoT domain. Supporting a wide range of southbound interfaces and the distributed control paradigm, it seems to be the controller of the Internet of the Future. The support of OpenStack Neutron plugin in its architecture has also a remarkable importance when deploying software-defined edges responding to the Edge computing proliferation.

Ryu, presenting fair features, is a good choice for small businesses and research applications. Being Python coded, this controller presents facilities for applications and modules development. However, its lack of high modularity and its inability to run cross-platforms limit its wide use in real market applications.

In the next section, we will try to compare these controllers efficiency. Even though it will not be the only criterion to choose a controller, but processing requests at high rates with minimum latency is a key requirement of any controller.

## V. Performance Testing

### *A. Methodology*

Cbench is the most commonly used testing tool for SDN controllers [22]. Cbench essentially runs in two modes: throughput and latency modes. In the throughput mode, it sends as many packets as possible to compute the maximum number of packets handled by the controller. In the latency mode, Cbench sends a packet and waits for the reply to compute the time taken to process a single packet by the controller. Cbench

*Table 1: Feature based comparison of the most popular open source SDN controllers*

| | Programming Language | GUI | Documentation | Modularity | Distributed/ Centralized | Platform Support | Southbound APIs | Northbound APIs | Partner | Multithreading Support | OpenStack Support | Application Domain |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ONOS** | Java | Web Based | Good | High | D | Linux, MAC OS, And Windows | OF1.0, 1.3, NETCONF | REST API | ON.LAB, AT&T, Ciena, Cisco, Ericsson, Fujitsu, Huawei, Intel, Nec, Nsf.Ntt Comunnication, Sk Telecom | Y | N | Datacenter, WAN and Transport |
| **Open-Day-Light** | Java | Web Based | Very Good | High | D | Linux, MAC OS, And Windows | OF1.0, 1.3, 1.4, NET-CONF/ YANG, OVSDB, PCEP, BGP/LS, LISP, SNMP | REST API | Linux Foundation With Memberships Covering Over 40 Companies, Such As Cisco, IBM, NEC | Y | Y | Datacenter |
| **NOX** | C++ | Python + QT4 | Poor | Low | C | Most Supported On Linux | OF 1.0 | REST API | Nicira | NOX _MT | N | Campus |
| **POX** | Pyth on | Python + QT4 | Poor | Low | C | Linux, MAC OS, And Windows | OF 1.0 | REST API | Nicira | N | N | Campus |
| **RYU** | Pyth on | Yes | Fair | Fair | C | Most Supported On Linux | OF 1.0, 1.2, 1.3, 1.4, NETCONF, OF-CONFIG | REST For Southb ound | Nippo Telegraph And Telephone Corporation | Y | Y | Campus |
| **Beacon** | Java | Web Based | Fair | Fair | C | Linux, MAC OS, And Windows | OF 1.0 | REST API | Stanford University | Y | N | Research |
| **Maestro** | Java | - | Poor | Fair | C | Linux, MAC OS, And Windows | OF 1.0 | REST API | RICE, NSF | Y | N | Research |
| **Flood-Light** | Java | Web/ Java Based | Good | Fair | C | Linux, MAC OS, And Windows | OF 1.0 , 1.3 | REST API | Big Switch Networks | Y | N | Campus |
| **Iris** | Java | Web Based | Fair | Fair | C | Linux, MAC OS, And Windows | OF 1.0, 1.3, OVSDB | REST API | ETRI | Y | N | Carrier-Grade |
| **MUL** | C | Web Based | Fair | Fair | C | Most Supported On Linux | OF 1.4, 1.3, 1.0, OVSDB, OF-CONFIG | REST API | Kulcloud | Y | Y | Datacenter |
| **Runos** | C++ | Web Based | Fair | Fair | D | Most Supported On Linux | OF 1.3 | REST API | ARCCN | Y | N | WAN, Telecom and Datacenter |
| **Lib-Fluid** | C++ | - | Fair | Fair | - | Most Supported On Linux | OF 1.0, 1.3 | - | ONF | Y | N | - |

and the tested controller can be run on the same machine or on different machines.

Thus, due to the limitation of having a 10 Gbps link, we run our tests on a single machine. The machine used to perform the experiments has an Intel® Core™ i7-3630QM CPU @ 2.40GHZ x 8 (8 cores). 16 GB of memory was available. Operating system was Ubuntu 14.04 LTS-64 bit. We used the command below for running tests:

**./cbench –c localhost –p 6633 –l 14 -m 10000 –M 1000 –s 8 –t**

Where:

| | |
|---|---|
| -c | controller (IP or hostname) |
| -p | controller port number |
| -l | loops per test |
| -m | test time per second |
| -M | number of mac addresses per switch |
| -s | number of switches |
| -t | throughput mode |

### *A. Analysis of Results*

The tests were conducted for the listed controllers except Runos, which does not function with the current Cbench version even though it supports OpenFlow 1.3. It is important to note that OpenDaylight, as stated in [23], has problems with Cbench. So we have done the OpenDaylight's tests using Wcbench, which is a Phyton wrapper around Cbench.

Performing the tests in two modes, varying the number of switches, and in the throughput mode varying the number of threads binding to the controller instance, we have obtained the results shown in Figures 1, 2, and 3. In Fig. 1, the results show that the controllers coded by the C language gave the highest performance (Mul, Libfluid_msg) and below them performed the java coded controllers such as: Beacon, Iris and Maestro.

However, in the latency mode as shown in Fig. 2, Maestro, using the adaptive batching mode, gave the best latency records. In addition, note that the controllers coded in C and java offer a higher performance when varying the number of threads as shown in Fig. 3. However, POX coded in Python does not show any significant difference when using multithreading because Python's support of multithreading is not very efficient.

However, reading these results we have to be cautious as performance in transactions per second and response time must not be the only factors used to choose among the different controllers. For instance, the high modularity present in ONOS and OpenDaylight is a very important factor for application development and in the open source community. Therefore, the comparison must take into consideration all factors as listed in Table 1 and decisions be based on the specific requirements.

## VI. Conclusion

In this paper, we conducted a comparison of several controllers based on multiple criteria. In addition, a performance test using Cbench was done. Thus, due to the diversity of SDN applications and the controller's uses, the choice of the best-fitted controller will be somehow application dependent. We have found that OpenDaylight is a good choice as a full-featured controller. Supporting wide range of applications with a good ecosystem gives it a real chance to become the über-controller. Its integration of the IoT data broker and new IoT specific southbound interfaces in its last release make it the first competitor in the election of the "Internet of the future" controller.

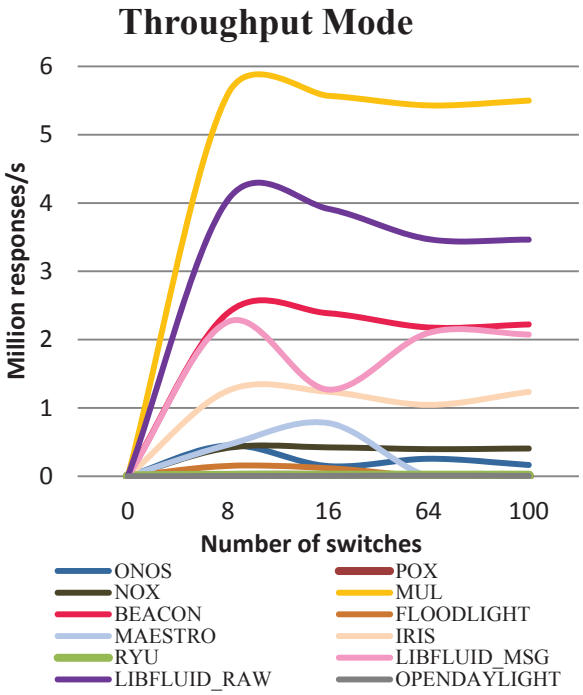


*Figure 1: Cbench test in throughput mode with different number of switches*

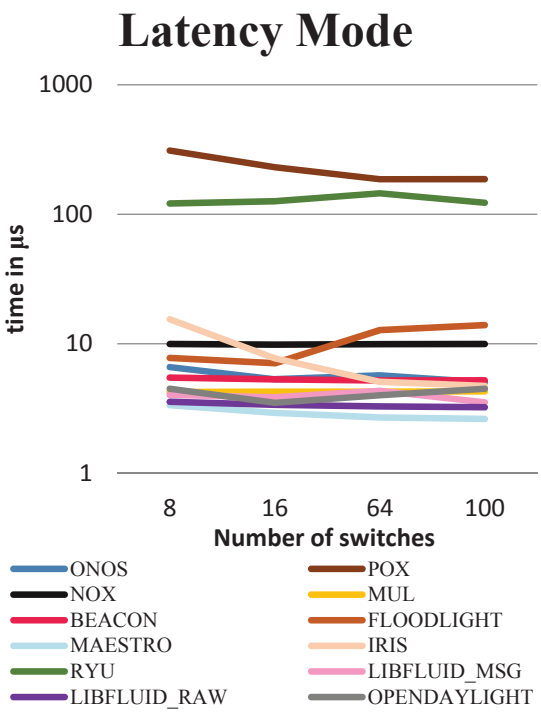


*Figure 2: Cbench test in latency mode with different number of switches*

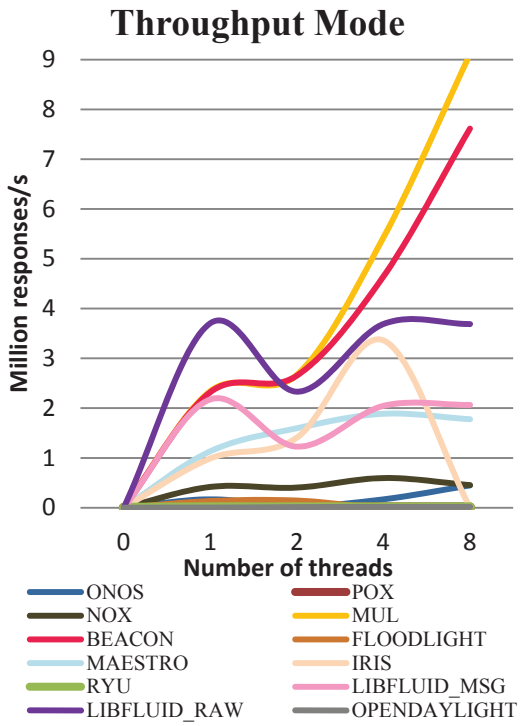


*Figure 3: Cbench test in throughput mode with different number of threads*

## ACKNOWLEDGMENT

This research is funded by TELUS Corp., Canada.

## REFERENCES

[1] M. Monaco, O. Michel and E. Keller, "'Applying Operating System Principles to SDN Controller Design," *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, ACM, 2013, pp. 2.

[2] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado and R. Sherwood, "'On controller performance in software-defined networks," *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, vol. 54. 2012.

[3] R. Khondoker, A. Zaalouk, R. Marx and K. Bayarou, "'Feature-based comparison and selection of Software Defined Networking (SDN) controllers," *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*, pp. 1-7.

[4] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov and R. Smeliansky, "'Advanced study of SDN/OpenFlow controllers," *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, ACM, 2013, pp. 1.

[5] M.P. Fernandez, "'Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive," *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pp. 1009-1016.

[6] S.A. Shah, J. Faiz, M. Farooq, A. Shafi and S.A. Mehdi, "'An architectural evaluation of SDN controllers," *Communications (ICC), 2013 IEEE International Conference on*, pp. 3504-3508.

[7] D. Erickson, "The beacon openflow controller," *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, ACM, 2013, pp. 13-18.

[8] F. Alencar, M. Santos, M. Santana and S. Fernandes, "'How Software Aging affects SDN: A view on the controllers," *Global Information Infrastructure and Networking Symposium (GIIS), 2014*, pp. 1-6.

[9] Feng Wang, Heyu Wang, Baohua Lei and Wenting Ma, "'A Research on High-Performance SDN Controller," Cloud Computing and Big Data (CCBD), 2014 International Conference on, pp. 168-174.

[10] O.N. Foundation, "'Software-defined networking: The new norm for networks," *ONF White Paper* (2012).

[11] Ashton, Metzler. "Ten Things to Look for in an SDN Controller." *white paper [online]*. Available at: www. necam. com/Docs (2013).

[12] Sridhar Rao, "SDN Series Part Eight: Comparison of Open Source SDN Controllers" [online]. Available at: http://thenewstack.io/sdn-series-part-eight-comparison-of-open-source-sdn-controllers/. Accessed on: October, 2015.

[13] "How Do SDN Southbound APIs Work?" [Online]. Available at: https://www.sdxcentral.com/resources/sdn/southbound-interface-api/. Accessed on: October, 2015.

[14] "What are SDN Northbound APIs?" [Online]. Available at: https://www.sdxcentral.com/resources/sdn/north-bound-interfaces-api/. Accessed on: October, 2015.

[15] M. Jammal, T. Singh, A. Shami, R. Asal and Y. Li, "'Software defined networking: State of the art and research challenges," *Computer Networks*, vol. 72, no. 0, 10/29, pp. 74-98, Oct. 2014.

[16] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, D. Niyato and Haiyong Xie, "'A Survey on Software-Defined Networking," *Communications Surveys & Tutorials, IEEE*, vol. 17, no. 1, pp. 27-51, Jun. 2014.

[17] B.A.A. Nunes, M. Mendonca, Xuan-Nam Nguyen, K. Obraczka and T. Turletti, "'A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 3, pp. 1617-1634, Jun. 2014.

[18] D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky and S. Uhlig, "'Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14-76, Jan. 2015.

[19] Fei Hu, Qi Hao and Ke Bao, "'A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 4, pp. 2181-2206, Jan. 2014.

[20] H. Farhady, H. Lee and A. Nakao, "'Software-Defined Networking: A survey," *Computer Networks*, vol. 81, no. 0, 4/22, pp. 79-95, Apr. 2015.

[21] W. Braun and M. Menth, "'Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," *Future Internet*, vol. 6, no. 2, pp. 302-336, May 2014.

[22] M. Jarschel, F. Lehrieder, Z. Magyari and R. Pries, "'A Flexible OpenFlow-Controller Benchmark," *Software Defined Networking (EWSDN), 2012 European Workshop on*, pp. 48-53.

[23] Z.K. Khattak, M. Awais and A. Iqbal, "'Performance Evaluation of OpenDaylight SDN Controller," *Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on*, pp.671-676, 16-19 Dec. 2014.