Journal of Computer and Knowledge Engineering, Vol. 2, No.2. 2019.
DOI: 10.22067/cke.v2i2.84917

Performance Evaluation of Software-Defined Networking Controllers: A Comparative Study

Research Article

Seyed Akbar Mostafavi[1]* Vesal Hakami[2] Fahimeh Paydar[3]

Abstract. Software-Defined Networking (SDN) is a viable approach for management of large and extensive networks with flexible quality of service requirements and huge data traffic. Due to the central role of SDN controllers in traffic engineering and performance of software-defined networks on one hand, and diversity of available SDN controllers on the other hand, an evaluation framework is required to study and compare the architectural choices and performance of distributed and centralized SDN controllers in action. In this paper, we propose a comprehensive framework for performance evaluation of OpenFlow SDN controllers. In this simulation platform, we analyze both centralized and decentralized architectures for controller deployment. Performance of controllers is evaluated based on Quality of Service (QoS) measures including delay and throughput in different network topologies under different workloads. The impact of routing protocols on controller performance in data center networks is also analyzed. Our results can provide valuable insights for scalable design and proper deployment of SDN controllers in the real world scenarios.

Keywords. Software-Defined Networking (SDN); OpenFlow Controllers; Performance Evaluation; Quality of Service

I. Introduction

The improvements of the Internet and mobile technology and the increasing scale of networks has led to a more flexible approach in network management called Software-Defined Networking (SDN) [1][2]. Management of the conventional network models were challenging, especially in large scale networks and were not efficient for satisfying today's requirements of large communication systems such as datacenter networks. As traditional networks expand, they become more complex and managing them becomes more costly. They become less flexible and less controllable over time and their performance might decline as well.

SDN technology and its standards has been recently established towards more flexible, programmable, cost-effective, scalable, and vendor-agnostic networks by decoupling control logic from forwarding devices like switches and routers [3]. This enables central network programming and policy enforcement, which facilitates network management remarkably. SDN is a multi-layered architecture consisting of three main layers such as the application layer, control plane, and data plane. Forwarding devices like switches, routers and firewalls belong to the data plane layer, while the control plane layer is where the controller or the brain of the network resides. The application layer consists of applications that define different policies like traffic engineering and network security. Communication between layers is done by two types of interfaces called the South Bound API (SBAPI) and the North Bound API (NBAPI).

There are various types of SDN controllers that offer different services with different Quality of Service (QoS) requirements and scalabilities. Thus, a comprehensive evaluation framework is needed to choose the best controller in each scenario based on the QoS requirements, type of topology, workload and routing protocols, all of which affect the controller performance tremendously. In this paper, we will provide such a framework and describe our benchmarking metrics and tools in order to facilitate controller choice and subsequently network management. We will analyze both centralized and distributed controllers including POX [4], RYU [5], Floodlight [6], ONOS [7] and OpenDayLight [8]. We choose different metrics for each group of controllers accordingly as they seemed to be the most effective in each type. We analyze different routing protocols and traffic loads in both Google fat-tree and Facebook fat-tree [9] topologies as performance metrics for distributed controllers while delay, throughput and traffic loads in different types of topologies are considered for centralized controllers as metrics in simulations.

In the second section, we introduce the terms and definitions related to SDN. In section three, we review some of the related work done in controller evaluation. We perform two types of evaluation in our study, qualitative comparison which is based on specific features of controllers and mostly theoretic and qualitative study carried out by benchmarking tools and analyzing the results and conclusions toward presenting our evaluation frame work in sections four and six consequently. In section five, we introduce the tools utilized for our evaluation process and final findings, conclusions, and future work are presented in the seventh section.

II. SDN controllers

SDN has made centralized network management possible by decoupling the control logic from network infrastructure devices or data plane. SDN utilizes open source standards instead of proprietary systems and protocols, which increases network flexibility remarkably **Error! Reference source not found.**. SDN has a multi-layered architecture including the application, control plane, and data plane from top to bottom as shown in Fig.1.

Manuscript received November, 2, 2019; accepted February, 21,2020.
[1*] Assistant Professor, Computer Engineering Department, Yazd University, Yazd, Iran, a.mostafavi@yazd.ac.ir
[2] Assistant Professor, School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran, vhakami@iust.ac.ir
[3] Computer Engineering Department, Yazd University, Yazd, Iran, Paydarfahimeh@gmail.com

Control plane: In a software-defined network, the controller is the central management point which has an overall view of the whole network. With SDN controllers there is no need to configure each of the network devices through their terminals, but instead, the softwarized controller programs the whole network [11]. Controller works as a platform for applications to communicate with network devices and manage them by enforcing policies through the controller. SDN controller is able to communicate with different network components using its different interfaces. Communication with low level network components and data forwarding devices is done through the SBAPI, while it communicates with high level applications through the NBAPI.

OpenFlow is a standard SBAPI which is used to control and manage network devices, for instance to send and install flows on the switches. On the other hand, the NBAPI is used by the applications in the top layer to control and manage the network. REST API is the most well-known NBAPI, which is similar to an HTTP server and is used to enforce policies specified by the applications in the top layer to manage the whole network [12].

OpenFlow controllers: OpenFlow was first introduced at Stanford University in 2008 and it made research on real campus and large scale networks like GENI possible [13]. OpenFlow [14][15] encouraged manufacturers towards supporting this protocol in their future switching products. An OpenFlow controller manages flow tables in network devices, while the controller and hardware connect by SSL or TLS encryptions through a secure channel [17][18]. An OpenFlow switch consists of one or more flow tables, a secure channel, and a controller. Also, each flow table consists of multiple flow entries each of which include three parts including a header field, an action, and flow counters. Every incoming flow is reviewed to match against existing flows in a flow table by their header fields. According to the matched flow, the action dictates what happens with the packet, while the counter counts the number of packets received for each flow and the byte counts and flow duration [16]. Fig. 2 shows the structure of a flow entry in an OpenFlow switch. The search initiates as soon as a new flow entry arrives at a switch. If no match is made then the switch sends a message to the controller requesting and action for the unmatched flow. There are several possible actions concerning the flow including sending out the packet to an outgoing port, sending the packet back to the controller, dropping the packet, and sending the packet to the next flow table or specific tables.

There has been six different version of OpenFlow so far. The first was the OpenFlow v.1.0 which was presented at Stanford in 2008. This version has been recently accepted by most OpenFlow vendors but is limited to only one flow table and focuses on layer 2 and ipv4 addressing. OpenFlow v.1.1 added support for MPLS and v.1.2 made matchings more flexible and added ipv6 support. OpenFlow v.1.3 added parallel communication channels between controller and switches. In 2014 v.1.4 was published in which matching processes were mainly improved and optional ports support was also an addition. The last and most recent version, OpenFlow v.1.5 was also released in 2014, in which the outgoing table was introduced and the matches were made by outgoing ports [17].

OpenFlow utilizes three types of messages to communicate with the infrastructure layer including controller-to-switch, asynchronous, and symmetric messages. The first type as apparent by its name is messages from controller to switches, the second from switches to controller, and the third is bidirectional messages between controller and switches.

Controllers are categorized into two groups of centralized and distributed controllers based on their structure and behaviors. Fig. 3 demonstrates this classification.

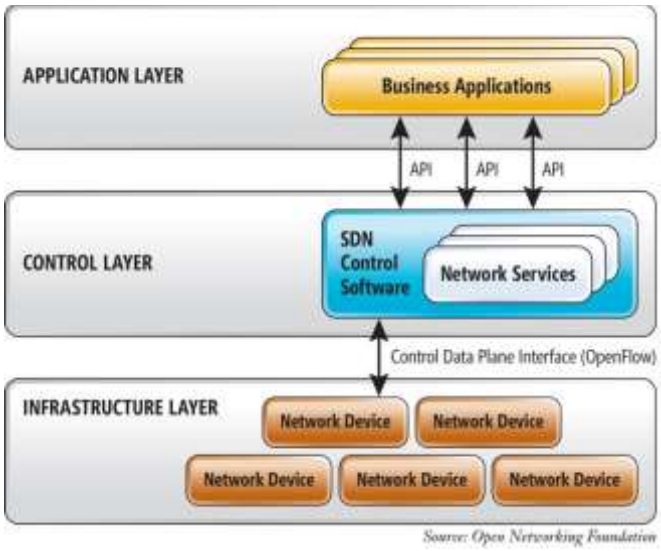

Fig. 1. Three-layered SDN architecture

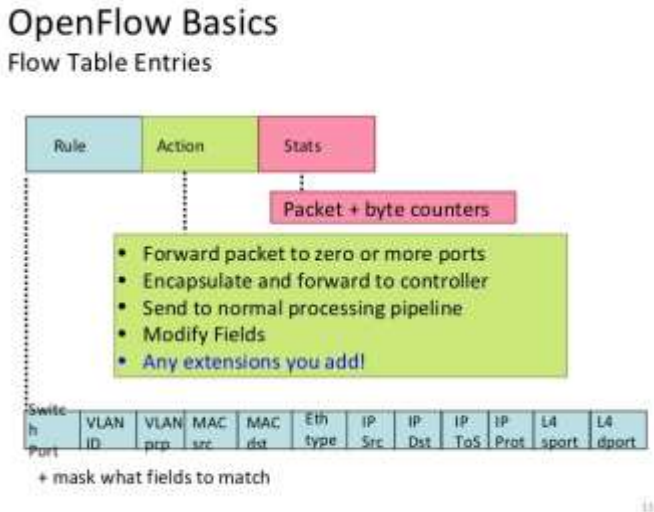

Fig. 2. Flow entry structure

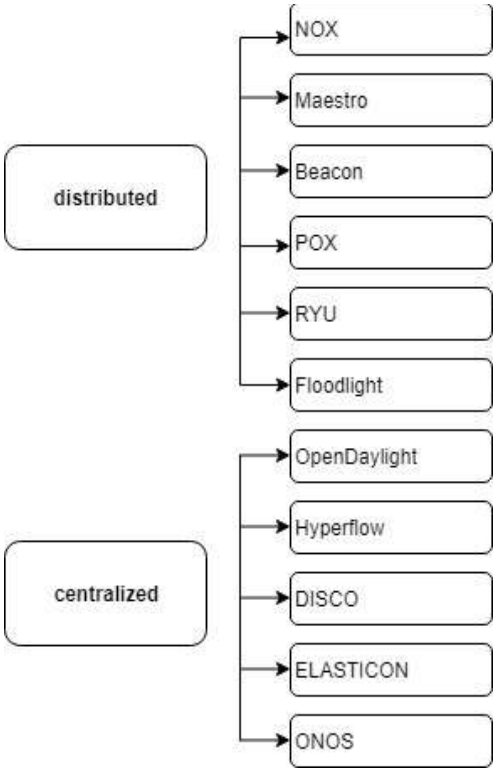

Fig. 3. controllers classification

III. Related work

Due to the separation of data planes from control planes in SDN and with programmability features and centralized controlling nature, flexibility and scalability increases and makes it capable for industrial use. In previous years, several studies have been focused on the OpenFlow controller evaluation most of which analyzed controllers including Bacon, Floodlight, Maestro, NOX, POX, and RYU based on linear performance evaluations and their throughput rates. In several other studies, new or improved versions of controllers were proposed with some previous issues resolved.

Tootoonchian, Gorbunov, Ganjali, Casado, and Sherwood [19] evaluated the performance of four open source controllers including NOX, Beacon, Maestro, NOx, and NOX-MT which was presented by the authors as an optimized version of NOX controller. They discussed that the real performance of these controllers could be considerably better in an optimized network environment than previously assumed. They also designed Cbench that works as a benchmarking tool specifically designed for OpenFlow switch emulations to measure some specific performance metrics. NOX-MT is a multithreading version of NOX which uses optimization methods (e.g., I/O batching) for baseline performance improvements and was proven to outperform NOX by a factor of 33 in the overall performance. Note that this study is outdated because newer versions of these controllers were introduced and also the evaluation of other important controllers such as ONOS, Floodlight, and OpenDaylight had been left out.

Stancu, Halunga, Vulpe, Suciu, Fratu and Popovici [20] analyzed performances of RYU, POX, ONOS, and OpenDaylight controllers based on end-to-end delay and bandwidth parameters. A fixed four level tree topology containing 16 hosts had been utilized as a test environment for the controllers' performance. The study was concluded in RYU having the least end-to-end delay and ONOS with the highest bandwidth among these four controllers. Also, the best suited controllers based on the objectives or outcome expectations were introduced. POX was introduced as the most suitable in an environment with configuration simplicity as the highest priority, although in case of performance POX is not comparable to RYU, OpenDaylight, and ONOS controllers. This study is limited to a static network topology and limited performance parameters in all levels of test, thus leading to a non-general conclusion about controllers in all other types of networks with various parameters in priority.

A performance benchmarking of OpanDaylight and Floodlight was performed in [21] concerning latency and throughput in Cbench. The authors concluded that OpenDaylight is not efficient for use in the industry and Floodlight would be a more mature choice in comparison. They also argued that Cbench lacks traffic models similar to data center network models for testing, and proposed

changes to this tool in order to support datacenter traffic models. Similar to the previously stated works, this study shows some limitations as well like the few number of analyzed controllers (in this case two), irrelevance of the chosen set of controllers to each other in case of function, and the incomplete parameters and network variables in order to find the best suited controllers.

Shiva, Vajihe and Manije [22] discussed the performance of Floodlight and OpenDaylight controllers in different scenarios with different networks topologies under various traffic loads in terms of network latency and packet loss. They state that OpenDaylight acts better in case of latency in networks with tree topologies under half-bandwidth traffics, while Floodlight performs better under heavy loads in terms of packet loss in a tree type network. It is also stated that the comparison between the two could be different in more complex network scenarios (one of the limitations that can be stated about this study is again the few number of controllers studied, limited comparison parameters, and using few network parameters in evaluations).

Darianian, Williamson and Haque [23] evaluated OpenDaylight and ONOS, two novel and adequate distributed OpenFlow controllers. They used Cbench as a benchmarking tool for throughput and latency of the controllers' performances (in both physical and virtual environment). They concluded that ONOS has better throughput and less delay than OpenDaylight. One of the limitations of this study is the fact that these two controllers are not evaluated in test scenarios similar to data centers and cloud networks, which are where they are mostly used.

Fancy and Pushpaltha [23] analyzed POX -a python-based controller- and Floodlight -a java-based controller- as representatives of all controllers developed by these two programming languages. They executed a performance comparison between them on throughput and delay. The tests were done in Mininet with various network topologies. This study was also limited to two controllers that do not necessarily cover all other controllers developed by python or java; also limited network variables were measured.

IV.Qualititative comparison of ten OpenFlow controllers

Table I and Table II present separate qualitative comparisons for both centralized and distributed groups of controllers, each by different evaluation features and from various perspectives. This part is mostly based on theories and features each controller possesses which might be used as a way of categorizing controllers and predicting their properties, but does not necessarily categorize behaviors in each group as numerous factors take part in the controller performance.

V. Configurations and benchmarking tools

A. Configurations and evaluation scenarios

ONOS: To launch and execute the ONOS controller, we need at least a 2-core processor with 2 GBs of RAM. One of the advantages of this controller is its active support team and complete documentation which facilitates troubleshooting. ONOS is java-based and requires JRE 1.8 or higher in order to be executed on our system. We used the latest version of ONOS at the time in our tests and evaluations.

OpenDayLight: ODL is executed in a Java Virtual Machine (JVM) and can run on any system that supports java. However, on a system with a Linux distribution and a 1.7 JVM, ODL performs its best. We used the latest version of ODL at the time in our tests.

Floodlight: Floodlight is a java-based controller and one of the most popular and applied controllers which is mostly used in small networks and academic environments.

RYU: RYU is a python based controller, also popular in academia and small networks. This controller supports 1.1, 1.2, 1.3, 1.4, and 1.5 versions of the OpenFlow protocol as well as Netconf [26] and OF-config protcols.

POX: POX is a python-based controller and is known as one of the most popular controllers for academic purposes.

B. Test environment and benchmarking tools

We use Mininet [24] and Cbench [26] to establish a test environment. Also using python, we created datacenter topologies to establish tests based on them as well. The datacenters are crucial to our study because of the role they play in today's world and most possibly in the future [28].

Table. 1. Qualitative comparison of general-purpose controllers

controller	platform	Port number	GUI	Multithreading	Programming language	Open source
NOX [29]	Linux	6633	Yes	Yes	C++/python	Yes
POX	Linux, Mac, Windows	6633	Yes	-	Python	Yes
Maestro [30]	Linux	6653	No	Yes	Java	Yes
Beacon [31][32]	Linux	6653	Yes	Yes	Java	Yes
Floodlight	Linux	6653	Yes	-	Java	Yes
RYU	Linux	6633	-	-	Python	Yes

Table. 2. qualitative comparison of distributed controllers

Controller	Network type	Data structure instances	Mapping		Controller-switch arrangement		Related projects
			Dynamic	Static	Master/Slave	IPalias	
Elasticon [33][33]	Data centers	Hazelcast	✓		✓		-
DISCO	Data centers-WAN	Extended DB		✓	-	-	Floodlight
HYPERFLOW	WAN	Wheelfs		✓	-	-	NOX
ONOS	Data centers	Raftlog		✓	✓		

Establishing the test environment: A test environment is built on a system with a 7-core processor and 16 GB RAM. Two virtual machines with 4GB RAM and a 3-core processor are built on this server that runs Ubuntu 16.0.4 LTS. One of the virtual machines is used for the controller installations and the other for establishing the test network with Cbench and Mininet installed on it. Cbench, Mininet, and wireshark are the tools chosen based on functionality and the metrics that can be measured by them, for our study towards benchmarking the controllers.

Cbench: Cbench is a popular tool, especially designed for OpenFlow controllers' evaluations. It produces packet-ins in order to simulate asynchronous flows. Also, it emulates some switches that are connected to the controller. These switches send packet-ins to the controller. Reception of a packet-in by the controller results in a new flow entry for the switch. In reply, the controller runs its algorithm and adds new rows in the flow table for the requesting switch and sends the flow table back to the switch. The switch will wait until it receives a flow-mod message from the controller. Cbench uses fake switches instead of physical switches to send packets to the controller in order to be able to specify the exact number of packets sent and other parameters from the process initiation to obtain precise output statistics and analyze them. Fake switches determine the following parameters including OFP_Hello exchange messages that ban response and request messages, FS parameter which is a fake switch pointer, SOCK parameter which identifies a socket connection, BUFSIZ parameter which shows the size of the in/out buffer, MODE parameter which distinguishes the latency or throughput mode from each other, and total_mac_adresses that is the number of all the hosts connected to the switch.

Mininet: Mininet is a network emulator that we use as a bench marking tool in our study. It allows the creation of networks with various topologies consisting of a number of virtual hosts, links and switches. With the use of this tool you can easily access any of the network components by the Mininet CLI and design the network specifically according to your needs, share it with others, and eventually develop it using real hardware. Mininet enables us to test a network and improve it in an emulated environment before the actual implementation of our network. The default topology in Mininet consists of a switch and hosts that are connected to the switch and each other while the switch itself is connected to an OpenFlow controller. In Mininet hosts are capable of running on separate linux CLIs [15] e.g. the iperf command which returns the bandwidth between user and the server is easily obtainable in this manner. A desired topology in Mininet can be created by python script writing while there are only a few topologies available in this tool such as tree, linear, and single topologies as default.

Wireshark: Wireshark [36] is a powerful open source tool for capturing and analyzing various traffics and network protocols. It is easy to use and is popular among network and security specialists.

VI. Quantitative comparison of five OpenFlow controllers

In section five, we introduced some of the basic concepts of SDN networks and summarized some of the previous work done in performance evaluation of OpenFlow controllers. Since the related studies have analyzed limited parameters in their evaluations and less focus had been put on decentralized controllers, in this paper, we propose a new, more thorough and comprehensive framework for performance evaluation of OpenFlow controllers with more emphasis on the distributed type. Out of the ten controllers presented in our qualitative review, five of them were chosen that represent unique controller features best suited for our study as shown in Table III. Different approaches, parameters, and criterion has been presented based on the category which controller belongs to.

We present a quantitative comparison of OpenFlow controllers that can be utilized as a comprehensive evaluation framework by network managers. In our framework we separately evaluate centralized and distributed controllers. We first measure performance metrics that we have specifically chosen for each category of controllers. Delay, throughput and the overall QoS are the performance parameters chosen for centralized controllers while for distributed controllers it is the network delay. Then we identify and analyze other effective factors on controllers' performance including topology type and scale of a network for both centralized and distributed controllers. However, for distributed controllers, traffic load and routing protocols are also analyzed. At last, we will be able to determine any controller's performance behavior with specifically-defined policies and features based on our results.

Table. 3. controllers used in our qualitative evaluation

controller	Support partners	NBAPI	SBAPI	Supported platforms	Centralized /distributed	Programming language
ONOS	ON.LAB, AT&T, Ciena, Cisco, Ericsson, Fujitsu, Huawei, Intel, Nec, Nsf.Ntt Communication, SK Telecom	REST API	OF 1.0, 1.3, NETCNF	Linux, MAC OS, and Windows	Distributed	Java
OpenDayLight	Linux Foundation with memberships covering over 40 companies, such as Cisco, IBM, NEC	REST API	OF 1.0, 1.3, 1.4, NETCONF/Yang, OVSDB, PCEP, BGP/LS, LISP, SNMP	Linux, MAC OS, and Windows	Distributed	Java
POX	Nicira	REST API	OF 1.0	Linux, MAC OS, and Windows	Centralized	Python
RYU	Nippo Telegraph And Telephone Corporation	REST For Southbound	OF 1.0, 1.2, 1.3, 1.4, NETCONF, OFCONFIG	Most supported on Linux	Centralized	Pyrhon
Floodlight	Big Switch Networks	REST API	OF 1.0, 1.3	Linux, MAC OS, and Windows	Centralized	Java

Centralized controllers: These controllers were the first introduced SDN structures in which a single controller manages all the devices in a network. Each flow is received at the controller as a *packet-in* and after the controller processes the packet based on the defined policies by the application layer and defines an action for that flow, it sends the requesting switch *packet-out* and *flow-add* packets. There are various controllers developed by java and python programming languages namely, RYU [29], Beacon [9], floodlight [16], POX [28] and NOX [23]. The main concept of these networks refer to the physical centrality which means each controller is aware of the overall state of the network and is connected to all the switches in topology. These types of controllers mostly have academic and research use in small scale networks. Among POX, NOX, and RYU, three python-based controllers, we only evaluate POX and RYU as POX is the improved version of NOX. Between Floodlight and Beacon that are java-based and require large memory allocations, we chose Floodlight as according to [2] it requires less memory than Beacon.

Decentralized controllers: One of the main challenges of networks with centralized controllers is scalability and their inefficiency for data centers and cloud networks that are nowadays crucial. Distributed contollers such as HYPERFLOW, Elasticon, ONOS, DISCO, and Open DayLight have some unique features like the physical distribution of the particiapting controllers in a network topology and the fact that there is no need for all switches in the network to be connected to the controller. These types of controllers are used in large scale networks like WANs and data centers. For our study, we only evaluate ONOS and OpenDayLight, two inherently distributed controllers as they are the most commonly used controllers in industry utilized in data centers and cloud infrastructures. Another advantage of these two controllers is being supported by well-known brands such as Cisco, Intel, Ericsson, Fujitsu, and Huawei. ONOS and OpenDayLight are both java-based and are executable on platforms containing OSGi.

1) Determining the appropriate evaluation criterion

QoS performance in centralized controllers: End-to-end delay and network bandwidth are two very effective factors on QoS and performance of centralized controllers. Network delay refers to the transmission time from source to destination, and bandwidth between two hosts refers to the available bandwidth for data transmission. The lowest end-to-end delay and highest bandwidth results in the highest QoS. For centralized controllers in our study namely POX, RYU, and Floodlight, we analyze QoS in different scales and topologies and introduce the controller with the highest QoS in each of the scenarios. In each stage we perform tests 10 times and take the average of these 10 outcomes as the final result. Also, we gradually increase the size of the network from 1 to 63 OpenFlow switches. According to Fig. 4, RYU performs better than POX and Floodlight in terms of latency as the scale increases, while POX has the highest delay in networks with more than seven switches. According to Fig. 5, all three controllers have similar decreasing patterns of bandwidth but again RYU closely out performs the other two and thus is the best choice in an increasing-scale network.

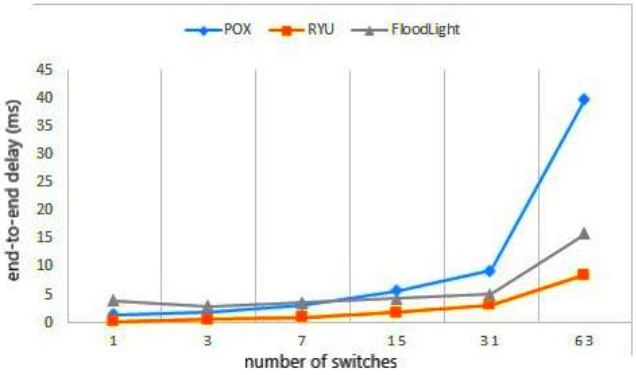

Fig. 4. End-to-end delay vs. network scale

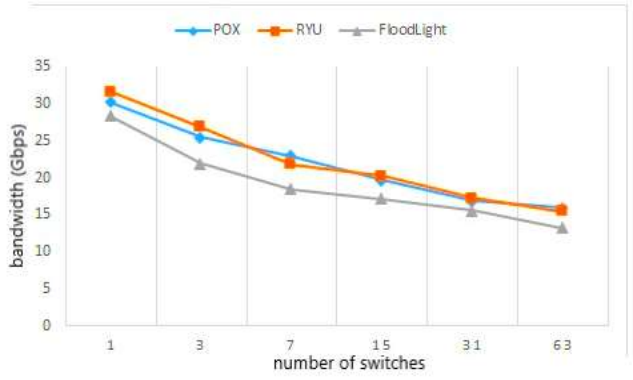

Fig. 5. Bandwidth vs. network scale

Delay and throughput performance in centralized controllers: When a new flow arrives at a switch, firstly the switch looks up its flow table for matches and if any matches are made, the corresponding action is executed on the flow otherwise the switch sends the flow towards the controller in the form of a packet-in and requests an action for the flow. Then the controller responds with a defined action in form of a packet-out. The amount of time needed for this process or in other words the amount of time required by the controller to process packet-ins and send out packet-outs is referred to as the controller delay. On the other hand, throughput is the amount of configuration requests a controller can handle over a time unit. Throughput in a control layer is the main parameter towards determining the number of required controllers in a specific network scenario in order to sufficiently handle traffic loads. Fig. 6 illustrates POX, RYU, and Floodlight delay results against different number of switches. RYU has somewhat static delay in every scale while delay increases as the network scale does in POX and Floodlight has relatively the highest delay in every scale than all of them. Thus, in a delay-sensitive network RYU is the best choice. Also from the perspective of the development language of a controller python-based controllers showed less delay than the ones that are java-based. In terms of throughput, Fig. 7 shows that RYU has the least but static throughput throughout all scales while Floodlight has the highest throughput than others and is the best suited.

Fig. 6. dealy against network scale

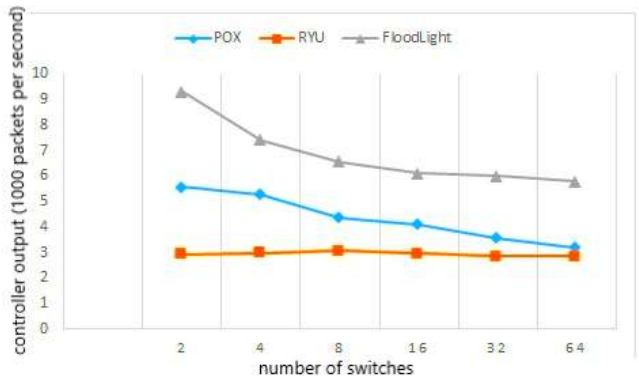

Fig. 7. Throughput against network scale

RTT in distributed controllers: Comparison between OpenFlow controllers is done through the evaluations on communication efficiency between hosts. Round-Trip Time (RTT) between hosts is calculated by a ping command using ICMP request-response. In a test environment, ping is run on two hosts farthest away from each other. We specifically introduce controller with the lowest RTT in each scenario.

2) Effective network variables on performance

One of the effective factors on the controllers' performance is the structural variables of a network which directly impact performance such as network size and topology type.

Network scale in centralized controllers: Increases in network scale mean more switches and more hosts which leads to more requests towards the controller directly affecting controllers' performance.

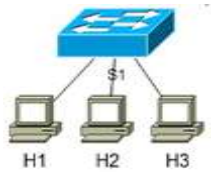

Fig. 8. Single topology structure

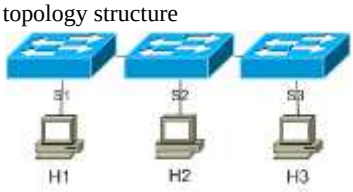

Fig. 9. Linear topology structure

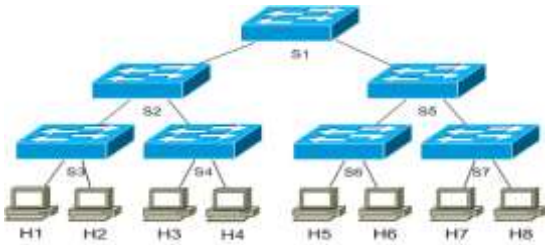

Fig. 10. Tree topology structure

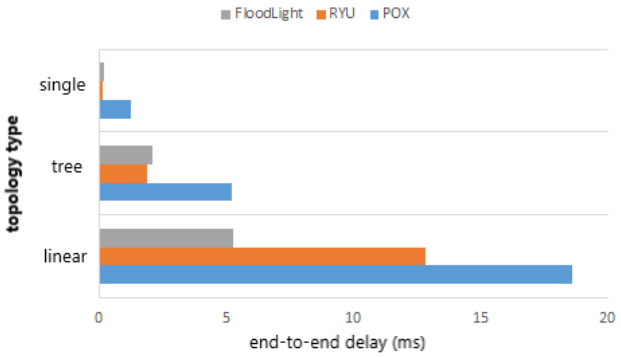

Fig. 11. En-to-end delay against topology type

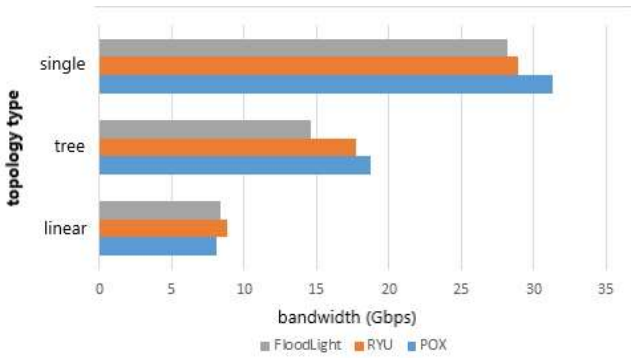

Fig. 12. Bandwidth against topology type

Topology in centralized controllers: An effective factor on QoS of OpenFlow controllers is the network topology type including single, linear, and tree topologies as shown in Fig. 8, Fig. 9, and 0 that are mostly seen in small scale networks and academic environments. We evaluate QoS of POX, RYU, and Floodlight controllers by delay and throughput amounts as stated before.

- Single topology: Networks with this type of topology contain only one OpenFlow switch with several hosts connected to it and the switch is connected to the controller. Results as illustrated in Fig. 11 and Fig. 12 show that if we arrange the three controllers in this manner all of them obtain similar QoS considering their delay and throughput.
- Linear topology: In this type of topology, there are several switches that have linear connections. Each host is connected to a switch linearly connected to other switches all of which are connected to the controller. According to Fig. 11 and Fig. 12 Floodlight has the least delay than others after RYU, while all three of them show similar bandwidths. Thus, in a network with a linear structure Floodlight provides the best QoS.
- Tree topology: In a tree topology all OpenFlow switches and hosts are hierarchically connected. Fig. 11 and Fig. 12 illustrate that RYU and Floodlight have roughly the same delay whereas RYU has higher bandwidth than Floodlight and POX. Thus,, RYU is the proper choice in a network with tree topology.

Transmission packet size in distributed controllers: Maximum Transmission Unit (MTU) refers to the size of the largest packet allowed to pass through network links. By standard the largest transmission packet is 1500 bytes and larger packets are discarded. Data center networks require low delay and large MTUs. In this section, we gradually increase MTU from 1400 to 1500 and observe the impact of packet size on end-to-end delay. Fat-tree is the network topology with 16 hosts. Fig. 13 shows that OpenDayLight's performance is negligible to packet size while ONOS shows increasing delay with increases in MTU. Thus OpenDayLight is the suitable choice in this scenario.

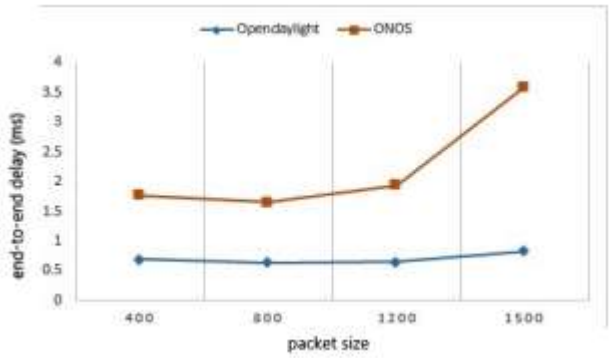

Fig. 13. End-to-end delay against MTU in facebook fat-tree topology

Network scale in distributed controllers: By increasing the number of switches and hosts, the number of requests toward controller increases as well which directly impacts performance.

Datacenter topology in distributed controllers***:*** Data centers are rapidly increasing and act as an essential part of cloud computation and online web services. Data centers consist of thousands of nodes that demand high bandwidths. For instance corporates like amazon, google and microsoft profit from high scale data centers carying out their cloud computations. Even corporates such as Dropbox and Apple are leaning toward private cloud centers. Statistics show data centers' increasing traffic loads which is a confirmation of every day increasing bandwidth requirements in these types of networks. Data centers play an important role in cloud-based applications' performance efficiency. Many topologies have been developed over the years to overcome these challenges and fullfill data center demands. One of these solutions is the fat tree topology which have showed better outcomes than others. Google fat-tree and Facebook fat-tree [13] are among the most used and most adequate topologies in case of performance in datacenter networks. Thus, we analyzed these topologies in our tests and evaluation environments.

- Google fat tree: This topology is currently the most popular in data centers and consists of different levels as shown in 0 .This topology has three switch layers including edge, aggregation, and core switches. Edge switches are directly connected to hosts in a way that ***n***

ports of the switch are connected to n/2 hosts and the rest are connected to higher level switches called the aggregation switches. In the same manner the aggregation switches are connected to core switches. In this topology there are several paths between every two hosts which eventually increases fault tolerability. We evaluate performance ONOS and OpenDayLight in a google fat-tree topology based on delay and with 8, 16, 32, and 64 switches. Fig. 15 shows their performance based on delay against the number of switches. OpenDayLight has less delay and shows little increase with more switches while ONOS shows ascending increase over more number of switches. Thus OpenDayLight is the best suited for a google fat-tree topology.

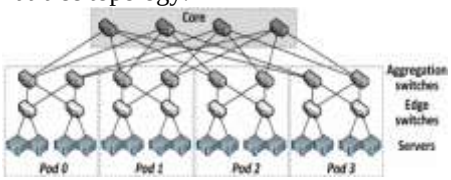

Fig. 14. Google fat-tree topology

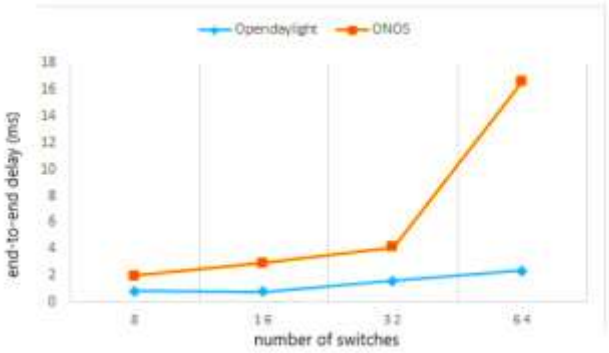

Fig. 15. End-to-end delay against network scale in google fat-tree topology

- Facebook fat-tree: This topology is illustrated in Fig. 16. In this setting, the whole illustrated structure is called a pod. Each pod in this topology consists of a number of fabric switches and Top-Of-Rack (TOR) switches that can be used up to 48 in maximum. For test purposes we can initiate the network by 4 fabric switches and 16 TORs and enlarge the network gradually. We evaluate performance ONOS and OpenDayLight in a Facebook fat-tree topology based on delay and with 16, 24, and 32 hosts. Based on Fig. 17 ONOS outperforms OpenDayLight in delay and is better suited in this scenario. According to Facebook fat-tree structure, it can be said that in a network with the least number of switch layers and the least number of steps from a host to controller, ONOS is best suited for managing the network.

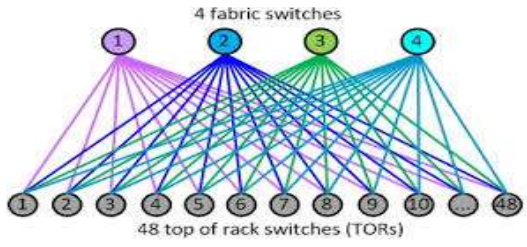

Fig. 16. Facebook fat-tree topology

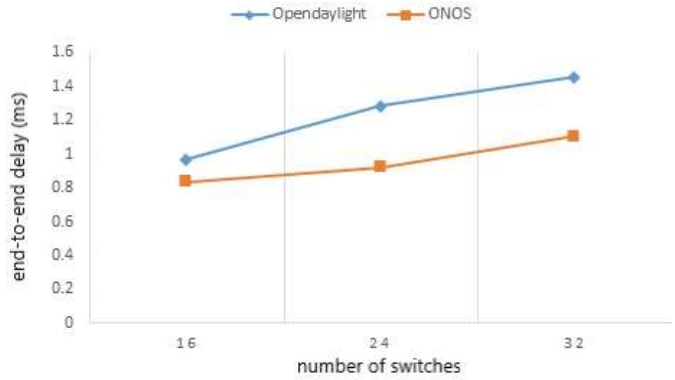

Fig. 17. End-to-end delay against network scale in facebook fat-tree topology

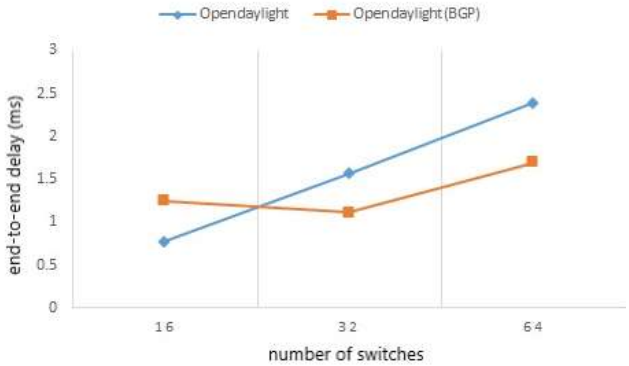

Fig. 18. OpenDaylight and ONOS performance without a routing protocol against different network scales

Routing protocols of distributed controllers: Protocols are also one of the effective factors on performance of OpenFlow controllers. In SDN, controller possesses an overall vision of the network and is aware of all the existing paths in the network. Controller chooses the best path based on its knowledge. With the use of OpenFlow routing protocols controllers transmit paths' information to network devices and this protocol is basically utilized for communication between the controller and network hardware. By default, controllers do not use a specific algorithm or routing protocol. They simply forward packets by their overall vision of the network and shortest-path policies. By the use of appropriate protocols, we can positively affect controller performance. Firstly, we evaluate the performance of ONOS and OpenDayLight without a routing protocol based on end-to-end delay against network scale of 16, 32, and 64 hosts. Then we install BGP protocol on both of the controllers and repeat the test. In this test, we use fat-tree to implement the network topology. As illustrated in Fig. 18 and Fig. 19 controllers utilizing a routing protocol perform better and ONOS outperforms OpenDayLight and the effect of BGP was greater on ONOS. In Fig. 20 the effect of BGP on ONOS performance improvement is illustrated [34]. Based on the policies defined for BGP the virtual topology is divided into areas in each of which routing takes place separately from other areas. Due to this policy the number of requests sent towards the controller decreases and leads to better overall performance. As shown in Fig. 18 and Fig. 19 the effect of BGP is only visible in large networks while under small size networks, the effect in negligible and it is better to use the default routing without any specific routing protocols.

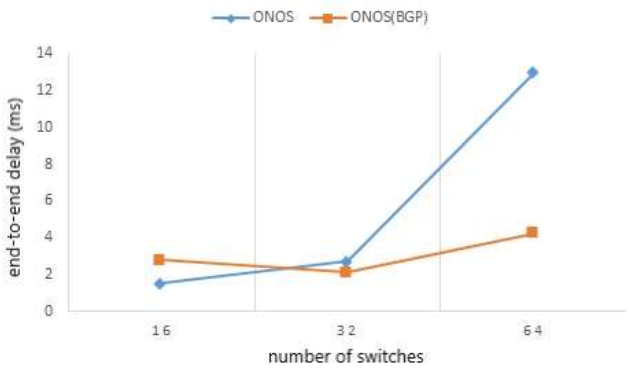

Fig. 19. OpenDaylight and ONOS performance with bgp against different network scales

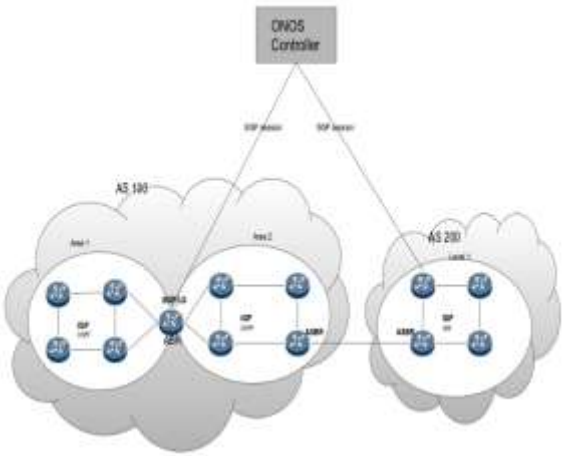

Fig. 20. Routing in ONOS with BGP

VII. Conclusions

In this paper, we introduced SDN and its different layers including the controller layer and its fundamental role on the overall network performance. We performed a comprehensive study of the OpenFlow controllers and proposed an evaluation platform of centralized and distributed controllers based on multiple criteria.

We first categorized the controllers into two groups of centralized and distributed. Centralized controllers are used in small networks while distributed controllers can manage large scale networks like datacenters or cloud networks. Then we presented the methods and scenarios used in our study with specific parameters to aid us choose the best suited controller for each situation. In our evaluations for the centralized controllers parameters consisting of QoS, latency, and throughput were measured in various topologies and different scales, while end-to-end delay was measured for distributed controllers in Google fat-tree and Facebook fat-tree, two of the most well-known datacenter topologies with different scales under various workloads.

At last the following evaluation results were achieved:

- Among our chosen centralized controllers, RYU performed best in network management for large scale networks in term of QoS while in small networks Floodlight outperformed others in QoS.
- Among our chosen set of centralized controllers, RYU outperformed others in a network with a tree topology and Floodlight performed best in a network with a linear topology than others although for a single topology the performance differences between controllers were negligible.
- Among our chosen centralized controllers, RYU performed best in terms of least delay in a large scale network while Floodlight showed the highest throughput than others.
- Among our chosen distributed controllers, OpenDayLight outperformed ONOS in a Google fat-tree topology while in a Facebook fat-tree topology ONOS performs better.
- Among our chosen distributed controllers, OpenDayLight was the more stable in a network with increases in workload overtime than ONOS which showed increased delay.

In distributed controllers, changing the default routing protocols to BGP caused performance improvements in both OpenDayLight and ONOS while this effect was greater on ONOS.

Data Availability

The data used to support the findings of this study are available upon request to the corresponding author.

Conflict of Interest

The authors declare that there are no conflict of interest regarding the publication of this paper.

Refrences

[1] Lu, Jie and Zhang, Zhen and Hu, Tao and Yi, Peng and Lan, Julong, " A survey of controller placement problem in software-defined networking," *IEEE Access*, vol. 7, pp. 24290-24370, (2019).

[2] Zhang, Yuan and Cui, Lin and Wang, Wei and Zhang, Yuxiang, "A survey on software defined networking with multiple controllers," *Journal of Network and Computer Applications*, vol. 103, pp. 101-118, (2018).

[3] D. Kreutz, F. Ramos, P. Verissimo, C. Rothenberg, S. Azodolmolky, and S. Uhlig,"Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, pp.14–76, (2015).

[4] "POX," http://noxrepo.org, visited: 2017-8-17.

[5] NTT, "Ntt laboratories osrg group," http://osrg.github.com/ ryu, Visted: 2019-8-28.

[6] "Floodlight," http://www.projectfloodlight.org/floodlight/ Visted: 2019-8-23.

[7] P. Berde, M. Gerola, J. Hart, Y. Higuchi, "ONOS: Towards an open, distributed SDN OS," In *Proceedings of the third workshop on Hot topics in software defined networking* , pp.1-6, (2014).

[8] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," In *Proceedings of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pp. 1–6, (2014).

[9] M. Jarschel, C. Metter, T. Zinner, S. Gebert, and P. Tran-Gia, "Ofcprobe: A platform-independent tool for openflow controller analysis," in *IEEE International Conference on Communications and Electronics*, pp. 182–187, (2014).

[10] M. Sanaei and S. Mostafavi, "Multimedia delivery techniques over software-defined networks: A survey," *5th International Conference on Web Research (ICWR)*, pp. 105-110, (2019).

[11] J. Liao, H. Sun, J. Wang and K. Li, "Density cluster based approach for controller placement problem in large-scale software defined networkings," *Computer Networks*, vol.112, pp.24-35, (2017).

[12] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-defined networking: a comprehensive survey," In *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14-76, (2015).

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," In *Proceeding of the ACM SIGCOMM Computer Communication Review*, pp.69–74, (2008.

[14] "Open Networking Foundation specifications," https://www.opennetworking.org/sdn-resources/onf-specifications, Visited: 2019-1-20.

[15] OpenFlow Switch Specification Version 1.0.0. Open Networking Foundation (ONF), December (2009).

[16] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti," A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," In *Proceeding of IEEE Communications Surveys and Tutorials*, pp. 1617–1634, (2014).

[17] "OpenFlow," http://www.openflow.org/ ,visited:(2018-2-15.

[18] "Open Networking Foundation," http://www.opennetworking.org , Visited:(2018).

[19] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood," On Controller Performance in Software-Defined Networks," In *Proceedings of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, (2012).

[20] A. L. Stancu, S. Halunga, A. Vulpe, G. Suciu, O. Fratu and E. C. Popovici, "A comparison between several Software Defined Networking controllers," In *Proceedings of the 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS)*, pp. 223-226, (2015).

[21] Z. Khattak, M. Awais, and A. Iqbal, "Performance Evaluation of OpenDaylight SDN Controller, " In *Proceedings of the 20th IEEE International Conference on Parallel an Distributed Systems*, pp. 671-676, (2014).

[22] R. Shiva, A. Vajihe and K. Manijeh, "Performance evaluation of sdn controllers: Floodlight and OpenDaylight," *IIUM Engineering Journal*, vol. 17, pp. 47-57, (2016).

[23] M. Darianian, C. Williamson, I. Haque, "Exprimental evaluation of two openflow controllers," In *Proceeding of the 25th international conference on Network Protocols*, pp. 1-6, (2017).

[24] C. Fancy and M. Pushpaltha, "Performance Analysis of SDN/Openflow controllers: POX Versus Floodlight," *Wireless Personal Communications*, vol. 98, no. 1, pp. 1679-1699, (2018).

[25] "MiniNet,"http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet,visited: 2017-9-10.

[26] "Cbench,"http://www.openflow.org/wk/index.php/Oflops, visited: 2018-3-11.

[27] . R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)". RFC 6241, Internet Engineering Task Force (IETF), June 2011.

[28] S. Mostafavi and V. Hakami, "A new rank-order clustering algorithm for prolonging the lifetime of wireless sensor networks", *International Journal of Communication Systems*, vol. 33, no. 7, (2020.

[29] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown and S. Shenker, "NOX: towards an operating system for networks," In *Proceeding of the ACM SIGCOMM Computer Communication Review*, pp. 105-110, (2008).

[30] Z. Cai, A. L. Cox and T. S. Eugene Ng. "Maestro: A system for scalable openflow control," In *Proceeding of the Technical Report*, pp. 10-11, Rice University, Dec (2010).

[31] "BeaconOpenFlowController," https://openflow.stanford.edu/display/Beacon, Visited:2017-10-24.

[32] D. Erickson,"The Beacon OpenFlow Controller," In *Proceedings of The Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, p. 13-18, (2013).

[33] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman and R. R. Kompella, "ElastiCon: an elastic distributed SDN controller," In *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pp. 17-27, (2014).

[34] A. Dixit, F. Hao, S. Mukherjee, T. V. Laseshman and R. Kompella, "Towards an elastic distributed SDN Controller," In *Proceedings of the second ACM SIGCOMM workshop on Hot Topics in Software Defined Networking*, pp. 7-12, (2013).

[35] "ONOS and BGP Protocol," https://wiki.onosproject.org /display/ONOS/BGP+ protocol +with+Link-State +Distribution, visited: 2019-1-18.

[36] "wireshark," http://wireshark.org, Visited: 2017-11-4.