HETEROGENEOUS IMPLEMENTATION OF OPENFLOW DATA-CENTRE NETWORK TESTBED

Gojko Gogic[*], *Slavica Tomovic*[**], *Igor Radusinovic*[***]

Keywords: *Data center, SDN, OpenFlow, testbed.*

Abstract: In this paper, we present results of performance evaluation of OpenFlow data-center (DC) network testbed, developed as part of BIO-ICT research and innovation platform. The testbed provides rich testing environment for experimental research in the area of software-defined networking (SDN), which is widely recognized as a fundamental technology for next generation DCs. The cost-efficient and solid performance testbed design is achieved with the heterogeneous data plane, consisted of software and hardware OpenFlow switches. In order to identify the most suitable solution for the control plane, we compared performance of four open-source SDN controllers (ONOS, Floodlight, POX and Ryu) via Cbench benchmarking tool. In addition, we have conducted several testbed experiments which provide useful insight into the data plane performance when different controllers are used. The obtained results indicate that Floodlight controller outperforms ONOS, POX and Ryu controllers in terms of throughput, processing latency and scalability.

1. INTRODUCTION

Software Defined Networking (SDN) is a new networking paradigm that changes traditional network architecture by separating the control and management logic from the data forwarding devices [1-2]. Instead, control and management planes are localized on external entity – SDN controller, which maintains centralized view of the network state and is able to dynamically reprogram the network behavior. SDN is commonly associated with OpenFlow protocol [3], a southbound API which defines rules in communication between OpenFlow switches and SDN controller. Using OpenFlow protocol, the controller can pro-actively or reactively instruct OpenFlow switches how to identify and serve different traffic

[*] G. Gogić, Faculty of Electrical Engineering, University of Montenegro, Montenegro (e-mail: gojkog@ac.me).

[**] S. Tomović, Faculty of Electrical Engineering, University of Montenegro, Montenegro (e-mail: slavicat@ac.me).

[***] I. Radusinović, Faculty of Electrical Engineering, University of Montenegro, Montenegro (e-mail: igorr@ac.me).

flows in the network. These instructions are stored in data structures called Flow Tables [3]. Each entry in the Flow Table consists of: i) matching fields - that serve to differentiate traffic flows based on the packet header content; ii) actions - which define how a traffic flow should be handled (e.g. forward to, drop, etc.) and iii) counters – which serve for statistical purposes. When a network device receives a packet that does not match any of the Flow Table rules, it requests a new Flow Table entry from the controller. Once the controller installs an appropriate Flow Table rule for a new traffic flow, all packets of that flow will be processed in the data plane, until validity time for the rule expires.

Although OpenFlow was originally proposed for campus and wide-area networks, there are many quantified arguments that OpenFlow applications are able to increase quality of service (QoS) and resource utilization in data center (DC) networks [4-6]. By making control plane programmable, SDN/OpenFlow fosters development of innovative network management applications. However, in order to reach production-ready state, SDN applications need to pass thorough evaluation in experimental environment. Important challenge here is to create a large-scale testbed in the most efficient manner. The data plane of the testbed might involve hardware and software solution of OpenFlow switches. While open-source software solutions are cost-efficient and convenient for educational purposes, they cannot reach performance of the specialized hardware switching devices. Thus, the testbed should be designed by taking into account tradeoff between the implementation expenditures and the expected performance.

This paper presents implementation of DC network testbed, developed as a part of the BIO-ICT research and innovation platform [7]. The main contributions of the paper are as follows:

1) **Virtualization of the network data plane**. We show how SDN hardware-based OpenFlow switches can be partitioned into a number of virtual OpenFlow switches, each considered as independent physical device from the controller's perspective. In this way, it is possible to create a full network of high-performance DC switches from a single physical device with sufficient number of network interfaces.
2) **Heterogeneous testbed design**. In order to further increase cost-efficiency of our testbed, we use software-based OpenFlow switches as well, and open-source SDN controllers.
3) **Comparison of different OpenFlow controllers.** Considering that controller scalability is one of the main challenges in SDN networks, we perform performance evaluation of four popular open-source controllers: ONOS [8], Floodlight [9], POX [10] and Ryu [11]. As a benchmarking tool we used Cbench software [12], which can emulate OpenFlow networks with thousands of switches and hosts.
4) **Overall testbed performance testing**. We investigated how different parameters impact network performance by conducting a range of experiments on the testbed: from ping, over TCP transfer of different data sizes, to UDP transfer with different data rates [13]. We found that controller-switch delay degrades throughput more significantly when larger data sizes are transferred. Further on, we showed that Java based controllers (Floodlight and ONOS) outperform Python-based POX and Ryu. The obtained results also indicate the level of that software-based solutions of OpenFlow switches are able to achieve very high throughput if deployed on physical machine with sufficient resources available.

The rest of this paper is organized as follows. In Section 2 we present our SDN testbed and brief background on the analysed OpenFlow controllers. Section 3 explains the experimental methodology we used. Experimental results with the corresponding discussion are given in Section 4. The paper concludes in Section 5.

2. SDN TESTBED

Fig. 1 shows the design of OpenFlow DC network testbed that will be discussed in the rest of the paper. The logical testbed topology includes 20 network nodes, but physical data plane is made of only two hardware-based OpenFlow switches: Pica8 3295 [14] and HP Procurve 6600 [15], and four software-based OpenFlow switches.

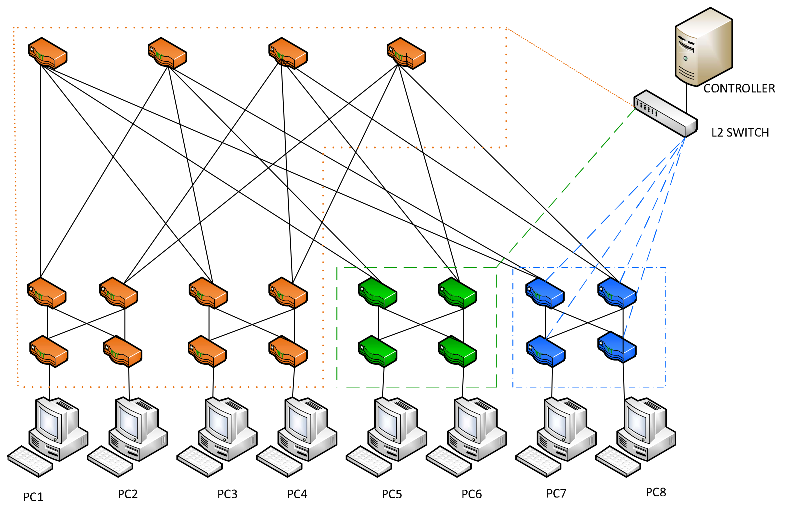

Fig. 1. SDN Data Center topology

Software solutions of OpenFlow switches are Open vSwitch (OVS) software instances [16], running on HP Z620 Workstations with Centos 6 Operating System. These switches are colored in blue in Fig. 1.

Pica8 is white-box L3 switch that can run OVS mode in order to support OpenFlow. Like other Linux-based machines with OVS installed, Pica8 could be logically partitioned into multiple independent OpenFlow switches with dedicated physical interfaces. For our purpose, Pica8 is virtualized into twelve virtual switches (colored in orange in Fig. 1). These virtualized switches are managed in the same way as OVS switches, but have possibility to store the controller's instructions in the hardware.

While white-box switches allow creation of multiple virtual switches in the form of bridges [17] and assignment of physical ports to them, HP ProCurve 6600 switches cannot be virtualized in the same way. In order to create multiple virtual OpenFlow switches from HP ProCurve 6600, a separate VLAN (Virtual Local Area Network) has to be assigned to each virtual switch. In our testbed, HP ProCurve is logically separated into four OpenFlow virtual switches, colored in green in Fig. 1. They have independent configurations and connections towards the OpenFlow controller.

The intelligence of SDN network resides on the controller device, which instructs all OpenFlow switches in its control area. Due to scalability concerns, it is necessary to run controller on machine with respectable performances. The PC that is dedicated for this purpose in our testbed is HP Z620 Workstation with Ubuntu 16.04 OS. It's provided with 16GB of main memory and Intel Xeon E5-2640 CPU with 24 physical cores and 2 threads per core. On this machine we installed four open-source controllers: Floodlight, ONOS POX and Ryu.

ONOS is Java-based open-source SDN controller developed at Stanford University. It is designed with distributed architecture in order to meet the needs of service providers for scalability, high availability and performance. The system offers REST (REpresentational State Transfer) API, CLI (Command Line Interface) and an extensible, dynamic web-based GUI (Graphical User Interface). It also supports multi-threading.

Floodlight is multi-threaded Java-based SDN controller. It is considered as one of the enterprise class OpenFlow controllers that is easy to use, build and run. It is supported by developers all around the world, and Big Switch Networks Company.

POX is single-threaded user-friendly SDN controller, whose components are written in Python. It is widely used in education and research as a learning and prototyping tool. The components of POX controller are programmed to implement different networking functions.

Ryu is another python-based OpenFlow controller that which received wide support from research institutions and vendors. It supports management protocols such as NetConf and OF-Config, and several versions of OpenFlow protocol, from 1.0 to 1.5. This is important advantage over POX, which supports only OpenFlow 1.0.

3. EXPERIMENTAL METHODOLOGY

In this section we explain experimental methodology behind performance evaluation of our testbed.

Considering that processing packets at high rates with minimum latency is a key requirement for any SDN controller, we used throughput and processing latency as key indicators of SDN controller's performance. To compare performance of ONOS, Floodlight, POX and Ryu we used Cbench benchmarking tool. Cbench emulates OpenFlow switches which communicate with the controller. As input arguments it takes a number of switches to emulate, the number of hosts per switch and the controller's address. It supports two working modes: latency and throughput mode. In the latency mode, it sends PACKET_IN messages to the controller. These messages are used in OpenFlow protocol to inform the controller when a packet received by a switch does not match any entry in the Flow Table. As a response, controller generates FLOW_MOD message, which contains a new entry for the flow table. If a controller cannot make routing decision, (e.g. because packet destination is unknown), it will just instruct switch to drop or flood a packet via PACKET_OUT message, depending on the control application used. When Cbench works in the latency mode, it measures the time needed to handle a single packet. The next

PACKET_IN message is generated only after a corresponding response is received from the controller. In throughput mode, the emulated switches send as many PACKET_IN messages as possible to the controller, making sure that the controller always has messages to process [18]. Thus, the results of both, throughput and latency tests, are expressed in number of received responses per second. In order to examine how multi-threading impacts controllers' performances, we conducted multiple experiments where controllers have been run with different number of threads (using *taskset* Linux command).

While Cbench is very compelling tool for testing controller scalability, it just emulates bunch of independent OpenFlow switches, which are not connected in any specific network topology. Thus, in order to get indication regarding the performance of our testbed (Fig. 1), we conducted several additional experiments. Our first experiment involved running ping between hosts reachable by 1, 3 and 5 hops. The purpose of this experiment was to contemplate the effect of the switch-controller communication on ping RTT.

In second set of experiments we measured the impact of controller-switch RTT delay on transfer time of TCP flow. For this purpose, we installed simple client-server application on PC1 and PC8. At the beginning of the experiment we ensure that flow tables of OpenFlow switches are empty. Then we run the client-server application that measures time from the moment when TCP connection is established until the connection is closed [19].

In third set of experiments we generated UDP traffic and measured packet loss during the first second of UDP transfer. Traffic is generated between hosts PC1 and PC8 via Iperf software [20]. We varied the rate of UDP traffic.

Each of the experiments explained above is repeated under conditions of increased controller-switch delay of 5ms and 10ms. The additional propagation delay between the controller and switches is emulated by using Linux *tc* command. This allows one to observe how controller-switch delay degrades quality of the network service.

At the beginning of each experiment, all ARP caches and Flow Tables in the network were empty.

4. THE EXPERIMENTAL RESULTS

Fig. 2(a) shows the results of Cbench latency test when ONOS, Floodlight (FDL), POX and Ryu controllers were run with single CPU thread, for a different number of emulated switches. The results indicate that Floodlight and ONOS controllers process incoming PACKET_IN messages significantly faster than POX and Ryu. It should be noted that all controllers were running simple L2 learning control script for the sake of fairness and simplicity. From the figure, one can observe that ONOS performs slightly better than Floodlight when number of switches is smaller than 64. For larger number of switches Floodlight outperforms ONOS. Difference in performance between ONOS and Floodlight on one side, and POX and Ryu on the other side, are most obvious from the Fig. 2(b), which shows results of the Cbench latency test when 5 CPU threads were used to run controller software. Since POX and Ryu does not support multi-threading, their performance did not improve at all. Latency performance of ONOS and Floodlight increases drastically.

The results of the Cbench throughput test are shown in **Error! Reference source not found.**. One can see that for less than 128 switches, ONOS and Floodlight performed almost the same. In scenario with 128 switches, there is a sudden drop in throughput of Floodlight controller. Again, the worst results are achieved with POX and Ryu. Thus, one can conclude that these controllers are not suitable for large-scale production environments. The results from **Error! Reference source not found.**(b) go in line with this claim. It can be observed how Floodlight and ONOS can exploit good hardware capabilities of the host machine (multiple threads) to speed up processing.

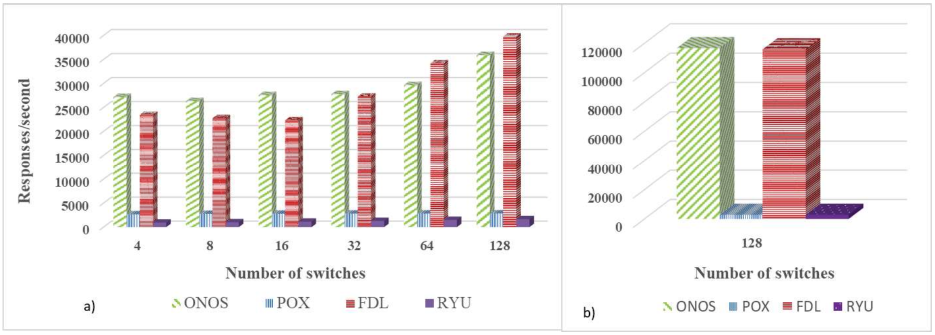

Fig. 2. Results of the latency tests: (a) Single-threaded; (b) Multi-threaded

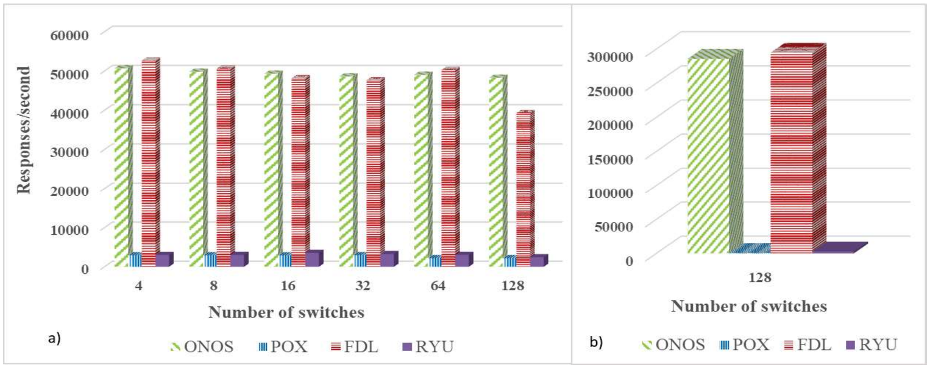

Fig. 3. Results of the Cbench throughput test: (a) Single-threaded; (b) Multi-threaded

In the rest of the section we discuss results of the experiments performed on the DC network testbed. Fig. 3 and 5 show the results of the ping experiment for different values of switch-controller delay. Ping was run between PC5 and PC6 (one-hop route), between PC7 and PC8 (three hop route) and between PC1 and PC8 (five hop route). The results from Fig. 3(a) refer to RTT of the first ICMP Echo Request/Reply transfer for cases with and without additional delay on the controller-switch link. Since ARP tables of hosts and flow tables of switches were empty at the beginning of the experiment, RTT for first ICMP request is approximately equal to three times controller-switch RTT delay increased by packet processing time. This is because switch seeks instructions from the controller three times during the transfer period of first ICMP Echo Request. Firstly, access switch encapsulates ARP request from the sending host in PACKET_IN message. The controller responses by flooding ARP packet, because location of destination is still unknown. Then, access switch of the destination host sends ARP reply in the form of PACKET_IN. Now location of the packet destination is known, thus, controller installs Flow Table rules for ARP packets. The third interaction with controller happens when ICMP request reaches the access switch of the sending host. Controller installs routing rules as a response [5].

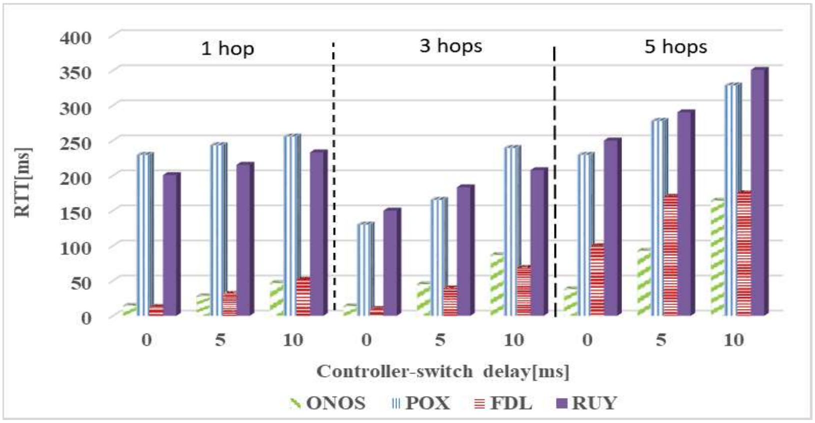

Fig. 3. Comparison of the average RTTs for the first ICMP packet.

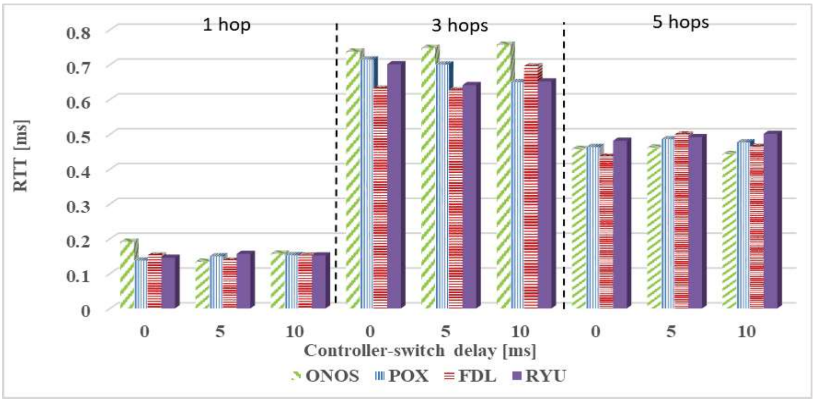

Fig. 5. Comparison of the average RTTs for the second ICMP packet.

Fig. 4 confirms the conclusion of our previous Cbench experiments that POX and Ryu introduce the highest processing latency. Fig. 5 shows RTT measurements for the second ICMP packet during the same ping experiment. These results are interesting because they give indication regarding the data plane performance. In this case, switches do not communicate with the controller (the route already exists), so RTT depends only on delay introduced by the data plane. It can be noted that delay in 3-hop experiments is larger than in 5-hop experiments. This could be attributed to the fact that 3-hop route goes over 3 software OpenFlow switches (OVS), which have more limited performance than Pica8 and HP switches.

Fig. 6 shows results for the client-server experiments explained in Section 3. The server process was running on PC8, while PC1 was used as a client. As a performance indicator we used transfer time of the data file. In experiments we varied size of the file and initial delay on control links. In this testing scenario, initial interaction with the controller slows down the procedure of TCP handshaking and route setup. In case of Floodlight and ONOS controllers, the additional propagation delay on control links was the main cause of the increased transfer time of TCP flow. On the other side, when POX and Ryu are used, transfer time is large even when propagation delay between controller and switch is negligible [5].

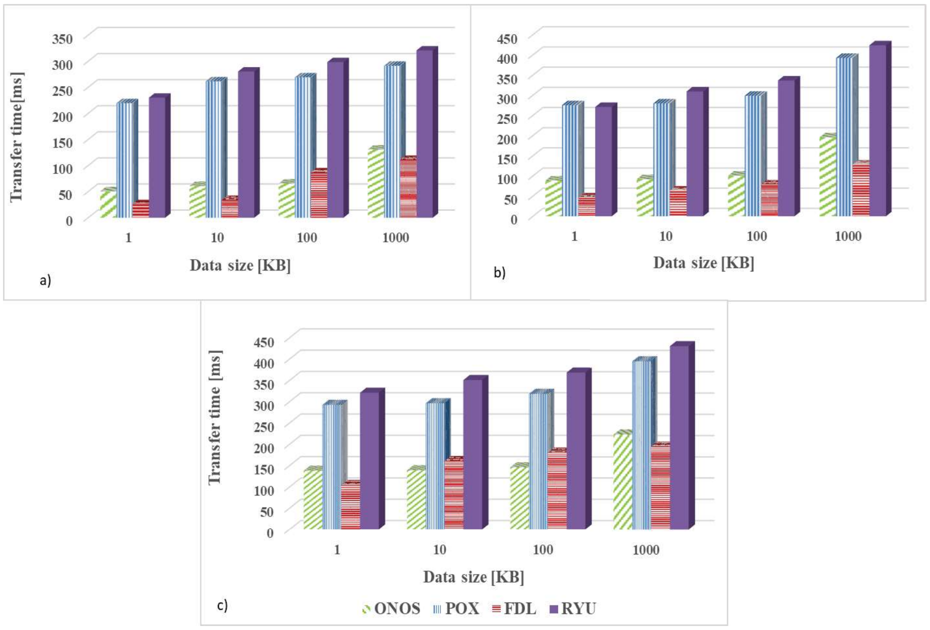

Fig. 6. Average TCP transfer time for different file sizes in experiments with added controller-switch delay of (a) 0 ms, (b) 5 ms and (c) 10 ms

The results of UDP experiment are presented in 7. The graphs show packet loss rate during the first second of UDP transfer in function of the data rate and added controller-switch delay. Considering that UDP doesn't use handshaking procedure, when a first packet of UDP flow enters the network, there are no routes installed and packet is directed towards the controller. The actual problem is that not only the first packet is directed to the controller, but also all subsequent packets until the Flow Table rule is installed. In case of large and high-speed UDP flows, packets can overburden the controller and got lost. The results from Fig. 7 show that packet loss rate increases with the UDP data rate. For data rates higher than 800 Mb/s the network gets unstable, with very high level of packet loss rate. This network bottlenecks are not only software switches but also HP ProCurve switch, which according to the results cannot support data rates over 850 Mb/s.

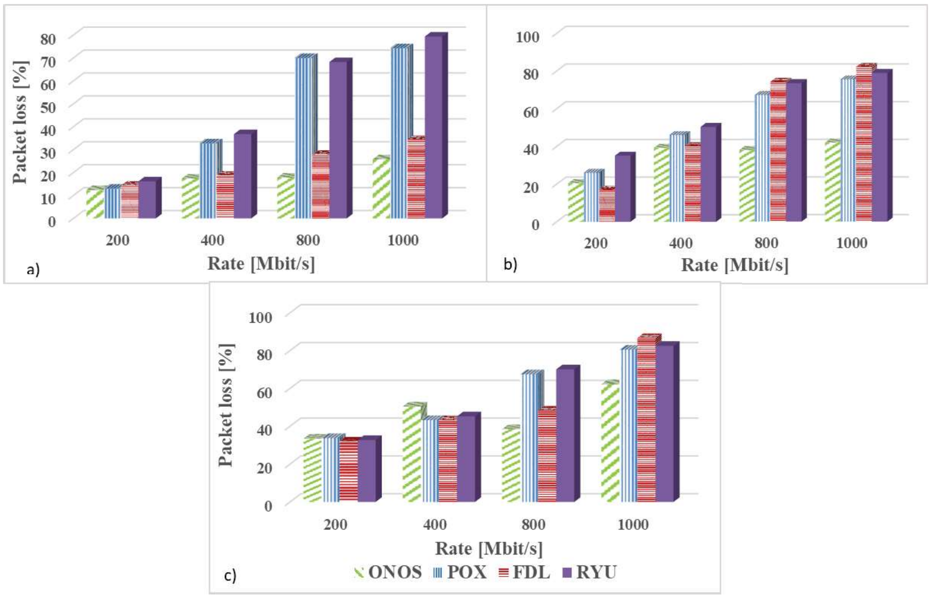

Fig. 7. Packet loss rate during the first second of UDP transfer for different data rates in experiments with added controller-switch delay of (a) 0 ms, (b) 5ms and (c) 10 ms.

5. CONCLUSIONS

In this paper, we explained heterogeneous implementation of DC OpenFlow network testbed and presented results of experiments in which testbed performance was verified. The data plane of the testbed consists of two hardware and four software OpenFlow switches. The hardware switches are logically partitioned in 16 virtual OpenFlow switches. In this way, compact and high performance SDN testbed is created, connected in large-scale DC network topology. In order to determine the most suitable OpenFlow controller, we tested four open-source solutions: ONOS, Floodlight, POX and Ryu. Performances of the controllers in terms of throughput and processing latency have been tested via Cbench software, which allows emulation of thousands OpenFlow switches and host connected. The results show that ONOS and Floodlight outperform POX and Ryu significantly. In scenarios with large number of emulated OpenFlow switches, Floodlight introduces the lowest processing delay, but cannot cope with large amount of switch queries as good as ONOS. More complete insight into the testbed performance is obtained through ping, TCP transfer and UDP transfer experiments. In our future work, we will use the testbed for experimental evaluation of traffic engineering applications.

ACKNOWLEDGMENT

This work has been supported by the Montenegrin Ministry of Science and the HERIC project through the BIO-ICT Centre of Excellence (Contract No. 01-1001).

REFERENCES

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

[2] S. Tomovic, K. Yoshigoe, I. Maljevic, and I. Radusinovic, „Software-Defined Fog Network Architecture for IoT.," Wirel. Pers. Commun., Vol. 92, No1, pp. 181-196, January 2017.

[3] Open Networking Foundation - OpenFlow v1.4 specification. [Online]. Available: https://www.opennetworking.org/, 2017.

[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee "DevoFlow: Scaling Flow Management for High-Performance Networks," SIGCOMM, pp. 254-265, August 2011.

[5] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N Huang, and A. Vahdat,"Hedera: dynamic flow scheduling for data center networks," In Proc. of the 7th USENIX conference on Networked systems design and implementation, CA, USA, 2010.

[6] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: low-overhead data-center traffic management using end-host-based elephant detection", in *Proc. 30th IEEE Int. Conf. Comp. Commun (INFOCOM 2011)*, Shanghai, China, pp. 1629–163, 2011.

[7] BIO-ICT Centre of Excellence: http://www.bio-ict.ac.me/.

[8] ONOS controller: http://onosproject.org/

[9] Floodlight controller: http://www.projectfloodlight.org/floodlight/

[10] POX controller: https://github.com/noxrepo/pox.

[11] Ryu controller: https://osrg.github.io/ryu/.

[12] C. Laissaoui, N. Idboufker, R. Elassali, K. El Baamrani, „A measurement of the response times of various OpenFlow/SDN controllers with Cbench, AICCSA 2015, pp. 1-2.

[13] H. D. Vu and J. But, "How RTT Between the Control and Data Plane on a SDN Network Impacts on the Perceived Performance", International Telecommunication Networks and Applications Conference (ITNAC), pp. 1-4, 2015.

[14] Pica8: http://www.pica8.com/documents/pica8-datasheet-48x1gbe-p3290-p3295.pdf

[15] HP Procurve:http://h20564.www2.hpe.com/hpsc/doc/public/display?docId=emr_na-c01840731

[16] OpenvSwitch: http://openvswitch.org/

[17] Bridge networking. Web tutorial: http://www.innervoice.in/blogs/2013/12/02/linux-bridge-virtual-networking/

[18] O. Salman Imad H. Elhajj Ayman Kayssi Ali Chehab, "SDN Controllers: A Comparative Study," Lebanon: American University of Beirut, pp. 1-6, 2016.
[19]A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, R. Smeliansky, "Advanced Study of SDN/OpenFlow controllers," Proc. of the 9th Central & Eastern European Soft. Eng. Conference, 2013, pp. 3-9.