Journal Of Research Technology & Engineering
www.jrte.org

PERFORMANCE COMPARISON OF ONOS AND ODL CONTROLLERS IN SOFTWARE DEFINED NETWORKS UNDER DIFFERENT NETWORK TYPOLOGIES

*Maheshi B. Dissanayake[1], A. L. V. Kumari[1,2,3], and U.K.A. Udunuwara[2]

[1] Dept. of Electrical and Electronic Engineering, Faculty of Engineering, University of Peradeniya, Sri Lanka.
[2]Sri Lanka Telecom PLC, Colombo.
[3] Ceylon Electricity Board, Colombo.

*maheshid@eng.pdn.ac.lk

Received:24 June 2021; Revised: 04 July 2021; Accepted:06 July 2021; Available online: 10 July 2021

Abstract: A controller is a fundamental component of the Software-Defined Network (SDN) architecture that will contribute to the success or failure of SDN. Although it can be implemented using different open source and proprietary tools, selecting the best would eventually contribute to the success of implementation. Therefore, there is a need to assess and compare other existing controllers for SDN in the market and research domains. In this research, a comparative performance analysis between Open Network Operating System (ONOS) and Open Daylight (ODL) open source SDN controllers is carried out by installing the latest stable version of ODL, ODL-Nitrogen and ONOS, ONOS-Nightingale on a virtual test environment, Mininet. The initial packet latency, average round-trip time and Transmission Control Protocol (TCP) bandwidth, i.e. throughput, are measured with respect to network topology in Mininet emulator using ping and *iperf* commands. The experimental results show that in terms of latency performance ONOS controller is more efficient compared to ODL controller, ODL controller has better flow-setup latency performance than ONOS controller. Jitter variation indicates that ONOS controller is more consistent and leads to more stable network connection and the *iperf* measured TCP bandwidth shows that ONOS Controller processing power is better than ODL controller processing power. Overall analysis illustrates that ONOS is more robust than ODL in our setup.

Keywords: Jitter, latency, network topologies, Open source SDN controllers, Software Defined Network (SDN), throughput,

1 INTRODUCTION

Software-defined networking (SDN) architecture is developed with the expectation of providing agility, flexibility, and virtualization of a communication network. SDN improves network control by enabling enterprises and service providers to respond quickly to changing business requirements by separating the control plane of the network. Understanding the performance of the control plane in SDNs is crucial, as it acts as the "brain" of the network impacting the performance of the entire network, applications, services, etc. Thus, understanding the behavior of the SDN controller will assist the users and future developers to leverage this technology effectively.

Several studies have been conducted with the aim of comparing SDN controllers in the past couple of years. Majority of them have conducted benchmarking study on some centralized controllers such as Maestro [1], Beacon [2], Floodlight [3], Ryu [4], NOX [5] and POX [6]. These studies were carried out to identify the baseline performance of controllers, and determine which controller outperforms the others under certain test scenarios.

In a highly cited study, [7] evaluated the performance of Beacon, Maestro, and NOX. The study conducted by [8] provides a set of requirements that are at the base of the comparison between the controllers, and compares five controllers using a Multi-Criteria Decision Making (MCDM) method named Analytic Hierarchy Process (AHP). Fernandez, M. J. in [9] focused on the operation modes for OpenFlow controllers and evaluated the performance of Floodlight, NOX, POX, and Trema in both reactive and proactive modes. Ref. [10] studied OpenFlow controller's architecture in a customized testbed and benchmarked Beacon, Floodlight, Maestro, and NOX controllers with different performance metrics. A preliminary study of the Floodlight and Open-Daylight (ODL) controllers using *Cbench* was conducted in [11]. Shalimov et. al in [12] conducted advanced research on SDN with Controllers NOX, POX, Beacon, Floodlight, MUL, Maestro and Ryu using a tool named *hcprobe*. In this study, the maximum throughput was demonstrated by Beacon. Furthermore, [13] presents a performance study of centralized OpenFlow controllers, Beacon, Floodlight, IRIS, Libuid MSG, Libuid RAW, Maestro, MUL, NOX, POX, and Ryu, and distributed OpenFlow controllers, Open Network Operating System (ONOS) and ODL. The authors have used *WCbench*, for their evaluation. In another performance study, [14] evaluated on the performance of Beacon, Floodlight, NOX, POX, and Ryu.

The controller efficiency is measured through different parameters such as performance, scalability, reliability and security that characterize a controller. In this research, we evaluate only the performance of the controllers. To the best of our knowledge, very few studies in the literature focus on the performance evaluation of ONOS [15] and ODL [16], SDN controllers by taking active measurements. Most of the prior works have considered prototyping a new controller, or evaluated centralized OpenFlow controllers. ONOS and ODL have multiple releases and are mature enough in their development phase. In [17], authors experimentally assessed the performances of ONOS and ODL SDN controllers in terms of topology discovery and topology update. The comparison between the two SDN controllers presented in [18], is neither extensive nor exhaustive. The authors focus specifically on the controllers' path restoration and software reliability while not covering the many other aspects that may interest an industrialist. The performance of the ODL and ONOS in terms of burst rate, throughput, Round Trip Time (RTT) and bandwidth is studied in [19]. Yet, the research only considers the tree topology and the experiment is design differs from the methodology adopted in this manuscript. To better understand performance improvements (if any), it is important to re-evaluate new releases of these controllers. We are hopeful that this study would add more depth to the performance of ODL and ONOS SDN controls under different topological settings.

The rest of the paper is structured as follows. Section 2 briefly introduces the two popular open source SDN controller ONOS and ODL. Then section 3 presents the methodology adopted in this research, focusing on experimental setup and performance matrices used to access the performance of the ONOS and ODL. Comparative results and in depth discussions are presented in Section 4. We analyzed the performance of the two open source controllers under different networking topologies, using initial packet latency, jitter, average round-trip time, and throughput. Finally, the conclusions are drawn at section 5.

2 ONOS AND ODL OVERVIEW

This section provides a brief overview to the two well known open source SDN controllers, ONOS and ODL

2.1 ONOS

ONOS is written in Java and provides a distributed SDN applications platform atop Apache Karaf OSGi (Open Services Gateway Initiative) container. The system offers REST API, CLI and an extensible, dynamic web-based GUI. Applications (core extensions) can be loaded and unloaded dynamically, via REST API or GUI, and without the need to restart the cluster or its individual nodes. The ONOS architecture is with tiers of functionality, as outlined in [15] .

The top layer of ONOS architecture consists of business and network logic applications. The pivotal layer in the ONOS architectural tier is the Distributed core, which allows physical separation of data and control functions. This layer is also responsible for presenting a logically centralized view of the network state and a logically centralized access to network control functions.

The core is separated from the other tiers via two logically distinct interface boundaries namely Southbound API and Northbound API. Southbound API is the south-facing interface and it is a high-level API through which the core interacts with the network environment. ONOS core relies on protocol-specific adapters to conduct these interactions using the protocols of their choice, whether it is OpenFlow, NETCONF, OVSDB, TL1 or even other available means, such as CLIs. Northbound API conveys information about the network from the low-level topology abstractions, such as hosts, links and devices, to higher-level abstractions, such as the network topology graph to application layer and provide management interface for controlling lower layer components. ONOS interacts with the underlying network through its Providers. Providers are ONOS applications based on OSGi components. These applications can be dynamically enabled or disabled at run time. The primary purpose of providers is to abstract the configuration, control, and management operations of a specific family of devices (e.g. OpenFlow, NETCONF, etc.). Further, protocols used with ONOS, defines implementation of all features needed by the controller to communicate with real devices, for examples: OpenFlow, NETCONF, SNMP, etc.

2.2 ODL

ODL is an open source SDN controller platform implemented in Java, which can be deployed on any operating system platform as a Java Virtual Machine (JVM). The key features of architecture of ODL SDN Controller can be described as follows; Model-Driven Service Abstraction Layer (MD-SAL) is the Kernel of the ODL which uses yet another Next Generation (YANG) as the modeling language. The south-bound protocol of ODL is based on Open Service Gateway Initiative (OSGi) architecture. OSGI is known as the Dynamic Module System for Java, and it defines a standard for modular application development. ODL contains several components and projects, and consists of three layers. The top layer contains business and network logic applications used by the controller to gather network

intelligence, run algorithms to control and monitor network behavior. The Base network service functions, extensions, and platform services exist in controller platform. Controller provides an abstraction which allows to developer to focus on the development of application functionality rather than involving with writing device specific drivers. And the bottom layer consists of data plane elements such as virtual switches, physical device interfaces. More detailed description on these features can be found in [20]. ODL supports multiple protocols in southbound interface such as OpenFlow 1.0, OpenFlow 1.3, BGP-LS, LISP, SNMP, etc. as ODL is created with an objective of reducing vendor locking

3 METHODOLOGY

The test environment described in this section was set up on a single personal Computer (PC) with Intel Core i5-4200M 2.50 GHz CPU with 2 cores, 4 logical processors, and 8 GB RAM using virtualization technology. Oracle VirtualBox installed on the PC, was used to create virtual machines. Fig.1. and Fig. 2. illustrate the test bed with virtualization. Two virtual machines were created in the VirtualBox, and named ONOS-Ntg and ODL-Ny. 64-bit Ubuntu 18.04 desktop-amd64 operating system was installed on both virtual machines. ONOS-Ntg machine was installed with ONOS-Nightingale [20] controller and Mininet, whereas ODL-Ny was installed with ODL-Nitrogen [21] controller and Mininet. Both virtual machines created with 4 GB RAM and 10 GB Storage.

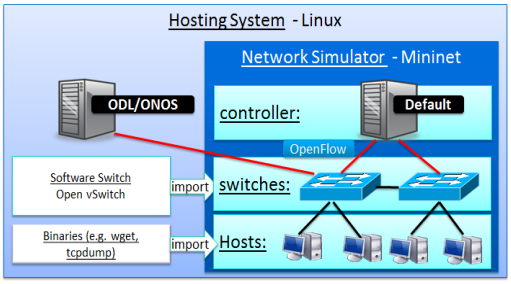


Fig. 1. Testbed Setup

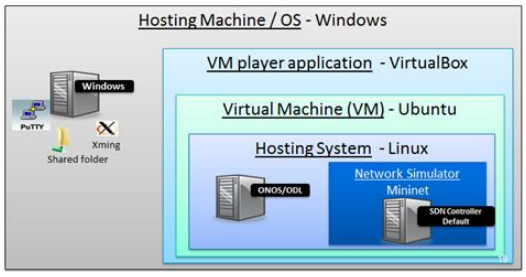


Fig. 2. Virtual Machine SDN Test Environment

3.1 Experimental method

Test configurations are arranged for standalone controller supporting reactive mode of flow setup. Fig. 3 depicts the common test setup for the standalone controller. Southbound Connection is a single OpenFlow channel established between each OpenFlow switch and controller, since single controller (standalone) manages a network. Given that a controller takes a considerable period of time to perform tests for various types of traffic and packet sizes, we used traffic type IP with ICMP message of size 64 bytes to benchmark SDN controller performance. All experiments were carried out for both controllers tested in two standalone test setups, and the performance metrics latency, ping delays, jitter, and throughput of the networks were measured. At the setting up of the experiments, virtual traffic flow is generated to load and unload the network topology tested, to facilitate performance measurements. In the "unloaded network" test case, the only traffic presented in the network is 64 byte ICMP traffic. In the "loaded network" test scenario, the network is loaded with two 64 Kilobyte ICMP traffic and the network performance is measured using 64 byte ICMP traffic.

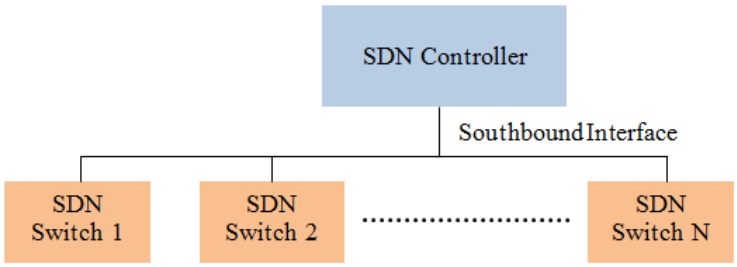


Fig. 3. Test setup for Standalone SDN controller in reactive mode of operation

3.2 Performance metrics with Topology Change

SDN controller is connected with the data plane emulated in Mininet with different network topologies. Six different topologies were created using python application programming interface (API) in Mininet. The topologies adopted, namely; Linear, Tree, Torus, Ring, Fat-tree and datacenterHAFull are depicted in Fig. 4. Then three sets of ten ICMP requests (ping message) were sent between the hosts which were logically apart from each other, and the results were recorded as in Table 1.

Table 1. Topology details

Topology	No of nodes			No. of links
	Switches	Hosts	Controller	
Linear	16	16	1	31
Tree	15	16	1	30
Torus	16	16	1	48
Ring	16	16	1	32
Fat-tree	14	16	1	41
datacenterHAFull	16	14	1	44

The very first ping delay was taken to compare the flow setup delay between two hosts which are logically separated from each other in each topology. The last set of ten ICMP messages were taken, to record Average Round Trip Time (ARTT) values and to calculate Jitter. The throughput is measured by running the iperf command in Mininet command-line interface (CLI) console which return the TCP bandwidth. All experiments run in two phases using the same network topology. In first phase the ODL controller was connected to the data plane and in the second phase the ONOS controller was connected to the data plane. At each run the performance metrics Latency, Jitter and Throughput were measured.

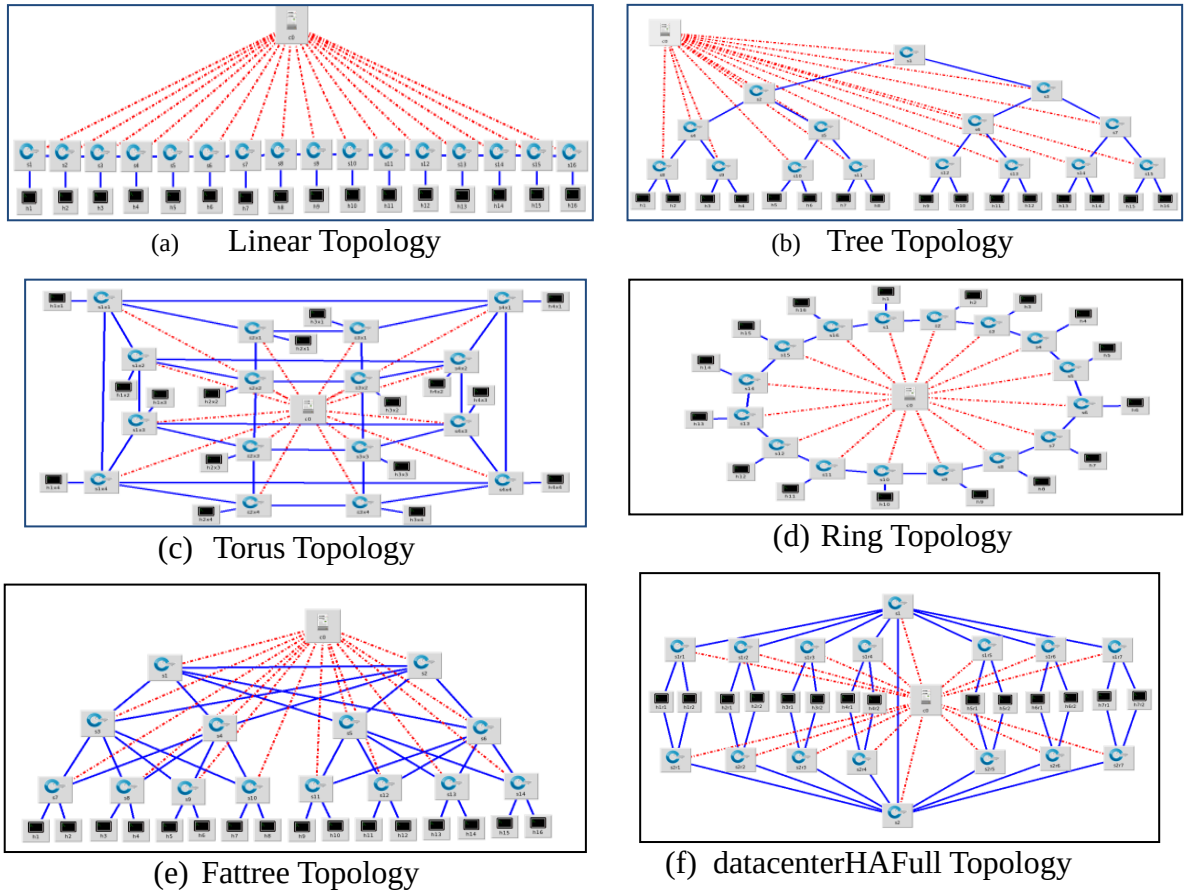


Fig. 4. Network Topologies tested in the research presented

4 Results and Discussion

4.1 Latency

Latency is the time required for a packet to arrive at its destination through the network. Latency can be measured either using the time needed for a packet to reach its destination or the round-trip time. Ping delay is employed to estimate on how close the platform is to real life implementation. There are two important types of ping delays. They are the Initial Ping Delay (IPD) and the Average Ping Delay (APD). The IPD is highly affected by the time it takes for the controller to add a flow table rule to the OF switch.

The APD is defined as the average of all the ping delays excluding IPD. Using an SDN experimental platform without initializing a controller is unrealistic and cannot be compared to real-life. On the other hand, adding the huge IPD into the APD will result in a significant increase that will not reflect the entire test as well as real-life implementation with pro-active controllers. Thus, in the analysis presented the ping delay was split into IPD and APD, and the results of IPD taken at the very first ping in each topology was analyzed.

4.1.1 Average Ping Delay (APD)

A single experimental round consisted of running the command *ping* ten times on a specific path in the topology. The path used was the logically longest path (e.g. h1 to h16 in the case of the Linear and Torus topologies and h1 to h8 in Ring Topology). Fig. 5. and Table 2., depict the average Round Trip Time (RTT) of 10 pings in each network topology connected to ONOS and ODL .

Table 2. Average Round Trip Time -10 pings

Topology \ Controller	ODL / ms	ONOS / ms
Linear	0.455	0.265
Tree	0.155	0.151
Torus	0.349	0.115
Ring	0.255	0.186
Fat-tree	0.411	0.106
datacenterHAFull	0.436	0.114

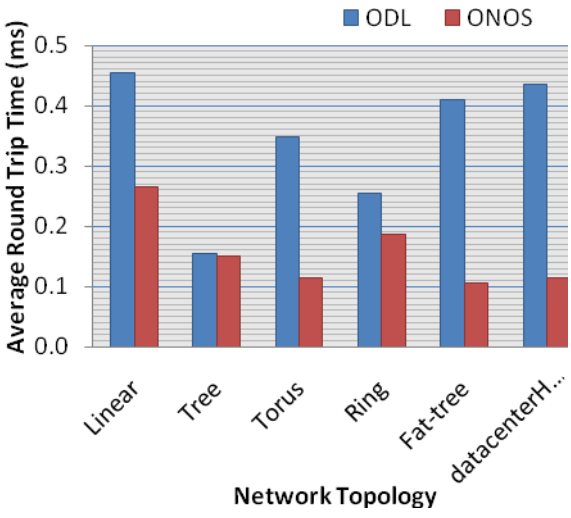


Fig. 5. Average Round Trip Time -10 pings

According to Fig. 5., the ONOS controller's average ping times are always slower than those of the ODL controller. While this in general is by a large margin, in the case of the Tree topologies, the margin for the average ping time is significantly small. This deduces, that packets sent under the ODL controller clearly take far more time to travel in all topologies except in tree topology than the time taken under the ONOS

controller. ODL's latency performance suggests it is not an efficient controller compared to ONOS controller.

Close examination yields, that ONOS controller shows smaller RTT values with the topologies which have more redundant links or multi paths, such as Fat-tree, datacenterHAFull. Hence, it is evident that ONOS controller has a better algorithm to find the shortest path to the destination. In contrast, ODL controller experienced larger RTT values with the topologies which have more redundant links or multi paths, like Fat-tree, and datacenterHAFull. This large margin indicates the ODLs' capability to finding the shortest path to destination is poor. Hence, ODL controller exhibits lower performance than the ONOS controller in terms of RTT.

Both controllers have largest average RTT in Linear network topology as ping messages need to pass through all (16 nos) OF switches to reach its destination (h16).

4.1.2 Initial Ping Delay (IPD)

Our results for the initial ping time are presented in Fig. 6. and Table 3. These results were obtained by tracing the time taken for the very first ping sent, after Mininet was started with a topology-controller combination, to complete the loop. The first ping is always higher than the next one because it is the first time that a packet browses the network to discover the path. Hence, the initial ping delay gives an estimate of how much time each controller algorithm needs as processing time, to understand the underlying network topology.

Table 3. Initial ping Delay

Topology \ Controller	ODL / ms	ONOS / ms
Linear	5.95	413.00
Tree	2059.00	312.00
Torus	10.00	192.00
Ring	5.39	408.00
Fat-tree	2060.00	334.00
datacenterHAFull	4.72	199.00

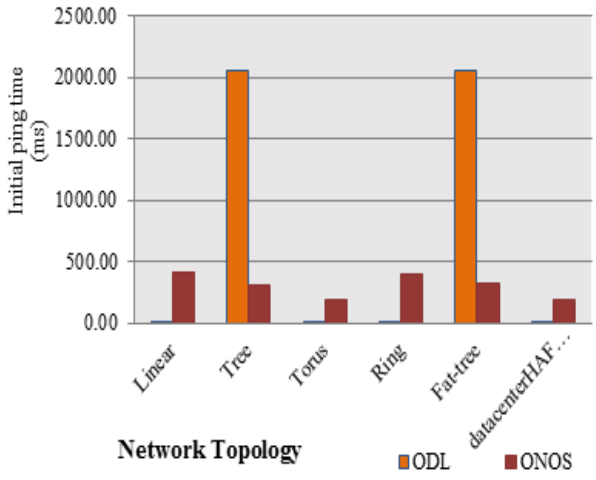


Fig. 6. Initial Ping Delay

According to Fig. 6., majority of network topologies (4 out of 6) connected to ODL controller have low initial ping delay but, surprisingly Tree and Fat-tree topologies connected to ODL controller shows very large initial ping delay. There exists consistency in initial ping delays of topologies connected to ONOS controller. Except for Tree and Fat-Tree topologies, the ODL controller has better flow-setup latency performance than ONOS controller.

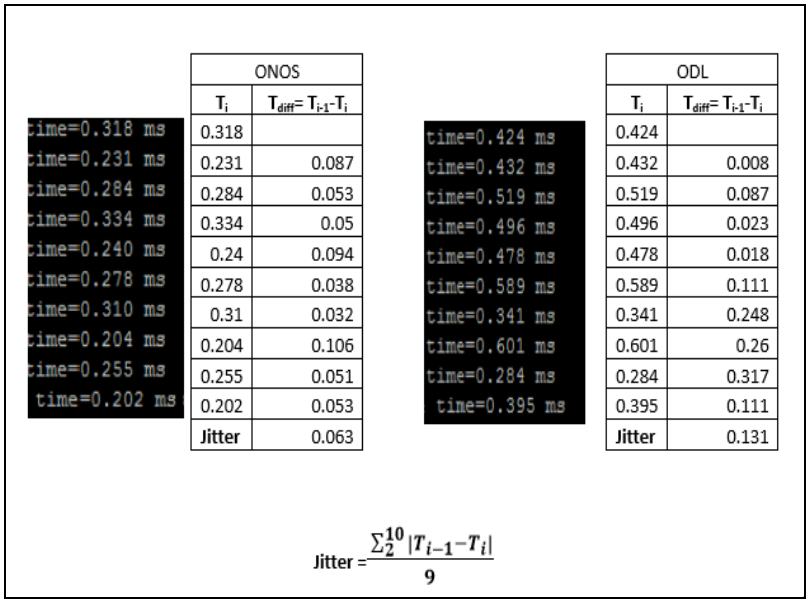
In essence, considering the APD and IPD results, the ONOS controller is more favorable to deploy in the networks than ODL. This may be due to ONOS consisting of a complex series of subsystems, as well as scalable functions, in contrast to ODL, which uses a model-view-control platform and operates off of a strong central abstraction layer. Further, it should be noted that the ONOS processing power is greater than the ODL processing power.

4.2 Jitter

Next a comparison between the jitter value of each controller is carried out to observe the performance of ODL and ONOS. Jitter is the irregularity in latency on data packets flow over a network. Similar to the latency test setting, in this test also the network topology was varied while keeping the number of switches and hosts around 16 to keep all network topologies with equal network equipment which ease the comparison of controller performance with respect to topology change. We have calculated the jitter using 10 ping messages obtained at the 3rd round of pinging. The process followed to calculate the jitter is illustrated in Fig. 7. while Fig. 8. shows the jitter values of network topologies with each controller.

In ODL controller connected network topologies, jitter varies between 0.031 ms to 0.202 ms and in ONOS controller connected network topologies, it varies between 0.028 ms to 0.069 ms. As seen in Fig. 8. the jitter variation with respect to network topologies is significant in ODL controller compared to ONOS controller, which indicates that ONOS controller is more consistent and lead to more stable network connection.

Although majority of ONOS controller connected topologies show lower jitter, in Torus, Ring and Hypercube topologies which contain loops, the jitter performance is better in ODL controller than ONOS controller. Hence, from these test results, it can be deduced that ONOS controller is better than the ODL controller in terms of jitter performance. The main reasons for this observations is that, although ODL and ONOS both have hybrid commercial strategies, the ONOS subsystems has been developed focusing on the telecoms, while ODL focused more on data centers. ONOS defines its services as a collection of sub systems, for instance, the topology subsystem manages time-ordered snapshots of network graph views, host subsystem manages the inventory of end-station hosts and their locations on the network etc. which ease ONOS controller to find and send network traffic consistently than ODL.



ONOS	
T_i	$T_{diff} = T_{i-1} - T_i$
0.318	
0.231	0.087
0.284	0.053
0.334	0.05
0.24	0.094
0.278	0.038
0.31	0.032
0.204	0.106
0.255	0.051
0.202	0.053
Jitter	0.063

ODL	
T_i	$T_{diff} = T_{i-1} - T_i$
0.424	
0.432	0.008
0.519	0.087
0.496	0.023
0.478	0.018
0.589	0.111
0.341	0.248
0.601	0.26
0.284	0.317
0.395	0.111
Jitter	0.131

Fig. 7. The procedure followed to calculate Jitter

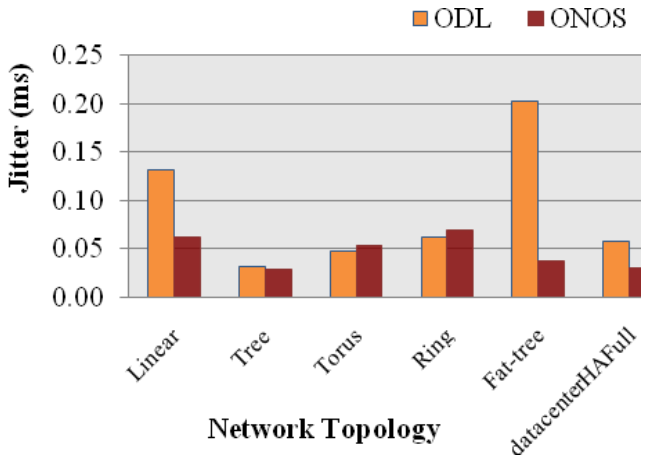


Fig. 8. Comparison of Jitter for ODL and ONOS controllers under different network topologies

4.3 Throughput

Throughput determines the number of flow requests a controller can process per unit time. Similar to the earlier test settings, the network topologies were varied while keeping the number of switches and hosts closed to 16 to maintain same scale network which makes the comparison of controller performances uniform with respect to topology change. Table 4. presents the average throughput results for ONOS and ODL, respectively.

Send and Response Rate parameter was utilized to measure the throughput. The *iperf* bandwidth-testing tool was run on *Mininet* console to obtain an estimate of network throughput for each topology-controller combination, providing an alternate means of measuring the networks potential to send traffic quickly. This command output two-element array of defined server and client speeds which are considered for the measurement of Throughput.

As of Table 4. it can be observed that *iperf* measurement based TCP throughput of the ONOS connected topologies are far higher than that of the ODL controller connected topologies. In the virtual test

environment, the TCP throughput measurement is solely dependent on the controller processing power. This allows us to affirm, in terms of throughput, ONOS offers better performance. This is likely because of its inherent support for very large scale networks.

Table 4. Throughput values for different network topologies

Controller / topologies	ODL **Server** Throughput / Gbits/s	ONOS **Server** Throughput / Gbits/s	ODL **Client** Throughput / Gbits/s	ONOS **Client** Throughput / Gbits/s
Linear	2.55	9.43	2.55	9.45
Tree	2.90	14.80	2.90	14.90
Torus	1.55	16.50	1.55	16.50
Ring	1.23	12.10	1.23	12.10
Fat-tree	2.19	14.60	2.20	14.60
datacenterHAFull	1.23	19.50	1.23	19.50

5 CONCLUSIONS

Understanding the performance of the control plane in software-defined networks is crucial, as it acts as the "Brain" of the network impacting the performance of the entire network, applications, and services. In this research, a comparative performance analysis of most popular open source SDN controllers, ONOS and ODL is carried out by considering different network topologies. An impartial experimental analysis based on active measurement was carried out for the selected Standalone reactive mode controllers by executing *ping* and *iperf* commands in *Mininet* console. Our experiments indicate that the latest stable release of ONOS, ONOS-Nightingale, outperforms ODL, ODL-Nitrogen, for throughput and latency with different network topologies. In our test settings, the communications between the switches and the controller are carried out through the loopback interface, and there is no delay while packets traverse the virtual ports as compared to the hardware ports. Hence, it should be noted that, the latency and throughput values observed in real implementations, may tend to be higher than the values presented in this study. In summary, ONOS shows more robust performance than ODL in the tested scenario.

REFERENCES

[1] Cai Z. , Cox A. L. , and Ng T. S. E. . Maestro: "A System for Scalable OpenFlow Control". Rice University, Tech. Rep. 2011.
[2] Erickson, D. "The Beacon OpenFlow controller", *HotSDN '13 Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.* New York: ACM, pp. 13-18, 2013.
[3] Ryan Izard. (2013). "Project Floodlight. Big Switch Networks", Accessed December 2017. http://www.projectfloodlight.org/.
[4] NTT. (2013). Ryu Network Operating System, Retrieved from http://osrg.github.com/ryu
[5] Gude N. ,Koponen T. , Pettit J. , Pfaff B., Casado M. , McKeown N. , and Shenker S. "NOX: towards an operating system for networks", Computer Communication Review, 2008,
[6] McCauley, M. (2012). POX. http://www.noxrepo.org/.

[7] Tootoonchian A. , Gorbunov S., Ganjali Y.,Casado M. , and Sherwood R. “On Controller Performance in Software-Defined Network”, 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, 12, San Jose, California, pp. 1-6. 2012
[8] Khondoker R. , Zaalouk A., Marx R. and Bayarou K. . “Feature-based comparison and selection of Software Defined Networking (SDN) controllers”, World Congress on Computer Applications and Information Systems (WCCAIS), Hammamet, pp. 1-7, 2014.
[9] Fernandez, M. J. “Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive”, 27th IEEE International Conference on Advanced Information Networking and Applications (AINA), Barcelona, Spain, pp. 1009-1016, 2013
[10] Shah S., Faiz J., Farooq M., Sha A. and Mehdi S. “An Architectural Evaluation of SDN Controllers”, In Proceedings of the IEEE International Conference on Communications (ICC), Budapest, Hungary, pp. 3504 - 3508, 2013.
[11] Khattak Z., Awais M. , and Iqbal A. “Performance Evaluation of OpenDaylight SDN Controller”, In Proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), Hsinchu, Taiwan pp. 671-676, December 2014
[12] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, “Advanced Study of SDN/OpenFlow Controllers”, in 9th ACM Central \& Eastern European Software Engineering Conference, Moscow, Russia, October2013.
[13] Salman O. , Elhajj I. H. , Kayssi A., and Chehab A. “SDN Controllers: A Comparative Study”, 18th IEEE Mediterranean Electrotechnical Conference (MELECON), Lemesos, Cyprus, pp. 1-6, 2016.
[14] Zhao Y. ,Iannone L., and Riguidel M. “On the Performance of SDN Controllers: A Reality Check”, In Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks, San Francisco, California.pp. 79 - 85, 2015
[15] Berde P. , Gerola M. ,Hart J. , Higuchi Y., Kobayashi M. , Koide T., Lantz B. ,OЏonnor B. ,Radoslavov P. ,Snow W. , and Parulkar G. “ONOS: Towards an open, distributed SDN OS” in Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, ser. HotSDN Ұ4. New York: ACM, pp. 1-6, 2014.
[16] OpenDaylight, “OpenDaylight: A Linux Foundation Collaborative Project,” (2017). Retrieved from http://www.opendaylight.org
[17] Bah. M. T., A. Azzouni, M. T. Nguyen and G. Pujolle, “Topology Discovery Performance Evaluation of OpenDaylight and ONOS Controllers”, 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 2019, pp. 285-291, 2019.
[18] Secci, S., Diamanti, A., Sanchez, J., Vilchez, M., Bah, M.T., Vizarreta, P., Machuca, C., Scott-Hayward,S. & Smith, D., “Security and Performance Comparison of ONOS and ODL controllers”, Information Report, Open Networking Foundation, 2019.
[19] Badotra, S., and Panda, S.N. “Evaluation and comparison of OpenDayLight and open networking operating system in software-defined networking.” Cluster Comput, 2019.