IST-Africa 2021 Conference Proceedings
Miriam Cunningham and Paul Cunningham (Eds)
IST-Africa Institute and IIMC, 2021
ISBN: 978-1-905824-66-3

A QoS-based Evaluation of SDN Controllers: ONOS and OpenDayLight

Lusani MAMUSHIANE, Themba SHOZI
Council for Scientific and Industrial Research (CSIR), Pretoria, 0081, South Africa
Tel: +27 842 7838, Email: Lravhuanzwo@csir.co.za

Abstract: SDN marks a paradigm shift towards an externalized and logically centralized controller, unlike the legacy networks where control and data planes are tightly coupled. The controller has a comprehensive view of the network, offering flexibility to enforce new traffic engineering policies and easing automation. In SDN, a high performance controller is required for efficient traffic management. In this paper, we conduct a performance evaluation of two distributed SDN controllers, namely ONOS and OpenDayLight. Specifically, we use the Mininet emulation environment to emulate different topologies and the D-ITG traffic generator to evaluate aforementioned controllers based on metrics such as delay, jitter and packet loss. The experimental results show that ONOS provides a significantly higher latency, jitter and low packet loss than OpenDayLight in all topologies. We attribute the poor performance of OpenDayLight to its excessive CPU utilization and propose the use of Hyper-threading to improve its performance. This work provides practitioners in the telecoms industry with guidelines towards making informed controller selection decisions

Keywords: SDN; distributed controllers; ONOS; Mininet; OpenDayLight; D-ITG; delay; jitter, packet loss.

1. Introduction

Software Defined Networking (SDN) has emerged as one of the key enablers to unlock the promises of the envisaged fifth generation of mobile networks (5G), namely, enhanced mobile broadband services, massive machine type communications, and ultra-reliable and low-latency communications. SDN decouples the network's control logic from the data forwarding plane and provides an abstracted global view of the network status to ensure efficient orchestration of network services.

This results in a three-tier architecture where the lower tier is the data plane (slave), middle tier is the control plane (brain) and top tier is the application layer. Between this tiers are abstraction layers namely, the northbound interface (NBI) and southbound interface (SBI). The NBI is used by applications such as OpenStack, load balancing, and intrusion detections to enforce high-level policies on the underlying hardware via the controller. Based on these policies, the controller uses the SBI to program the forwarding behaviorof the data plane for optimal traffic steering. Some of the SBIs supported by SDN controllers include but not limited to, OpenFlow, PCEP, LISP, BGP-LS, etc. [1]. A prevalent choice for a NBI is the RESTful networking protocol [1].

To date, a plethora of SDN controllers has been proposed within the networking research community. These can be divided into two categories: centralized controllers and distributed controllers. Ryu and Floodlight are the topmost famous centralized SDN controllers. These controllers were designed primarily for rapid prototyping on campus networks. The main drawback of these controllers is that they are a single point of attack/failure and they are not easily scalable. Distributed controllers have been designed to address these limitations. Among all open source distributed controllers, ONOS and OpenDayLight have received significant attention due to their modularity, multi-vendor support and high availability. In a distributed SDN environment, controllers use the east/westbound interface (E/WBI) to exchange network state information with their peers. This is to ensure strong consistency semantics of state information across the network. A widely adopted E/WBI is the external border gateway protocol (EBGP) networking protocol [2].

As the broker between the application plane and data plane, the controller performance is one of the most critical design metrics. In order to guarantee high QoS, the controller should be able to respond to packet in messages promptly. This means that the average latency (processing + queuing +propagation latency), jitter and packet loss must be minimal and throughput must be maximum. This work aims to evaluate the performance of the latest releases of ONOS (Junco version) and OpenDayLight (Oxygen version), to investigate if these controllers are ready for prime time deployment. To the best of our knowledge, there is no recent study exclusively focusing on benchmarking ONOS and OpenDayLight. Another motivation for this study is the fact that today's IT world is slowly transitioning towards virtual network infrastructures. Therefore conducting a performance evaluation of ONOS and OpenDayLight on a virtualized environment is quite timely and useful. The main tools used for the experimentation work are Mininet (a network emulation tool) and a Distributed Internet Traffic Generator (D-ITG).

The remainder of this paper is structured as follows. Section II reviews prior related work on SDN controller evaluation. Section III gives an overview of the controllers under evaluation as well as the tools used for the experiment. Section IV describes the experiment setup. Section V presents the results from our experiments and a discussion. Finally, Section VI concludes the paper.

2. Related Work

There currently exist a vast number of studies focusing on controller performance evaluation. Nonetheless, most of these evaluations featured controllers that are now considered obsolete. For instance studies in [3], [4], [5], [6] and [7] featured controllers such as Beacon, Maestro, NOT-NT and NOX, which are outdated. These studies were conducted to identify the baseline performance of controllers, and determine the best controllers in terms of metrics such as latency, throughput, security, and feature support. The study in [3], carried out a featured-based comparison of SDN controllers using an multi-criteria decision making method called analytical hierarchy process (AHP). In [4] and [5] a tool called Cbench was used to benchmark the controller performance, and in [6] Hcprobe was used to benchmark both performance and security vulnerability of SDN controllers.

The most recent works on SDN controller evaluation is by Salman et al. [8], Rowshanrad et al. [9], Zhu et al. [25] and Rastogi et al. [10]. Salman et al. [8] evaluated the performance of well-known centralized and distributed SDN controllers (MUL, Beacon, Maestro, ONOS, Ryu, OpenDayLight, Floodlight, NOX, IRIS, Libfluid-based, and POX). The results from this study indicated that controllers coded in C (such as Mul and Libfluid) have the best throughput performance, followed by java-coded controllers such as Maestro and Beacon. Authors attribute this performance to the fact that java and C programming languages support multi-threading, while python is virtually unaffected by an increase in number of threads.

Rowshanrad et al. [9] evaluated and compared the performance of Floodlight and OpenDayLight. The analyzed indexes include performance of the controllers when subjected to various topologies (single, linear and tree topology). To do this, the authors stressed the controllers with different traffic loads including VoIP and DNS. From their results, OpenDayLight exhibited the best latency results under low traffic loads (for single and linear topologies) as well as for tree topologies when subjected to medium traffic loads. Floodlight exhibited lower packet loss under high traffic volumes for tree topologies and the best latency results in linear topologies.

Zhu et al. presented a qualitative comparison of 34 SDN controllers, along with a quantitative analysis of their performance in different deployment scenarios. The authors present detailed description and comparison of SDN controller benchmarking tools as well as their capabilities and limitations. The conclusions from their benchmarking experiments is that the limitations of benchmarking tools has a direct impact on measured results. The authors also concluded that distributed controllers perform significantly better than centralized and single-threaded controllers.

Lastly, Rastogi et al. [10] carried out a performance evaluation of POX and Ryu when subjected to layer 1 and layer 2 switching, where layer 1 involves switching in hub mode and layer 2 has learning intelligence. Bitrate was used as a performance indicator. The results of the evaluation show that POX is better than Ryu when layer 1 is in question and vice versa as far as layer 2 is concerned. The author conclude that POX is better at handling functions like broadcasting ARP requests, while Ryu is better at packet routing.

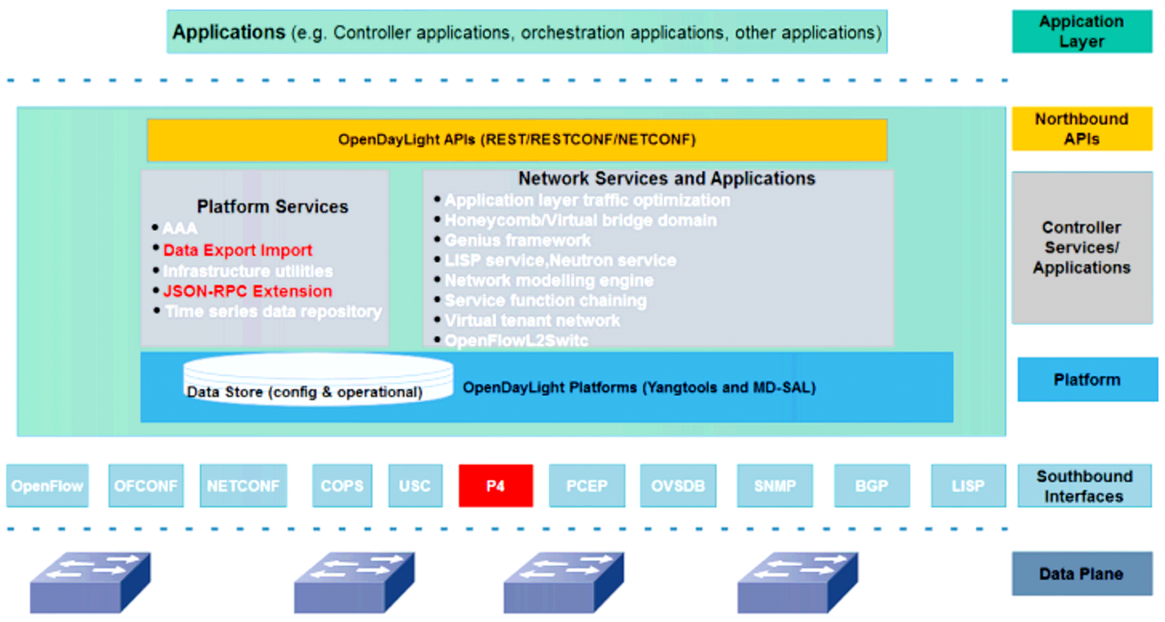
To the best of our knowledge, most studies on controller evaluation did not exclusively feature distributed controllers namely, ONOS and OpenDaylight. These controllers have emerged as the ideal candidates for real SDN deployments. Both this controllers have multiple releases and are matured in their development. Therefore, it is important to compare the performance differences of these controllers since they both offer compelling benefits from a feature based comparison. This work can be utilized by application developers and service providers to make informed controller selection decisions

3. Overview of OpenDayLight, ONOS, Mininet and D-ITG

3.1 OpenDayLight

OpenDayLight is a modular open source java-based SDN controller hosted by the Linux Foundation, used for customizing and automating networks of any scale and size [11]. This controller leverages a model-driven software engineering principle (MDSE) known as Model-driven service abstraction layer (MD-SAL), which uses YANG as the data modelling language for service and data abstractions. In OpenDayLight, the underlying hardware is represented as objects or models whose interactions are managed by the SAL. The modular architecture of OpenDayLight offers users and solution providers free rein to tailor a controller to satisfy their needs.

The MD-SAL resides within the control layer and is the “brain” of the SDN network. Its northbound interface translates policies from the application layer to the data layer via its southbound interface (see Figure 1). OpenDayLight supports a wide range of southbound protocols such as OpenFlow, BGP-LS, PCEP, LISP, NETCONF, OVSDB, etc. In terms of adoption coverage, OpenDayLight is at the core of open source management and orchestration frameworks such as ONAP [12], OpenStack [13] and OPNFV [14], as well as standard development organizations such as metro Ethernet Forum (MEF). For instance the UNI Manager [15] plugin release provides APIs for MEF’s Lifecycle Service Orchestration (LSO) project [16] [17]. To date there are a total of 9 OpenDayLight releases namely (in order of decreasing age), Hydrogen, Helium, Lithium, Beryllium, Boron, Carbon, Nitrogen, Oxygen and Fluorine. Each new release promises support of emerging use cases such as IoT, integrated NFV management and S3P (Security, Scalability, Stability and performance) achieved through clustering and federation.



Applications (e.g. Controller applications, orchestration applications, other applications)
Application Layer
OpenDayLight APIs (REST/RESTCONF/NETCONF)
Northbound APIs
Platform Services
AAA
Data Export Import
Infrastructure utilities
JSON-RPC Extension
Time series data repository
Network Services and Applications
Application layer traffic optimization
Honeycomb/Virtual bridge domain
Genius framework
LISP service,Neutron service
Network modelling engine
Service function chaining
Virtual tenant network
OpenFlowL2Switc
Controller Services/ Applications
Data Store (config & operational)
OpenDayLight Platforms (Yangtools and MD-SAL)
Platform
OpenFlow
OFCONF
NETCONF
COPS
USC
P4
PCEP
OVSDB
SNMP
BGP
LISP
Southbound Interfaces
Data Plane

Figure 1: OpenDayLight Architecture (Release: Oxygen)

3.2 ONOS

ONOS is an open source SDN controller, pioneered by the ON.Lab and primarily designed to enable service providers to build real SDN solutions. Similar to OpenDayLight ONOS has been optimized (through its distributed core) to deliver features such as network scalability, reliability and high performance (latency and throughput), which are indispensable in production networks. ONOS has two abstraction frameworks in its northbound interface namely, the intent framework [18] and global network topology view [19]. The Intent Framework is a subsystem that enables a network application to apply a service in the network in form of policy (i.e. what should be done) rather than mechanism (how it should be done).

The global network view exposes the current status of the entire network (e.g. resource utilizations) to the application layer. The controller northbound abstracts the complexity of the underlying hardware from the application. Another abstraction is enabled by the southbound interface which represents the underlying hardware as objects, and allows convergence of disparate data plane devices through its support for different protocol plugins, e.g. NETCONF, OVSDB and OpenFlow.

The core use case of ONOS is the Central Office re-architected as Datacenter (CORD), which capitalizes on SDN, NFV and cloud computing to transform the central office through hardware commoditization [20]. This features enable operators to achieve both the economies of scale (infrastructure build from generic servers) and business agility (rapid deployment and elastic scaling of network services to meet current demands) which are benefits currently enjoyed by cloud service providers. Currently there are a total of 9 releases of ONOS, namely, Avocet, Blackbird, Cardinal, Drake, Emu Falcon, Goldeneye, Hummingbird, Ibis, Junco which proves the extent of community support.

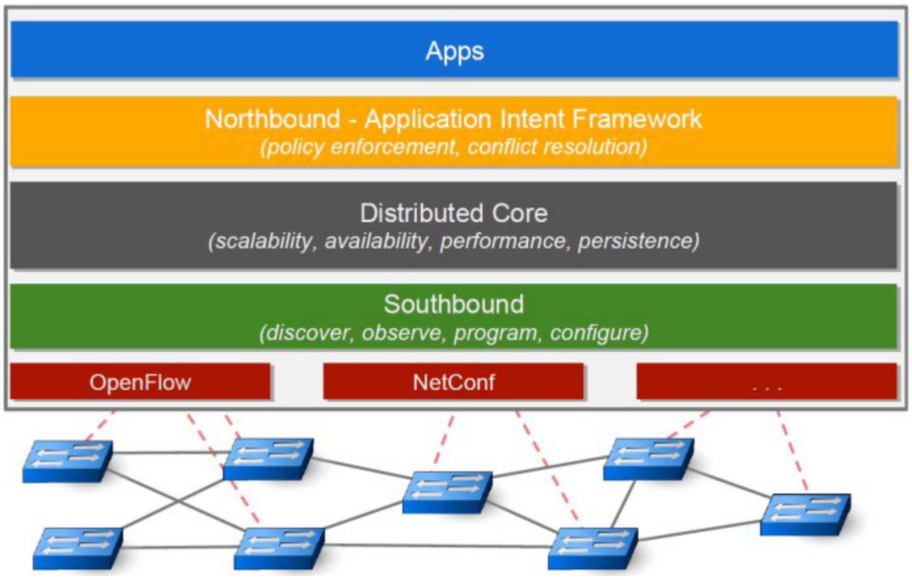

Apps
Northbound - Application Intent Framework
(policy enforcement, conflict resolution)
Distributed Core
(scalability, availability, performance, persistence)
Southbound
(discover, observe, program, configure)
OpenFlow
NetConf
. . .

Figure 2: ONOS high-level architecture.

3.3 D-ITG

D-ITG is a platform used to generate IPv4 and IPv6 traffic by accurately reproducing the workload of active Internet applications [21]. This platform is able to mimic the statistical properties of various application-level protocols such as VoIP- G.711, G.723, G.729, DNS, and Telnet. At the transport layer, D-ITG can monitor various transport protocols such as TCP (Transmission Control Protocol), UDP (User Datagram Protocol), SCTP (Stream Control Transmission Protocol), DCCP (Datagram Congestion Control Protocol) and ICMP (Internet Control Message Protocol). At packet level, D-ITG is able to monitor and analyze traffic flows so as to measure common performance metrics such as throughput, jitters, one-way-delay (OWD), round-trip-time (RTT) and packet loss.

As illustrated in Figure 3 below, D-ITG constitutes 5 components namely, ITGSend, ITGRecv, ITGManager, ITGLog, and ITGDec [22] [23]. ITGSend is responsible for generation of traffic flows and can operate in (i) single-flow (ITGSend generates only one flow toward a single instance of ITGRecv), (ii) multi-flow (ITGSend generates multiple flows towards one or more instances of ITGRecv) or (iii) daemon mode (run as daemon listening on a UDP port for flow instructions). The ITGRecv component is responsible for receiving one or multiple traffic flows generated by one or more ITGSend instances. ITGLog stores log information received over TCP or UDP from ITGSend and ITGRecv. ITGDec is used for decoding and analyzing the log files of experiments. Last but certainly not least, ITGManager is used for remote control of ITGSend from a single vantage point using the D-ITG API.

The decision to use D-ITG as the traffic generation platform was made after reviewing a study conducted by Kolahi et. al [26]. The authors found that D-ITG supports more measurement metrics, protocols, runs in dual-stack and supports both Linux and Windows platforms. Moreover, D-ITG is capable of producing traffic with various sizes and probability distribution, which is not the case with its rivals, such as Iperf, Netperf and IP traffic.

Controller performance can also be evaluated using SDN controller benchmarking tools as carried out in [7]. To the best of our knowledge, SDN controller benchmarking tools support a limited number of measurement metrics, namely throughput and latency. Moreover, these benchmarking tools do not allow test engineers to evaluate controller performance under different topological configurations. This paper intends to measure more than the aforementioned metrics and features measurements of metrics such as packet drop and jitter under different topological configurations.

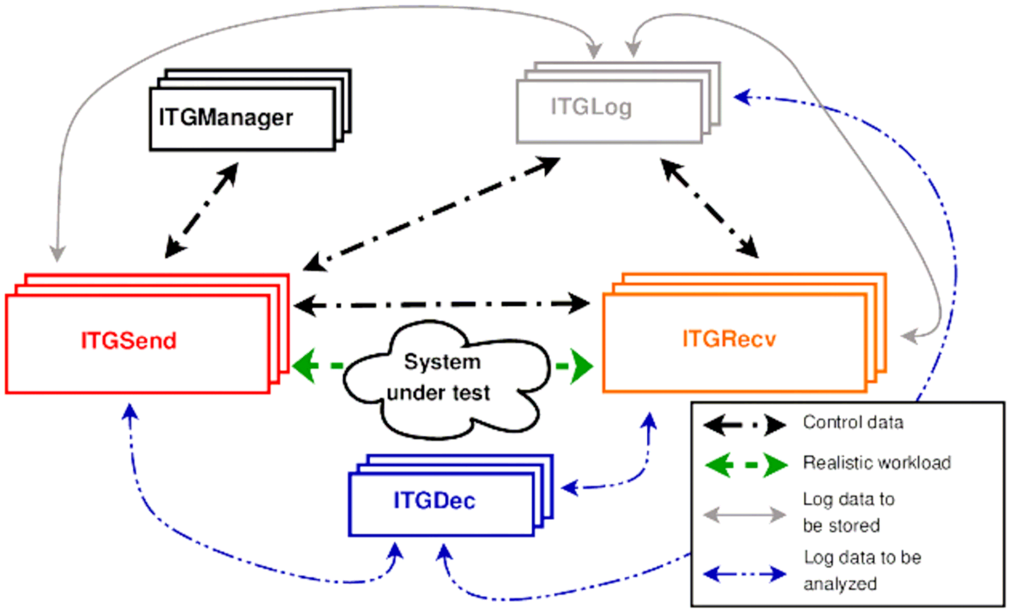


Figure 3: D-ITG Architecture

3.4 Mininet

Mininet is a network emulation orchestration platform commonly used to create realistic virtual networks on a single machine (virtual machine, cloud or native) for rapid prototyping of OpenFlow and SDN related solutions. Mininet is used to run OpenFlow controllers and emulates the SDN data plane using OpenFlow virtual kernel switches (OVS), links and end hosts. Hosts on Mininet are able to run underlying Linux and file system commands [24]. For instance, "iperf" generates and analyses traffic between a client and a server, and "topo" creates a virtual network topology using the Python API.

Mininet comes standard with three network topology configurations which are: single, linear and tree. A single topology constitutes one virtual switch with a customizable number of hosts. A linear topology has serial switch connections with a customizable number of hosts. Last but not least, a tree topology constitutes multiple topology levels with a customizable number of levels (depth and fanout). However, Mininet has unlimited support for user defined topology configurations. One of the biggest advantages of Mininet is that prototypes built on Mininet can be moved to a real network environment with minimal changes on the software code.

4. Experimental Setup

Figure 4 illustrates the experiment setup. All the experiments are performed on a 64-bit Ubuntu machine with 16G RAM and i7 processor. In order to evaluate the performance of ONOS and OpenDayLight, Mininet is used to emulate three topologies, namely, single, linear and tree topology. The topologies are built from simple software switches, called Open Vswitches (OVS). The latest release of Open Vswitch (OVS 2.9.90) with link bandwidth up to 10 Mbps FDX is used in the experiment. OpenFlow version 1.3 is used as the southbound protocol for control traffic. Each topology was configured to feature a total of 8 hosts. Host h1 was configured as the flow generator (ITGSend) and host h8 as the receiver (ITGRecv). Host h1 and h8 were chosen because between them is the longest path in the network. Before running each test, the new topology was loaded with cleanup to clear all flow table entries.

In order to have a near real experience, D-ITG was used for traffic generation. The controller was connected to the data plane via the loopback interface since it was running on the same virtual machine as the emulated data plane. ITGDec was used to decode the log information for each traffic flow. The parameters for each traffic flow were configured as follows: the payload was set to a constant 1000 bytes, number of packets was 100 000 at a constant inter-departure time, the duration of the generation experiment was set to 10s. The transport layer protocol was set to UDP. The metrics measured are one-way-delay, jitter and packet loss. Each test was repeated 10 times to eliminate noise.

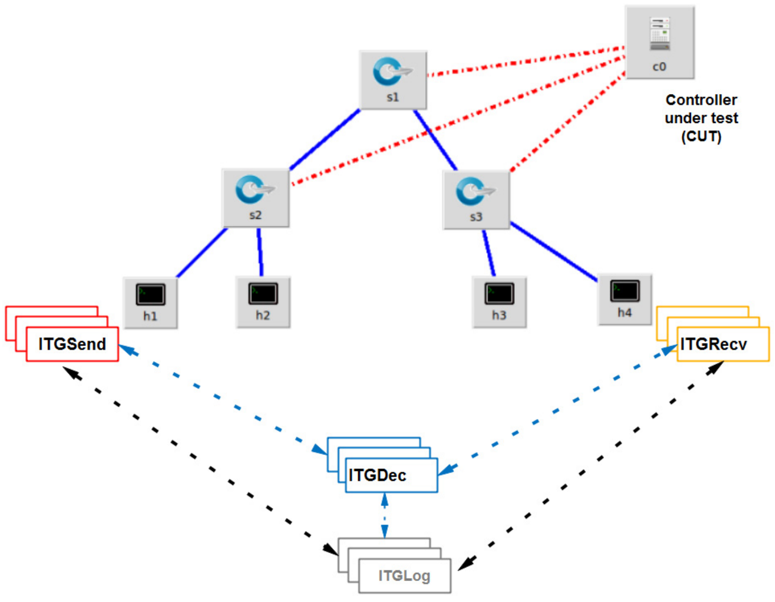


Figure 4: Experiment setup

5. Results and Discussion

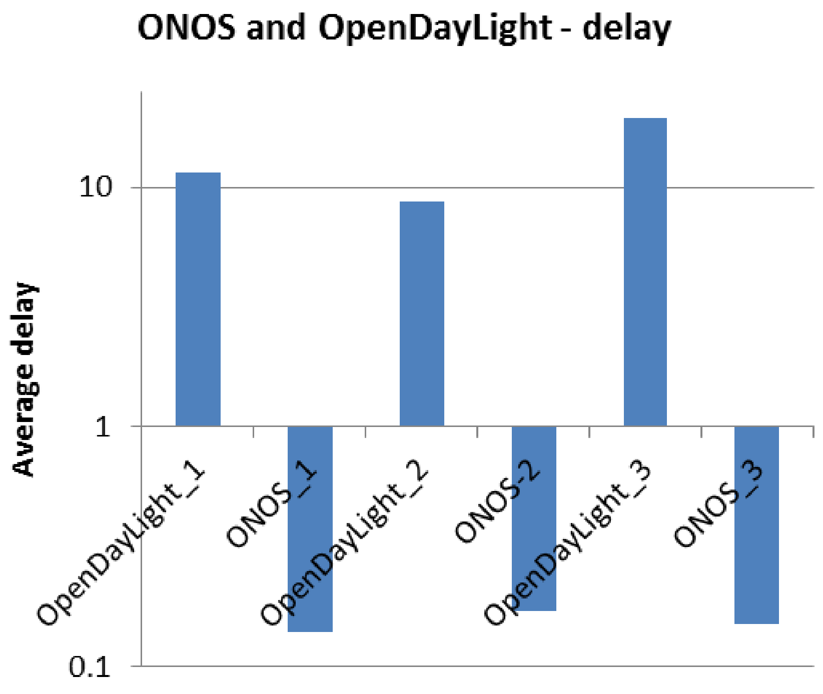
Table I below shows the delay, jitter, packets dropped and standard deviation for each scenario (single, linear and tree topology codenamed 1, 2, and 3 respectively). The standard deviation entry is used to measure the variation of the delay results from the average values. Delay is used to measure the time taken by packets from sender to destination. Jitter is used to measure delay variation.

Table 1: Results from simulation experiments

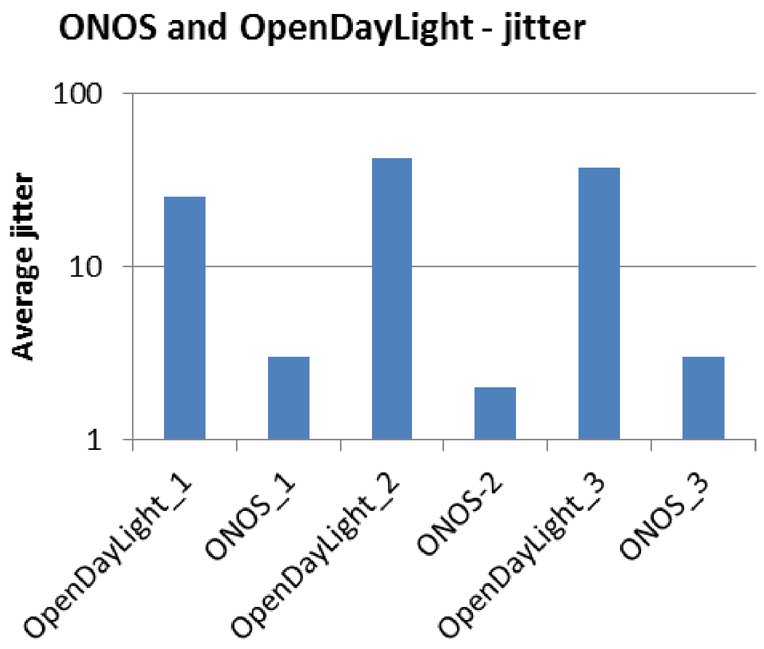
Topology	SDN Controller	Min delay (ms)	Max delay (ms)	Average delay (ms)	Average jitter (ms)	standard deviation (ms)	Packets dropped (%)
Single	OpenDayLight	0.004	41.645	1.145	0.025	4.501	0.25
	ONOS	0.004	2.384	0.014	0.003	0.037	0
Linear	OpenDayLight	0.031	33.231	0.868	0.042	2.453	0.09
	ONOS	0.008	4.035	0.017	0.002	0.063	0.02
Tree	OpenDayLight	0.028	49.956	1.937	0.037	6.392	0.63
	ONOS	0.007	1.212	0.015	0.003	0.016	0

Figure 5 (derived from Table 1) represents a performance comparison between ONOS and OpenDayLight when subjected to all three topologies in question. The X-axis represents topology and controller (for e.g. OpenDayLight-1 is representing OpenDayLight controller with Topology 1). As shown in Figure 5(a), OpenDayLight exhibited the worst delay for all scenarios in comparison with ONOS whose delay is almost 98.7%, 73.4%, and 98.9% better than that of OpenDayLight for the single, linear and tree topology respectively. However, OpenDayLight seems to perform better in linear topologies than in a more complicated topology (tree topology). ONOS on the other hand performs better in tree topologies than linear topologies. In terms of jitter, OpenDayLight still exhibits a higher jitter than ONOS for all scenarios.

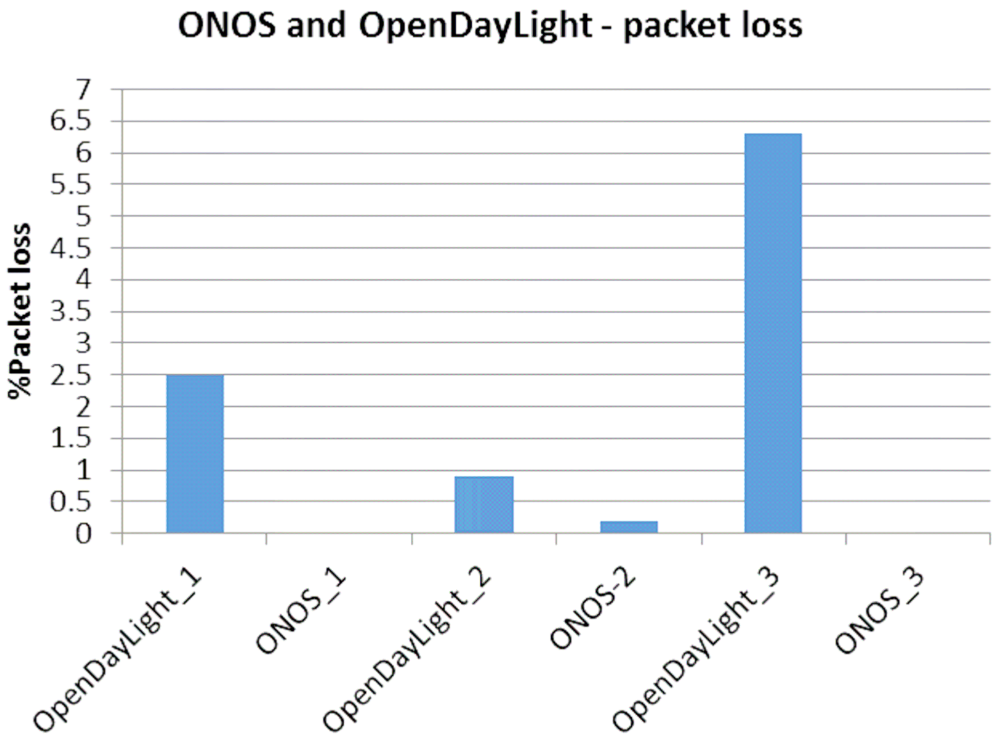
However, for the tree topology, OpenDayLight gave a better jitter than in linear topology. Jitter performance for ONOS is the same for tree and single topologies. From a packet loss viewpoint, OpenDayLight dropped more packets than ONOS for all scenarios, with ONOS only dropping packets in linear topologies. The standard deviation for delay measurement is much higher than that for ONOS. This means for each test done on OpenDayLight, the measured delay values were further away from the average delay.
We believe the reason for this drastic performance difference between ONOS and OpenDayLight might be a result of OpenDayLight excessively high CPU utilization which severely degrades its performance. This means that in order to improve the performance of OpenDayLight, Hyper-threading must be enabled. In other words, instead of using a single thread for computing, multiple threads must be interleaved for computational efficiency.



(a)



(b)



(c)

Figure 5: (a) Delay, (b) jitter and (c) packet loss under varying topologies.

6. Conclusions

This work evaluated two controllers namely, ONOS and OpenDayLight. These controllers are both open-source, modular, and can be programmed in java via a northbound API for new network services. The metrics used to gauge performance of these controllers are one-way-trip-delay, jitter and packet loss evaluated in single, linear and tree topologies. From the evaluations, ONOS exhibited the best performance for all scenarios and metrics. The performance of OpenDayLight was drastically poor in all scenarios. This was attributed to OpenDayLight's inherent excessive CPU utilization. Our recommendation is to enable Hyper-threading when using OpenDayLight to enjoy similar performance benefits as those exhibited by ONOS. We believe our work is useful to facilitate controller selection by service provides, researchers and developers. In future we intend to carry out similar evaluations in an environment running multiple cores with Hyper-threading.

References

[1] "Open Networking Foundation (ONF)," [Online]. Available: https://www.opennetworking.org. [Accessed 2019].

[2] D. R. M. Kreutz, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-defined networking: A comprehensive survey," Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, 2015.

[3] A. Tootoonchian, S. Gorbunov, Y. Ganjali, C. M and R. Sherwood, "On Controller Performance in Software-Defined Networks," Hot-ICE, vol. 12, pp. 1-6, 2012.

[4] R. Khondoker, A. Zaalouk, R. Marx and K. Bayarou, "Feature-based comparison and selection of Software Defined Networking (SDN) controllers," in Computer Applications and Information Systems (WCCAIS), 2014.

[5] S. A. Shah, J. Faiz, M. Farooq, A. Shafi and S. A. Mehdi, "An architectural evaluation of SDN controllers," in Communications (ICC), 2013.

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

[7] A. Lysko, S. Dlamini, L. Mamushiane, "A Comparative Evaluation of the Performance of Popular SDN Controllers," in 2018 Wireless Days (WD), 2018.

[8] O. Salman, I. H. Elhajj, A. Kayssi and A. Chehab, "SDN controllers: A comparative study," in Electrotechnical Conference (MELECON), 2016 18th Mediterranean (pp. 1-6). IEEE, 2016.

[9] S. Rowshanrad, V. Abdi and M. Keshtgari, "Performance evaluation of SDN controllers: Floodlight and OpenDayLight," IIUM Engineering Journal, vol. 17, no. 2, pp. 47-57, 2016.

[10] A. Rastogi and A. Bais, "Comparative Analysis of Software Defined -In Terms of Traffic Handling Capabilities," in Multi-Topic Conference (INMIC), 2016 19th International (pp. 1-6). IEEE., 2016.

[11] Linux Foundation, "OpenDayLight," 2018. [Online]. Available: https//www.opendaylight.org/what-we-do/odl-platform-overview. [Accessed 2020]

[12] Linux Foundation, "ONAP," January 2018. [Online]. Available: https//www.onap.org. [Accessed 2020]

[13] OpenStack, 2018. [Online]. Available: https://www.openstack.org. [Accessed 2020]

[14] "OPNFV," 2018. [Online]. Available: https://www.opnfv.org. [Accessed 2020]

[15] "OpenDayLight," 2017. [Online]. Available: https://www.opendaylight.org/view/unimgr:Documentation.

[16] Linux Foundation, "OpenDayLight," 2018. [Online]. Available: https://www.opendaylight.org/what-we-do/current-release/carbon. [Accessed 2020]

[17] MEF, "Service Operations Specification- Lifecycle Service Orchestration (LSO): Reference Architecture and Framework, page 9,15," 2016.

[18] H. Kumar, C. Russell, V. Sivaraman and S. Banerjee, " A software-defined flexible inter-domain interconnect using ONOS," in Software-Defined Networks (EWSDN), 2016.

[19] ONF, "ONOS project," 2017. [Online]. Available: https://onosproject.org/features/. [Accessed 2020]

[20] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart and G. W. Palukar, " Central office re-architected as a data center," IEEE Communications Magazine, vol. 54, no. 10, pp. 96-101, 2016.

[21] A. Botta, W. de Donato and A. Dainotti, "D-ITG 2.8.1 Manual," CoMputer for Interaction and Communications, 2013.

[22] A. Botta, A. Dainotti and A. Pescapé, "A tool for the generation of realistic network workload for emerging networking scenarios," Computer Networks, vol. 56, no. 15, pp. 3531-3547, 2012.

[23] G. Aceto, A. Botta, W. de Donato and A. Pescapè, "D-ITG: Distributed Internet Traffic Generator," PIK-Praxis der Informationsverarbeitung und Kommunikation, vol. 36, no. 1, pp. 49-49, 2013.

[24] M. Team, "Mininet an instant virtual network on your laptop (or other PC) (2017)," [Online]. Available: http://Mininet.org. [Accessed 2018].

[25] L. Zhu, M.M. Karim, K. Sharif, F. Li, X. Du and M. Guizani, "SDN controllers: Benchmarking & performance evaluation". arXiv preprint arXiv:1902.04491, 2019.

[26] S. S. Kolahi, S. Narayan, D. D. Nguyen, and Y. Sunarto. "Performance monitoring of various network traffic generators." In 2011 UkSim 13th international conference on computer modelling and simulation, pp. 501-506. IEEE, 2011