On the Performance of SDN Controllers in Real World Topologies

Ioannis Koulouras
Dept. Informatics & Telecommunications
University of Ioannina
Arta, Greece
koulouras.iwannis@gmail.com

Spiridoula V. Margariti.
Dept. Informatics & Telecommunications
University of Ioannina
Arta, Greece
smargar@uoi gr

Ilias Bobotsaris
Dept. Informatics & Telecommunications
University of Ioannina
Arta, Greece
i.bobotsaris @uoi.gr .

Eleftherios Stergiou
Dept. Informatics & Telecommunications
University of Ioannina
Arta, Greece
ster@uoi.gr

Chrysostomos Stylios
Dept. Informatics & Telecommunications
University of Ioannina
Arta, Greece
stylios@uoi.gr

2022 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN) | 978-1-6654-7334-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/NFV-SDN56302.2022.9974719

***Abstract*— Software Defined Networking (SDN) represents a new paradigm in networking that shifts the traditional architecture from a fully distributed model to a centralized one. The central principle is the separation of the data and control layers. The logic and intelligence of the control layer is moved to the controller, a software tool, usually open source, used to develop, standardize, and manage network applications. It bridges the communication of network applications and the underlying infrastructure, providing programmable interfaces and other functionalities. In this work, the controllers Ryu, ONOS, OpenDalight and Floodlight were evaluated using the Mininet emulator and the iPerf tool. We focus on the Fat-tree topology, which is a complex topology widely used in data centers. The metrics taken are throughput, latency, and Jitter. The aim is to investigate and compare the performance of the four controllers, providing in-depth analysis and giving insights about their behavior and effectiveness.**

***Keywords*— *Software defined networking, SDN controllers, performance evaluation, Fat-tree* .**

I. INTRODUCTION

These days, the huge volumes of data traffic needs impose the reconsideration of traditional network operations and demand innovative solutions to overcome limitations due to the scale of networks or variations in equipment. Software-Defined Networking (SDN) is a promising approach on network implementation and unfolds new opportunities for solution deployment capable to handle such problems. Towards this objective, the SDN architecture a) separates the data plane and control plane providing respectively flexibility to the forwarding process and intelligence to the routing process [1], b) centralizes control of all the data plane elements [2].

Network behavior is defined by the software that is accommodated on server(s) at a central point beyond the physical forwarding devices. In contrast with traditional networks, SDN is implemented on logically centralized topologies allowing for a global view of the network and providing central management of network resources and services [2]. Applications are running on an abstract and high level, communicate and interact through APIs overcoming the problems posed from multiple vendors' networked devices [3]. These four main characteristics of SDN, programmability, centralized control, openness, and the decoupling of the data plane from the control plane enhance and simplify network functionality, addressing issues in regards to common network services such as routing, access control, traffic engineering, bandwidth management, QoS, energy efficiency, security, reliability and, enforced management policies.

The SDN is vertically divided into three layers, namely an application layer, a control layer and a data layer as illustrated in Fig. 1. The data plane includes the network elements and uses the southbound interface for their communication with the control plane. OpenFlow is the most common standard to serve this type of communications [4]. The application plane accommodates for SDN applications and establishes their instructions or communication requirements with the control plane using the northbound interface.

The control plane is the core of the SDN architecture. It is placed between the other two planes and is responsible of processing application requests and the exertion of low-level control over the underlying network devices. All tasks (routing, topology discovery, load balancing) relative to the control plane are allocated to a software component, called SDN controller. It is a centralized or decentralized application running on one or more servers. The controller aggregates all the network intelligence while having authority to allow for centralized management and control, automation, and policy enforcement in physical and virtual network environments. It has the responsibility to manage data forwarding in the infrastructure. The key factor of success in this complex task, is the performance in terms of throughput, latency, and jitter. However, as there is a plentitude of open-source available controllers, choosing the optimum requires intense work [5], [6].

The companies and organizations, acting in data centers domain, expect benefits from the adoption of SDN and are working together towards its development. In data centers, cheap and commodity switches form complex topologies (e.g., Fat-tree) over which SDN controllers manage the traffic.

In this work, we demonstrate a comparative study of the following selected SDN controllers: ONOS, Ryu, Floodlight and OpenDaylight. We address their performance on Fat-tree topologies, which wide used in data center.

The contribution of this work is (a) a thorough investigation of four different SDN controllers, (b) an experimental comparative study using as key performance indicators, throughput, latency, and jitter and finally (c) an

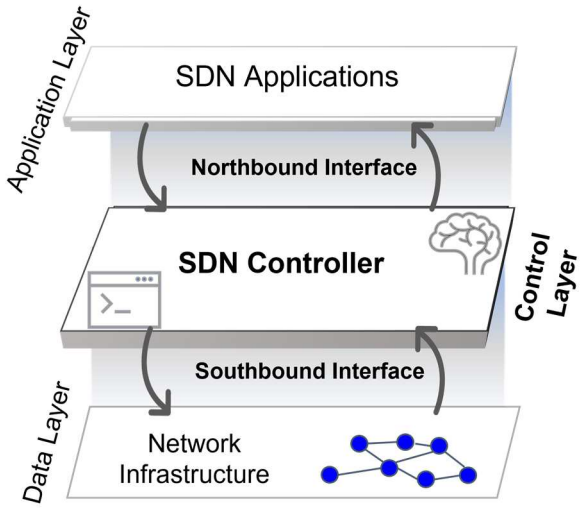

Fig. 1. The general architecture of SDN

evaluation of controllers' behavior on real world topologies used in data center.

The remainder of this paper is organized as follows: In Section II, we review the related literature on the topic of SDN controller comparison. In Section III, we describe the background of the compared controllers. In Section IV, we present the methodology and evaluation framework. In Section V, we present the simulations conducted and discuss the results achieved. Finally, in Section VI, we conclude this work and discuss future work.

II. Related Work

Having recognized the importance of SDN controllers, the research community is often engaged in comparative studies of them. For example, Zhu et al. [7] made a qualitative comparison of 34 controllers acknowledging how software aging, controller placement and benchmarking tools impact the reliability of performance metrics. According to their research and considering latency and throughput, multi-threaded controllers, including centralized and distributed ones (e.g., Floodlight, OpenDaylight and ONOS) perform better than single-threaded (e.g., Ryu) and are more suitable for complex environments.

OpenDaylight and ONOS are the two most popular SDN controllers used for evaluation from researchers. According to their benchmarking, ONOS outperforms in terms of burst rate, throughput, round trip time (RTT) and bandwidth, GUI, clusters, link-up, switch-up, while OpenDaylight performs better in regards of topology discovery and stability [5], [8]. Li et al. [9] make a point that different controllers have different advantages and characteristics and thus selection needs to consider specific application scenarios. They employed the Mininet emulator and qperf tools with VMware Workstation as the underlying virtualization platform to show how Floodlight clearly outperforms Ryu based on bandwidth and latency.

Lunagariya et al. [10] have compared the performance of several controllers including Ryu, OpenDaylight, Floodlight and ONOS, while others[11], [12] focus exactly on these controllers. All four controllers support multithreading, virtualization and TLS and are available with REST APIs. By employing the Mininet emulator, iPerf and ping tools considering throughput, jitter and latency, on different network topologies (linear, tree and mesh) with varying number of connections, researchers conclude on different results about the controller with the best performance.

Bispo et al. [13], focus on 2 Python based controllers (POX & Ryu) and 2 Java based controllers (OpenDaylight & ONOS). They used Cbench in both latency and throughput mode. Their qualitative comparison was based on i) interface versatility, ii) GUI, iii) REST API, iv) programming language v) OpenFlow support, modularity, vi) multithreading support and vii) documentation and showed OpenDaylight and ONOS as the most feature rich controllers of the four. The authors experimental setup utilized Openstack Icehouse and the KVM hypervisor and their results favour OpenDaylight in most cases with ONOS following close by.

All the aforementioned studies do not give a clear picture of the performance of SDN controllers and are mainly limited to simplified topologies. In contrast, our work investigates the behavior of SDN controllers and tries to shed light on their performance in real-world topologies such as Fat-tree used in data centers, which is not entirely present in terms of analytical benchmarking in any other work.

III. State-of The Art of SDN Controllers

Although each SDN controller consists of two key components the core and the interfaces, however, there are many implementations that differ in the way they are deployed (e.g., programming language), the number and type of module and the capabilities. Among them ONOS, Ryu, Floodlight and OpenDaylight are the four of the most well-known controllers. All these controllers are open source. Before continuing with the comparative study, we briefly present them.

A. Ryu

Ryu [14] is a SDN controller fully written in python and supported by Nippon Telegraph and Telephone (NTT). Its design is based on components aiming to the improvement of network agility. Communication down to the data plane and up to management applications is supported by the southbound and the northbound REST APIs respectively. Using various protocols, such as Netconf and OpenFlow, Ryu directs the actions of the forwarding plane in line with the rules and policies of management applications [15]. The Ryu controller supports all versions of the OpenFlow protocol (1.0 to 1.5). The component-based approach facilitates the quick and easy development as it allows the modification of existing components or the addition of new ones. It is a centralized controller, available under the Apache 2.0 license model and supports multi-threading functionalities.

B. ONOS

ONOS [16] is a Java based SDN controller developed in 2012 under the Apache 2.0 license. It works in a distributed fashion and supports multi-threading. It services communications using REST and Neutron northbound and southbound APIs based on OpenFlow version 1.0 and 1.3. It also provides eastbound and westbound APIs for inter-controller traffic with the Raft protocol. It offers command line interface and graphical user interface both simplifying the interaction with users towards configuration or deployment of new services. ONOS comes with high modularity, extensibility and consistency focusing on scalability and high performance [17].

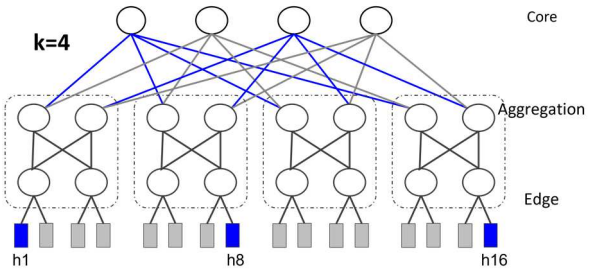

Fig. 2. A k-ary Fat-tree topology (k = 4)

C. OpenDaylight

OpenDaylight [18] is also a Java based controller, with distributed architecture and multi-threading support that appeared in 2013. OpenDaylight utilizes the REST, NETCONE, XMPP, and RESTCONF APIs to communicate with all available interfaces. It is distinguished from other controllers for the plethora of southbound interfaces that it supports. OpenFlow 1.0 and 1.3 assists switch-controller traffic while Akka or Raft supports inter-controller traffic [7]. OpenDaylight is available under the EPL 1.0 license and provides command line and graphical user interface. It utilizes OSGi bundles which can be deployed as Apache Karaf components. ODL provides various "features" that support several functionalities (e.g., DLUX supports graphical user-friendly topology representation).

D. Floodlight

Floodlight [19] is a Java - implemented, centralized controller first developed in 2014. It belongs to the centralized SDN controllers and offers multi-threading support. Communications with the other two planes is supported by REST, RPC, Java, and Quantum for the northbound API and OpenFlow 1.0 and 1.3 for the southbound API. It offers command line and web user interface and provides fair modularity and good consistency.

IV. The Comparative Framework

A. Methodology

In this work, the four principles of comparative evaluation have been adopted [20]: a) select the controllers for comparison, b) decide the level and scope of comparison c) define the criteria and metrics that we use as key performance indicators d) conduct performance analysis and analyze the findings.

B. The compared controllers

For this study, we have chosen four different SDN controllers: ONOS, Ryu, Floodlight and OpenDaylight. Our decision is driven by the popularity and the ranking of these controllers according to the research literature [6], [21], [22]. We evaluate the latest stable versions of SDN controllers except from OpenDaylight. In particular, we used ONOS version 2.2 [16], Ryu version 4.34 [14], Floodlight version v1.2 [18] and finally an older version of OpenDaylight, namely Carbon 0.6.4[18]. We chose the Carbon version as newer versions of OpenDaylight do not provide the L2 switching application necessary for MAC address learning.

C. The level and scope of comparison

We investigate the effectiveness of controllers in regard to TCP traffic in data centers. TCP traffic occupies 99.91% of the traffic in data centers [23]. It is consisted of long flows

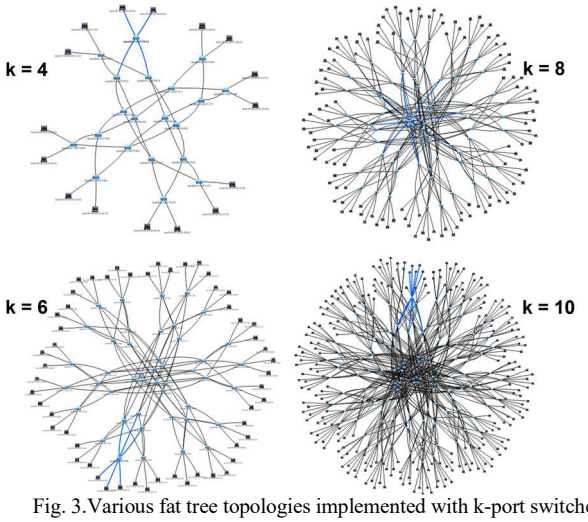

Fig. 3.Various fat tree topologies implemented with k-port switches (k = [4, 6, 8, 10]) as are represented by GUI of OpenDaylight. controller.

which are throughput sensitive and queries and small messages which are delay sensitive. Supposing the same network environment we examine the response of each controller for similar cases. All controllers utilize the OpenFlow protocol to establish the flow setup process to meet the needs for data forwarding. However, it is possibly the source of performance issues (bottleneck, latency) [24].

D. Evaluation framework

1) Topologies (Fat-tree topology)

Network topology defines the arrangement of network components and their interconnection (links) to each other. The most common topology in data centers is the Fat-tree topology.

The traditional data center topology is formed by 3 layers, namely core (the root), aggregation (the middle) and access (the lowest) layer. The access layer connects the end-devices hosts or servers to the network. The most common data center topology is the Fat-tree topology. It is used to interconnect many cheap commodity switches to establish communication in large-scale data networks. A crucial factor of Fat-tree topology deployment is the k parameter, which determines the number of supported ports of each switch. A k-ary Fat-tree topology is formed by k-port switches. Apart from the core layer switches, the others are grouped in k different subsets, named pods. Each pod also has k interconnected switches and an equal number for uplinks and downlinks. The Fat-tree topology is an economical and highly scalable solution for data centers [25]. The Fat-tree topology provides high availability through multiple (depending on the k factor) equal-cost paths from the aggregation layer to the access layer. This results in fast, deterministic convergence in the event of a link or node failure. Fig. 2 illustrates an example of Fat-tree topology.

2) Performance Metrics

For this comparative study, we define and capture the following performance metrics:

- *Latency*: Is the sum of the delays introduced by the controller, switches, and transmission links. We took latency measures using ping which measures the round trip time (RTT) between two communication end-point. RTT is defined as the time elapsed from the packet departure for a destination and back to the

sender. In the Fat-tree topology, the latency is slightly increased due to the lookup process in the switches routing tables with the size depended on the k parameter. Fluctuations in RTT indicate possible congestion on the communication network. We considered the RTT as the average of the *n* consequent estimations (RTT_i) taken by the ping tool [26].

$$RTT=(1/n)\ RTT_i \quad (1)$$

- *Throughput*: it is defined as the volume of transmitted data over a certain time slot between two nodes. In relevance to TCP traffic and using the client - server model the throughput is obtained by the following formula:

$$Throughput= Windowsize/RTT \quad (2)$$

where RTT is the minimum RTT value.

- Jitter: is defined as the time difference of arrival of consecutive packets at the end point of communication. Jitter is representative metric of RTT variability. Jitter affects the quality of services and the decisions of the controller on how to route flows [27]. Let T_i the arrival time of the packet *i*, then, the jitter for *n* consequent packet is calculated as [28]:

$$Jitter = \frac{\sum_1^n Ti}{n} \quad (3)$$

3) Use cases

Our study focusses on custom topologies and more specifically we use Fat-tree topologies for various degrees of branching (k factor). Table I summarizes the use cases for carrying out the experiments. Metrics are obtained for various paths of traffic flows between end-hosts. Here we present the following scenarios for communication:

- Scenario 1: two hosts located in the same pod
- Scenario 2: two hosts located in adjacent pods
- Scenario 3: two hosts located in furthest pods

TABLE I. THE NETORK TOPOLOGY SETTINGS FOR FAT-TREE

k	Fat-tree topology	
	Number of switches	*Number of hosts*
4	20	16
6	45	54
8	80	128
10	125	250

TABLE II. SELECTED HOSTS FOR EXPERIMENTS

k	Scenario 1		Scenario 2		Scenario 3	
	Server	*Client*	*Server*	*Client*	*Server*	*Client*
4	h1	h4	h1	h8	h1	H16
6	h1	h6	h1	h12	h1	H54
8	h1	h8	h1	h16	h1	H128
10	h1	h10	h1	h20	h1	H250

We apply all these scenarios for each topology using the client-server model for TCP communication. Table II details the test scenarios.

V. PERFORMANCE EVALUATION

A. The experimental environment

The experiments were conducted on a Desktop PC with the following technical specifications: AMD Ryzen 5 3600 6-Core Processor at 3.60 GHz, 16GB RAM, and 64 bit operating system.

The controllers' software was installed on an Ubuntu 20.04.4 virtual machine. Oracle VM VirtualBox was used for the virtual machine. The emulator tool Mininet [29] was used for the virtual network topology creation while for the measurements iPerf and ping tools are used. IPerf is a network tool that generates TCP or UDP traffic. Using the iPerf tool, the maximum bandwidth (throughput) that a controller can maintain is measured. IPerf is capable of measuring 10Gbps connections and higher, as it has the ability to perform multiple connections simultaneously.

Comparative evaluation of controllers' performance is carried out through deployment of *k* sized topologies where *k* is valued between 4 and 10. This is followed by benchmarking for TCP throughput, latency (or average delay) and jitter. The process is conducted by iterating through the steps below:

- The controller is initiated. The controller runs on port 6633 or 6653. In every instance only the evaluated controller is running.
- Using Mininet a Fat-tree topology is deployed. Mininet is remotely connected to the running controller using the controller's IP and port.
- The benchmarking tests are carried out and the metrics are obtained.

B. Latency Comparison

For the scenarios described above, we obtained the latency for each controller and compare the results using the ping network tool. Fig. 5 depicts the measurements collected. The results reveal an increase in delay for the Ryu, ODL and Floodlight controllers which is proportional to both the *k* parameter and the distance between the two nodes. The Ryu controller overperforms the others as it shows the lowest delay with Floodlight following. Surprisingly, ONOS' benchmarks produce high latency metrics for a k-ary topology where k equals to 6. Looking at the analytical metrics in these specific cases we observe that the increase in latency is due to Initial Ping Delay (IPD). IPD is the time consumed by the SDN controller to modify the flow table when the first packet of ping is sent.

C. Throughput Comparison

Throughput is evaluated using iPerf and conducting tests between two host (client and server) as mentioned in Table II. For example, in the first use case and scenario 1, on a 4-ary Fat-tree topology (k=4), hosts h1 and h4 exchange TCP messages. Fig. 5 shows the TCP throughput versus Fat-tree branching (k factor). More specifically, Fig. 5(a) presents the

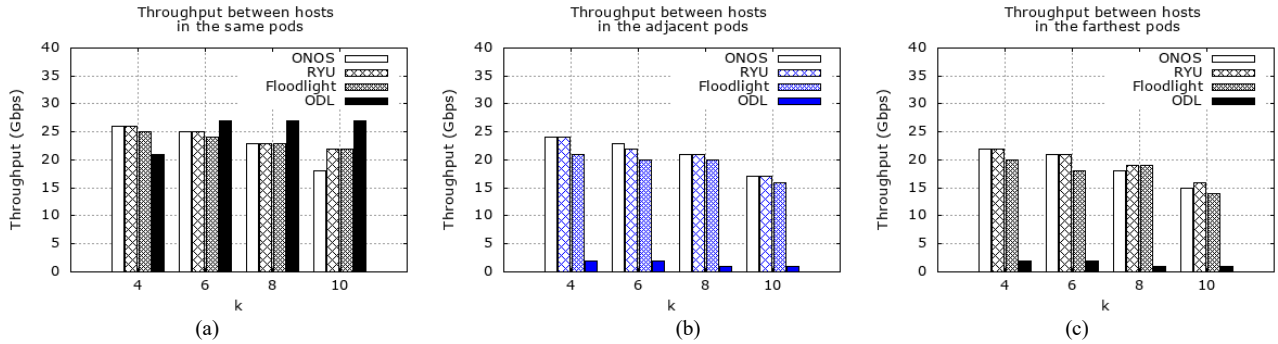

Fig. 4. The experimental results for topology throughput when the two hosts located at (a) same pod, (b) adjacent pods and (c) farthest pods

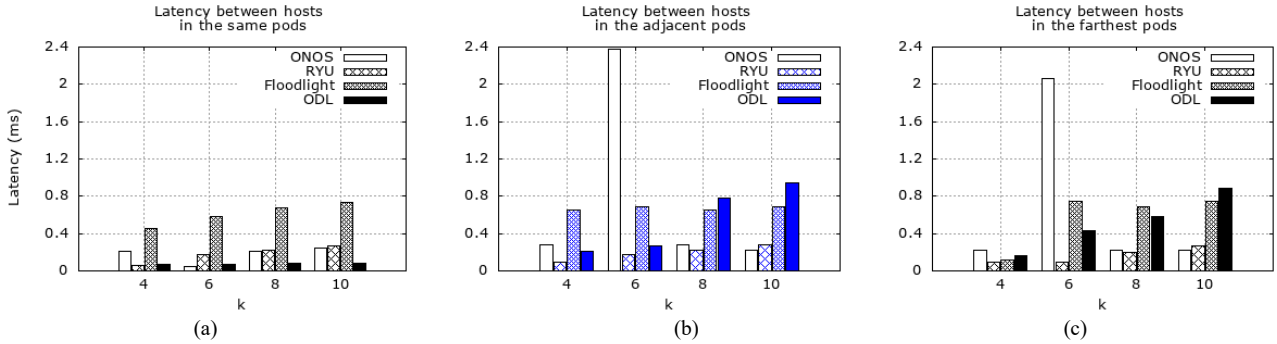

Fig. 5. The experimental results for topology latency when the two hosts located at (a) same pod, (b) adjacent pods and (c) farthest pods

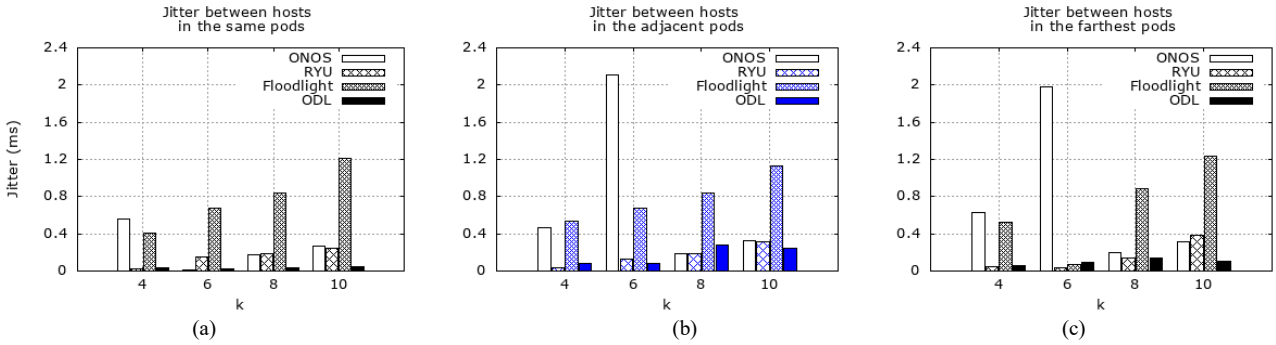

Fig. 6. The experimental results for topology jitter when the two hosts located at (a) same pod, (b) adjacent pods and (c) farthest pods

measurement throughput between nodes in the same pod for all controllers grouped by a factor of k. Likewise, Fig. 5(b) shows the throughput between adjacent pods, and Fig. 5(c) the throughput between farthest pods. Focusing on Fig. 5 (a), it can observed that the overall throughput performance of the controllers decreases slightly as the k factor increases. Fig. 5(b) and 5(c) demonstrate a throughput decrease as the distance between the two nodes increases. OpenDaylight behavior is noticeable due to its stability related to node distance and diminishing performance as the *k* factor increases.

D. Jitter Comparison

Jitter is also measured using the ping tool. Fig. 6 represents the time variation at average, as it is calculated by using the formula (3). All controllers display acceptable jitter values since it ranges up to 2ms and do not impact negatively the quality of services. The ODL and Ryu controllers have the lower jitter while Floodlight exhibits slightly higher values. The last place in the ranking is occupied by the ONOS. It has a high average value of 2,1ms if the two hosts are not in the same pod and $k = 6$ which could imply congestion for ONOS under specific types of networks.

E. Result Analysis

In this study, we compared the performance of ONOS, RYU, Floodlight and OpenDaylight in terms of Latency, Throughput, and Jitter. For this purpose, we created complex topologies, resembling real-world networks, as Fat-tree topologies which is used in data centers, using Mininet.

With the throughput tests, we found that Ryu in general outperforms the other controllers and OpenDaylight is unable to achieve as good a performance in large scale topologies as the other controllers. This was also observed in [20],[29],[31], by the authors.

At the same time for small deployments OpenDaylight exhibits optimal throughput, latency, and jitter. As OpenDaylight exhibits low jitter regardless of network size, its behavior indicates it could be better suited for quality of service oriented network applications such as VoIP.

ONOS's metrics on latency and jitter for k equal to 6 is an indication of how the topology can impact controller performance.

VI. Conclusion and Future Work

We have conducted a comparative study on four well-known controllers: ONOS, OpenDaylight, Floodlight and Ryu. Using the Mininet emulator and common network tools (iPerf and ping) as our means, we experimentally investigated the performance of these controllers in terms of throughput, average latency and jitter on size variations of the Fat-tree topology

We have observed that controller election clearly impacts network performance and care must be taken to ensure the optimal solution depending on the specifics of each network deployment.

Our future work is planned to carry out more in-depth investigation on the controllers and its behavior using more specific tools and metrics.

Acknowledgment

This research has been co-financed by the European Regional Development Fund of the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH – CREATE – INNOVATE, project name "Create a system of recommendations and augmented reality applications in a hotel" (project code:T1EDK-03745).

References

[1] H. Tong, X. Li, Z. Shi, and Y. Tian, "A Novel and Efficient Link Discovery Mechanism in SDN," 2020 IEEE 3rd Int. Conf. Electron. Commun. Eng. ICECE 2020, pp. 97–101, 2020, doi: 10.1109/ICECE51594.2020.9353035.

[2] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An intellectual history of programmable networks," Comput. Commun. Rev., vol. 44, no. 2, pp. 87–98, 2014, doi: 10.1145/2602204.2602219.

[3] W. Queiroz, M. A. M. Capretz, and M. Dantas, "An approach for SDN traffic monitoring based on big data techniques," J. Netw. Comput. Appl., vol. 131, no. January, pp. 28–39, 2019, doi: 10.1016/j.jnca.2019.01.016.

[4] N. McKeown et al., "OpenFlow," ACM SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, 2008, doi: 10.1145/1355734.1355746.

[5] S. Badotra and S. N. Panda, "Evaluation and comparison of OpenDaylight and open networking operating system in software-defined networking," Cluster Comput., vol. 23, no. 2, pp. 1281–1291, 2020, doi: 10.1007/s10586-019-02996-0.

[6] J. Ali and B. H. Roh, "A Novel Scheme for Controller Selection in Software-Defined Internet-of-Things (SD-IoT)," Sensors, vol. 22, no. 9, 2022, doi: 10.3390/s22093591.

[7] L. Zhu et al., "SDN Controllers: A Comprehensive Analysis and Performance Evaluation Study," ACM Comput. Surv., vol. 53, no. 6, 2021, doi: 10.1145/3421764.

[8] Y. Fan, L. Qing, and H. Qi, "Research and comparative analysis of performance test on SDN controller," 2016 1st IEEE Int. Conf. Comput. Commun. Internet, ICCCI 2016, pp. 207–210, 2016, doi: 10.1109/CCI.2016.7778909.

[9] Y. Li, X. Guo, X. Pang, B. Peng, X. Li, and P. Zhang, "Performance Analysis of Floodlight and Ryu SDN Controllers under Mininet Simulator," 2020 IEEE/CIC Int. Conf. Commun. China, ICCC Work. 2020, pp. 85–90, 2020, doi: 10.1109/ICCCWorkshops49972.2020.9209935.

[10] D. Lunagariya and B. Goswami, "A comparative performance analysis of stellar SDN controllers using emulators," Proc. 2021 1st Int. Conf. Adv. Electr. Comput. Commun. Sustain. Technol. ICAECT 2021, 2021, doi: 10.1109/ICAECT49130.2021.9392391.

[11] A. K. Arahunashi, S. Neethu, and H. V. Ravish Aradhya, "Performance Analysis of Various SDN Controllers in Mininet Emulator," 2019 4th IEEE Int. Conf. Recent Trends Electron. Information, Commun. Technol. RTEICT 2019 - Proc., pp. 752–756, 2019, doi: 10.1109/RTEICT46194.2019.9016693.

[12] L. Mamushiane, A. Lysko, and S. Dlamini, "A comparative evaluation of the performance of popular SDN controllers," IFIP Wirel. Days, vol. 2018-April, pp. 54–59, 2018, doi: 10.1109/WD.2018.8361694.

[13] P. Bispo, D. Corujo, and R. L. Aguiar, "A qualitative and quantitative assessment of SDN controllers," Proc. - 2017 Int. Young Eng. Forum, YEF-ECE 2017, no. January 2018, pp. 6–11, 2017, doi: 10.1109/YEF-ECE.2017.7935632.

[14] "Ryu SDN Framework." https://ryu-sdn.org/ (accessed Sep. 30, 2022).

[15] Y. Zhang and M. Chen, "Performance evaluation of Software-Defined Network (SDN) controllers using Dijkstra's algorithm," Wirel. Networks, 2022, doi: 10.1007/s11276-022-03044-3.

[16] "Open Network Operating System (ONOS) SDN Controller for SDN/NFV Solutions." https://opennetworking.org/onos/ (accessed Sep. 30, 2022).

[17] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A comprehensive survey of interface protocols for software defined networks," J. Netw. Comput. Appl., vol. 156, pp. 1–30, 2020, doi: 10.1016/j.jnca.2020.102563.

[18] "Home - OpenDaylight." https://www.OpenDaylight.org/ (accessed Sep. 30, 2022).

[19] "Floodlight Controller - Confluence." https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview (accessed Sep. 30, 2022).

[20] P. Vartiainen, "On the Principles of Comparative Evaluation," Evaluation, vol. 8, no. 3, pp. 359–371, 2002, doi: 10.1177/135638902401462484.

[21] M. M. Elmoslemany, A. S. T. Eldien, and M. M. Selim, "Performance Analysis in Software Defined Network Controllers," Proc. ICCES 2020 - 2020 15th Int. Conf. Comput. Eng. Syst., 2020, doi: 10.1109/ICCES51560.2020.9334577.

[22] A. E. S. F. Ahmed and H. A. Elsayed, "Performance Comparison of SDN Wireless Network Under Floodlight and POX Controllers," 13th Int. Conf. Electr. Eng. ICEENG 2022, pp. 91–95, 2022, doi: 10.1109/ICEENG49683.2022.9781874.

[23] M. Alizadeh et al., "Data Center TCP (DCTCP) Mohammad," Sigcomm, vol. 40, no. 4, p. 63, 2010, [Online]. Available: http://portal.acm.org/citation.cfm?doid=1851182.1851192.

[24] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," 2nd USENIX Work. Hot Top. Manag. Internet, Cloud, Enterp. Networks Serv. Hot-ICE 2012, 2012.

[25] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "A comparative analysis of data center network architectures," 2014 IEEE Int. Conf. Commun. ICC 2014, pp. 3106–3111, 2014, doi: 10.1109/ICC.2014.6883798.

[26] J. But, U. Keller, D. Kennedy, and G. Armitage, "Passive TCP stream estimation of RTT and Jitter parameters," Proc. - Conf. Local Comput. Networks, LCN, vol. 2005, pp. 433–440, 2005, doi: 10.1109/LCN.2005.101.

[27] J. K. Sojan and K. Haribabu, "Monitoring Jitter in Software Defined Networks," Lect. Notes Networks Syst., vol. 450 LNNS, pp. 635–645, 2022, doi: 10.1007/978-3-030-99587-4_54.

[28] A. Rodriguez and J. Qui, "A Comparative Evaluation of ODL and ONOS Controllers in Software-Defined Network Environments."

[29] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," Proc. 9th ACM Work. Hot Top. Networks, Hotnets-9, 2010, doi: 10.1145/1868447.1868466.

[30] A. Azzouni, R. Boutaba, N. T. M. Trang, and G. Pujolle, "SOFTDP: Secure and efficient OpenFlow topology discovery protocol," IEEE/IFIP Netw. Oper. Manag. Symp. Cogn. Manag. a Cyber World, NOMS 2018, pp. 1–7, 2018, doi: 10.1109/NOMS.2018.8406229.

[31] M. B. Dissanayake, A. L. V Kumari, and C. E. Board, "P Erformance Comparison of Onos and Odl Controllers in," vol. 2, no. 3, pp. 94–105, 2021.