FACTA UNIVERSITATIS
Series: **Automatic Control and Robotics** Vol. 21, Nº 1, 2022, pp. 25 - 35
https://doi.org/10.22190/FUACR211223003C

Regular Paper

INVESTIGATING THE IMPACT OF TREE-BASED NETWORK TOPOLOGY ON THE SDN CONTROLLER PERFORMANCE

UDC ((004.3/.4:0.034.2)+004.78)

Danijel Čabarkapa[1], Dejan Rančić[2], Petar Pavlović[1], Miodrag Milićević[1]

[1]Academy of Professional Studies Šabac,
Department of Medical and Business-Technological Studies, Republic of Serbia
[2]University of Niš, Faculty of Electronic Engineering,
Department of Computer Science, Republic of Serbia

Abstract. *Software Defined Networking (SDN) is an important technology that enables a new approach to how we develop and manage networks. SDN divides the data plane and control plane and promotes logical centralization of network control so that the controller can schedule the data in the network effectively through the OpenFlow protocol. The performance and capabilities of the controller itself are important. The impact of network topology type on controller performance can be very significant. In order to have better communication in SDN, it is essential to have an analysis of the performance of specific network topologies. In this paper, we simulate ONOS and RYU controllers and compare their different network parameters under the proposed complex custom Tree-based topology. A network topology has been designed using a Mininet emulator, and the code for topology is executed in Python. From the throughput, packet transmission rate, and latency analysis, the ONOS controller displayed better results than RYU, showing that it can respond to requests more efficiently under complex SDN topologies and traffic loads. On the contrary, the RYU controller provides better results for the less complex SDN networks.*

Key words: *Software defined networking, OpenFlow, software switching, Mininet, ONOS controller, RYU controller*

Received December 23, 2021 / Accepted April 05, 2022
Corresponding author: Danijel Čabarkapa
Academy of Professional Studies Šabac, Department of Medical and Business-Technological Studies, Hajduk Veljkova 10, 15000 Šabac, Republic of Serbia
E-mail: d.cabarkapa@elfak.rs

© 2022 by University of Niš, Serbia | Creative Commons License: CC BY-NC-ND

1. INTRODUCTION

To facilitate new networking evolution, a concept of programmable networks has been proposed. The fundamental idea has evolved into what today is called Software Defined Network (SDN). Compared to traditional networks, SDN decouples the control logic from network layer devices and centralizes it for efficient traffic forwarding and flow management. SDN is having the ability to separate the data and control functions of core devices and consolidates all the control in a single node called the network controller [1]. This centralized entity provides programmable control of the whole network and enables real-time control of all the underlying devices. The SDN controller is comprised of logically centralized "network intelligence" software and has a global view of the network. The controller architecture can be centralized or distributed.

Centralized SDN controllers are mostly used in small-scale SDN networks, whereas distributed controllers can span across multiple domains. A single-threaded SDN controller is more suitable for less complex SDN deployments. In contrast, multi-threaded controllers are suitable for commercial purposes such as 4G/5G, SDN-WAN, ISP, and optical networks.

The controller in a SDN is the core and critical component responsible for making decisions on managing traffic in the underlying network. The core functions of the controller are mainly related to network topology and traffic flow. In SDN networks, Ethernet switches are replaced by OpenFlow switches. Each OpenFlow switch dynamically maintains a flow table, which consists of flow rules that determine the handling of network packets [2]. In an SDN architecture, the infrastructure devices (switches and routers) have been designed to act like modules that process the incoming data packets to forward these packets towards their destinations. The forwarding of these packets is based upon the logic-based set of rules programmed into the SDN controller(s).

Although the fundamental function of an SDN controller is flow management, several different metrics can be used for its performance analysis. In this paper, we presented a performance evaluation in both throughput and latency perspectives for the current well-known OpenFlow controllers: RYU and ONOS. The controller benchmarking tool was implemented for the incremental number of switches connected to the controller under the simulation environment. We have compared RYU and ONOS controllers using the Mininet emulator, along with a detailed analysis of their performance in a custom Tree-based network topology named as Fat-Tree. Topology influence on the SDN network performance is done by analysis and evaluation of different network parameters such as throughput, packet transmission rate, and latency.

From the TCP throughput simulation, the ONOS controller displayed better results than RYU, showing that it can respond to requests more promptly under complex Fat-Tree topology traffic loads. Simulation outcomes indicate that in round-trip propagation delay between end nodes ONOS exhibited better results than the RYU controller. The ONOS controller was found to outperform RYU in the proposed Fat-Tree topology environment regarding the throughput and time between packets sent to the end hosts.

The rest of this paper is organized as follows. Section II presents the SDN and controller architecture and gives an overview of OpenFlow, RYU, and ONOS features. Section III presents the simulation environment, research methodology, and metrics. Section IV shows the results of the SDN controller analysis. The paper is then concluded in Section V.

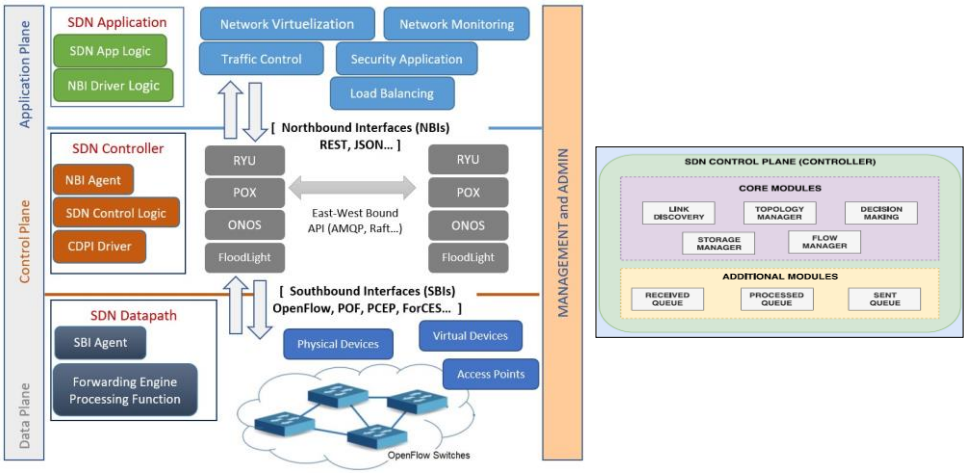

Fig. 1 Overview of a typical layered SDN architecture (left) and SDN controller architecture (right) [3]

2. Background and Related Work

A. SDN Network Architecture

The architecture of a SDN network can be divided into three planes: data plane, control plane, and application plane. The work in [3] discussed a three-layer SDN architecture model, as we can see in Fig. 1 (left). SDN separates the control and data plane of a traditional network device. The control plane is implemented through one or more logically centralized controllers. Control functionality is removed from network devices, that will become simple packet forwarding network nodes. The application plane interacts with the control layer to program the whole network and enforce different policies. The interaction among these layers is done through interfaces that work as communication protocols. SDN Application consists of one Application Logic module and one or more Northbound Interface (NBI) Drivers. The SDN is programmable through applications that interact with the underlying data plane devices. Higher-level logic can be implemented directly through these applications on top of controllers, which communicate through NBI Agents (REST, JSON, etc.) [4]. The SDN Datapath is a logical network device that comprises a Southbound Interface (SBI) Agent and a set of one or more traffic forwarding engines and traffic processing functions. The SBI defined as an interface between controller and Datapath, provides event notification, statistic reporting, capabilities advertisement, and high-level control of all forwarding operations. SBI interface is generally implemented using the OpenFlow protocol.

OpenFlow is the most widely accepted and deployed SBI standard for SDN and represents a protocol that is used for the communication between the controller and forwarding devices. OpenFlow modifies the SDN network in the sense that data plane nodes become simple devices that forward packets according to rules given by the controller. The main components of a SDN network are OpenFlow switches which can communicate with the controller via an OpenFlow protocol. An OpenFlow protocol can handle high-level routing, packet forwarding, and secure connection between the control plane and data plane. OpenFlow switch consists of one or more flow tables. Flow tables

determine data processing and forwarding with the help of flow entries. Each flow entry determines how data will be processed and forwarded in a network. The current version of OpenFlow is 1.5.1. as described in [5].

Fig. 1 (right) shows the architecture of the SDN controller. The core functions of the SDN controller are mainly related to topology and traffic flow, device management, and statistics tracking. All the controller functions are implemented via changeable modules, and the feature set of the controller may be adjusted to specific requirements of SDN networks. The topology itself is maintained by the topology manager. This provides the decision-making module to find optimal paths between nodes of the network. The controller tracks the topology by learning the existence of OpenFlow switches and other SDN devices and tracking the connectivity between them. Currently, there is a variety of open-source SDN controllers available for the community: POX, RYU, FloodLight, ONOS, ODL, OpenDayLight, etc. [6]. There is a lack of a standard for NBI, which has been implemented in several different forms. The ONOS and FloodLight controllers both use a Java and REST/RESTful API, while RYU and POX use Python API, etc. [7].

B. ONOS and RYU Controller

The Open Network Operating System (ONOS) is an open-source distributed SDN control platform, developed by the Open Networking Lab (ON Lab) [8]. The specific protocol feature of the system core makes ONOS available to be used for networks for various purposes such as company networks, campus networks, data center networks, etc. ONOS is specifically oriented to Internet Service Provider (ISP) networks, due to its distributed architecture and natively supports a distributed version of the controller, running on a cluster of servers. Each controller in the cluster is responsible for managing the OpenFlow switches under its domain. Each switch can connect to multiple ONOS controllers for reliability, but only one will be its master with full control over it in terms of read/write capabilities on the switch forwarding tables. The other controllers are denoted as slaves and one of them takes the control of a switch whenever the master controller fails. Generally, ONOS consists of NBI, SBI, and the Distributed Core. As a multi-threaded controller, ONOS is convenient for commercial purposes such as ISP and Data Center networks, SDN-WAN, and optical networks.

RYU controller is an open-source and component-based SDN framework implemented entirely in Python. RYU provides software components with well-defined APIs that make it simple to create control applications and SDN network management. RYU allows an event-driven programming paradigm in which the flow of the program is determined by events, and supports various protocols for managing network infrastructure, such as OpenFlow, Netconf (RFC 6241), OF-config, etc. [9]. The controller uses NBI APIs such as Restful, REST, REST/RPC user-defined API, etc. RYU provides a set of specific components such as OpenStack/Quantum virtualization, Firewall, OFREST, etc. for SDN applications.

C. Related works

Research in [10] gives a comprehensive investigation of open-source controllers RYU, POX, ONOS, and ODL. The authors were focused on parameters such as throughput and latency using Cbench tool. In [11], five controllers (RYU, POX, Trema, Floodlight, OpenDayLight) are compared, and the authors collect properties of each controller under specific evaluation: REST API support, modularity, virtualization, etc. In [12] authors

describe and evaluate two ONOS prototypes. The first version implemented logically centralized global network view, scale-out, and fault tolerance modules. The second version focused on improving ONOS performance in core network traffic engineering and scheduling. In [13] authors use a comparison of performance metrics such as delay, bandwidth, and received packets using network monitoring tools like IPERF and D-ITG to analyze the functionality of the POX controller. The results of this research were the recommendation of using a POX controller.

In [14], the authors have shown the performance comparison performed between the NOX, POX, Trema, and Floodlight in reactive and proactive modes. The results showed that the best performance is achieved when the controller is operating in the proactive mode because forwarding rules are installed on the switch. Authors in [15] presented a framework named HCprobe to compare seven controllers: RYU, Beacon, Maestro, MUL, FloodLight, NOX, and POX. To evaluate the efficiency of these controllers, the authors performed additional measurements like reliability, scalability, and security along with throughput and latency. The results show that FloodLight, Beacon, and MUL obtained minimum latency, while Beacon performed good results in the throughput test. Authors in [16] presented the crucial advantages and challenges of SDN security, flexibility, and performance against traditional TCP/IP networks.

3. Simulation Environment

The simulation hardware and software specifications are shown in Table 1. Performance analysis and network topology development were performed in an environment of a Windows 10 PC. The VirtualBox 6.1.22 hypervisor is used to instantiate two separate Virtual Machines (VM). Each VM is allocated 6 GB of RAM, and runs on Ubuntu 18.04, as the host Operating System (OS). Each VM separately contains Mininet with predefined ONOS and RYU controllers.

Table 1 Simulation hardware and software specifications

		PC	**VM**
Hardware	Processor	AMD Ryzen 5 3600, 3.6 GHz (6-core)	1 CPU, 6-core
	RAM	16 GB DDR4	6 GB DDR4
Software	OS	Windows 10 (64-bit), ver. 21H1	Ubuntu 18.04 (64-bit)
	VirtualBox	-	6.1.22
	Mininet	-	2.3.0d6
	RYU	-	4.3.2
	ONOS	-	2.5.0
	Python	-	3.8.5

The topology used in the simulation was based on Fat-Tree topology, as shown in Fig. 2. The Fat-Tree topology can be thought of as a reference data center topology. In this research paper, all OpenFlow switches are interconnected to each other, forming 3-level architecture: core, aggregation, and edge. This simulation deals with the results obtained

by Mininet emulating a custom Fat-Tree topology with 12 OpenFlow switches (*s1*-*s12*) and 32 hosts (*h1*-*h32*), where the 8 edge switches (*s5*-*s12*) have four emulated hosts each. The ONOS or RYU controller has the role of the SDN controller in the proposed topology. Host *h1* is denoted as Server, and *h2* is denoted as Client. After controller initialization, Mininet loads a Python script to instantiate the custom topology. Each OpenFlow switch is assigned a unique port for keeping track of network traffic.

Python 3.8 is used to write the network topology. Mininet Command Line Interface (CLI) is used to create the topology. Due to the topology complexity, we used a Mininet high-level API. We use *Topo* as the base class that provides the ability to create reusable and parametrized topology. We use methods *self.addSwitch()* and *self.addHost()* to import switches and hosts into topology and connect them. Further, method *self.addLink(node1, node2,**link_options)* is used for adding a bidirectional link that contains host and switch names and the number of options such as bandwidth or delay.

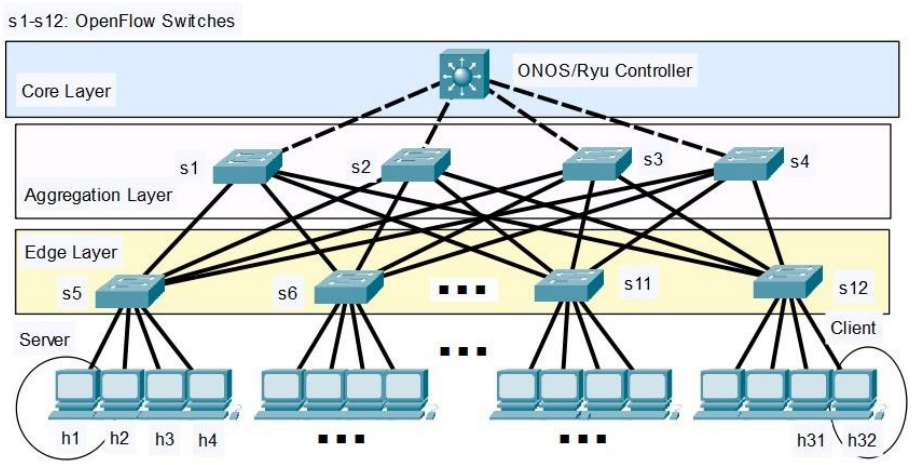

Fig. 2 The Fat-Tree SDN network topology

The Mininet CLI option --*mac* will automatically assign MAC addresses that match the host's names. All the nodes have been assigned a unique IP address from the 10.0.0.0/24 address range and a unique MAC address. To make switches connect to ONOS or RYU controller, we have used localhost 127.0.0.1 loopback IP address. The RYU controller uses port number 6633 to send messages to the OpenFlow switches. ONOS requires the 8181 port number to access the CLI and for GUI and REST API.

Spanning-Tree Protocol (STP) is a link management protocol that provides path redundancy while preventing undesirable loops in the network. Due to 64 links, 12 switches, and 32 hosts in our proposed topology “broadcast storms” are frequent. This undesirable network traffic circles endlessly in the network, due to the destination address in an unknown network. Both ONOS and RYU have a Python script for which the STP function is achieved using OpenFlow.

Latency Parameter: Latency is an important parameter to consider in the operation of a network, especially if it is used to transmit data from applications sensitive to delay or jitter. Round-Trip-Time (RTT) parameter identifies the time the packet spent on the up and downlinks, to and from the switches, and the time as a delay between end nodes. A performance comparison between RYU and ONOS controller in previously defined network topology is achieved by execution of ICMP query (Echo Request and Reply) connectivity test using the *ping* command. A *ping* test is performed between Client and Server hosts (*h1* and *h32* in topology). We make a latency analysis through the average RTT time for the first packet of the flow. We have chosen the average RTT of the first packet of a flow because the first packet going to the SDN network is first processed as a controller. Based on the first packet processing rule, the next packets are processed without connecting to the controller. Therefore, the first packet's RTT time is critical.

Throughput Parameter: Scalability in SDN can be achieved by improving the network throughput and creating a distributed network for data transmission. To evaluate throughput performance in the proposed SDN topology, we have considered TCP traffic between end hosts. A SDN throughput is generally defined as a rate for processing flow requests by the controller. For the performance analysis, we need to generate the TCP traffic between the Client and Server host and log the events using the IPERF networking tool. A typical IPERF output contains a timestamped report of the amount of data transferred through the network. With the IPERF tool, the TCP throughput and data loss are measured by sending and receiving TCP packets between pair of hosts (Client and Server host). Also, the time taken by TCP is calculated for data packets sent and received. For the OpenFlow packet capturing and analysis, we use Wireshark software [17-19].

4. Simulation Results

Considering SDN controller throughput, single-threaded RYU and multi-threaded ONOS show different results in the proposed Fat-Tree topology, where network traffic is significantly intensive. At first, we concluded that TCP throughput depends on the capabilities of the controller itself. The graph in Fig. 3 shows the results obtained by performing transmission between the farthest TCP Client *h1* and TCP Server *h32* in the proposed SDN topology. To measure ONOS and RYU controller throughput, the IPERF test has executed in 75 sec. on the Client, and data have been collected every 1 sec. on the Server host. TCP packets are sent from *h1* to *h32* with a default 1Mb/s sending rate. Let's analyze the observed results. It is calculated the average throughput stays at 18.6 Gbps for RYU, and 29.1 Gbps for the ONOS controller. According to the observation, the average throughput in RYU is 36.08% less than the ONOS controller. The graph also shows that the TCP throughput variations moderately fluctuate within the duration of the simulation. For the ONOS controller, the throughput variations are scientifically uniform to the RYU case. There are a few instances of excessive variations in the throughput for the RYU. Dropping instances were observed frequently between 41-51 sec. of simulation time, leading to degraded performance of the simulation run. This large drop now occurs again at 67 sec. of simulation, and the value of throughput was only 11.3 Gbps.

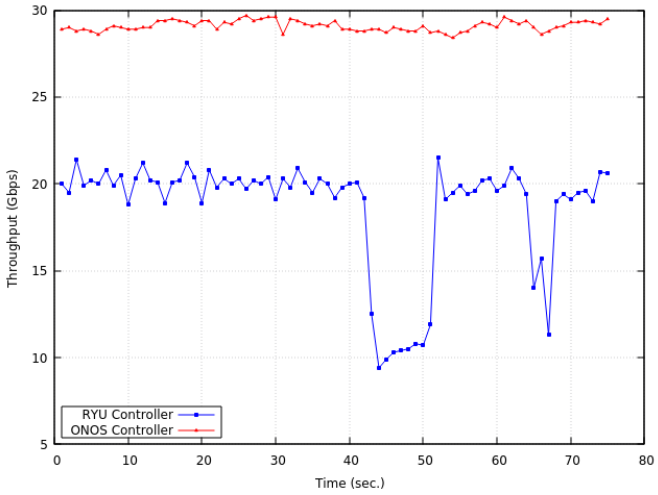

Fig. 3 TCP throughput between the farthest hosts (*h1* and *h32*) for the proposed topology with RYU or ONOS controller

This TCP throughput behavior can be caused by two phenomena. At first, this result shows that the RYU controller has a packet broadcast storm problem when controlling a complex proposed network with loops and a large number of OpenFlow switches and hosts. By default, the STP Python script is not built into the RYU controller. STP function for RYU is achieved using OpenFlow, and we used the *ryu.app.simple_switch_stp_13.py* script to enable STP. On the other hand, it appears that RYU requires more hardware resources (CPU utilization and memory) than ONOS. Here, the ONOS controller exhibits a better throughput performance. This is likely because of its inherent support for very large-scale networks. We decided to do the same simulation three more times to confirm whether this Ryu throughput behavior is accidental or not, and Mininet always generates similar results.

We use Wireshark to represent network traffic between the RYU/ONOS controller and the SDN nodes. Fig. 4 provides the results generated based on packets captured on the proposed Fat-Tree topology. As the controller and switch share the same VM guest the control channel is via the loopback interface, so monitoring the *loopback lo0* interface will give access to these packets. In this simulation, the messaging is using OpenFlow 1.3 so using the filter *openflow_v4* will show the communications between the hosts *h1* and *h32*. We have created a graph of the real-time captured network OpenFlow packets (*ofpt_packet_in*, *ofpt_packet_out*, *ofpt_stats_reply*). From the statistical analysis results, the continuous polling of data causes controller overhead. Fig. 4. shows the I/O graph for the proposed topology with ONOS (above graph), and RYU controller (bottom graph). Y-axis defines the number of transmitted packets, whereas X-axis denotes the time of simulation in seconds. As we can see from the graphs, the ONOS controller enables more network traffic and faster processing of packets. By making use of the I/O graph statistic evaluation, it was found that ONOS has a transmission rate that exceeds 1000 packets/sec, while RYU has a transmission rate that slightly exceeds only 100 packets/sec. This is because having a large number of OpenFlow switches causes conflict at the RYU controller data layer which demands high processing power and a worst packet transmission rate than ONOS. Furthermore, ONOS shows significantly better results in packet processing operations than Ryu, and the main reason is a multithread feature.

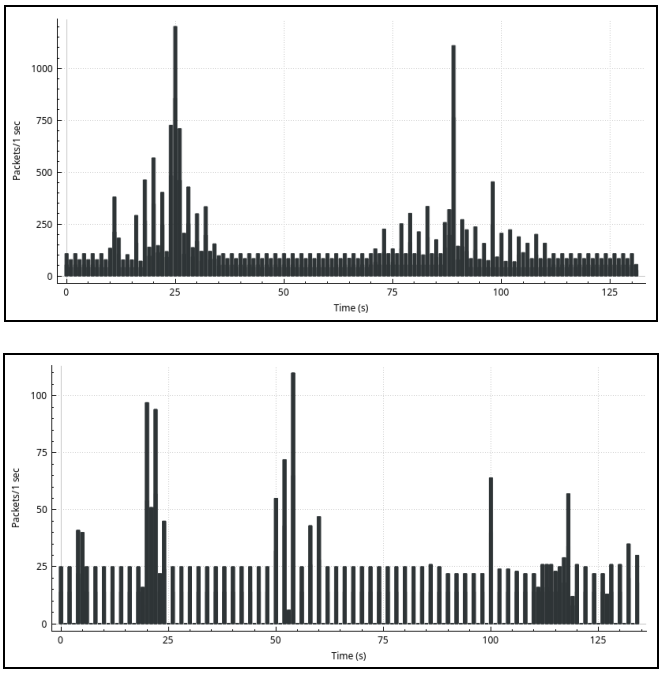

Fig. 4 I/O graph of OpenFlow packets transmitted per second for the proposed topology with ONOS (above) and RYU controller (below)

In the second part of the simulation, we observe the latency against ICMP packet size, from 100B to 1000B. In each performance test, we send 8 ping packets with the following sizes: 100, 500, and 1000 Bytes. We send the *ping* command from Server *h1* to Client *h32*, and the test is performed between the farthest nodes in the proposed topology. These *ping* commands are cycled 10 times for each test. Then we calculated the average values of measured RTTs of the first packet of the flow per test. These average values of the measured RTTs for both ONOS and RYU controllers are shown graphically in Fig. 5. It is visible that the propagation delay between nodes in the proposed topology is very different. From Fig. 5, the average RTT is the minimum for the ONOS controller and ICMP=100B packet size (RTT=7.89 ms). Therewith, the highest RTT value (RTT=15.3 ms) was measured for a RYU topology and ICMP=1000B.

From the obtained results, it is clear that a RYU controller is taking more time for transmission of the packet to its destination. The fact is that the RYU controller introduces larger latency in the network. In the RYU controller, the initial process of establishing network flow consumes time that introduces latency in the network. When the first packet sent by *h1* arrives at the OpenFlow switch, this switch does not know how to route it, encapsulate it, and forwards all the contents of the incoming packet to the controller, being responsible for managing the installation of the flow tables in each switch. From the above results, it can be concluded that longer ICMP packets require a longer processing time for the RYU controller, which affects the overall SDN network performance.

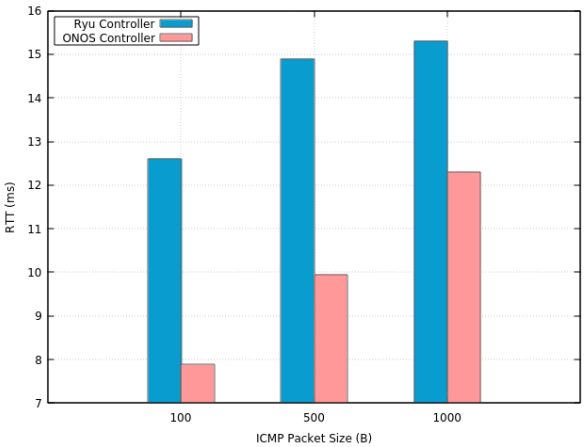

Fig. 5 Average RTT time for the first packet of the flow in proposed topology with RYU and ONOS controller

5. CONCLUSIONS AND FUTURE WORKS

RYU and ONOS are the two most powerful and widely used SDN controllers. There are many researchers currently working on the evaluation and comparison of these controllers. Both of them have their advantages and disadvantages. This paper can help researchers to choose between the RYU and ONOS in different use cases, especially for data centers and complex Tree-Based network environments with a large number of SDN controllers, switches, and links. The results of the simulation proved that ONOS performs better than RYU based on selected parameters.

In total, the ONOS controller was found to outperform RYU in the proposed Fat-Tree topology environment, especially regarding the throughput, time between packets sent to the end hosts, and response received at the OpenFlow switch. More sophisticated embedded ONOS algorithms, distributed architecture, and proactive installation of rules on the whole path that the packet will take, certainly lead to an overwhelming behavior of the ONOS controller in our proposed topology.

This research work opens opportunities for many other research directions. In the future, we plan to keep extending this study with the ONOS cluster multi-controller network environment and with some other SBI APIs. Furthermore, there were different sets of experiments that are left for future research. Some of the variations in experiments that can be conducted in the future to expand the scope of the investigations may include varying the size and/or numbers of the files being communicated. Moreover, it would also be interesting to investigate the results with different sizes of data packets and multi-controller topology in the experiments.

REFERENCES

[1] V. Nguyen, A. Brunstrom, K. Grinnemo, and J. Taheri, "SDN/NFV- Based Mobile Packet Core Network Architectures: A Survey," IEEE Communications Surveys Tutorials, vol. 19, no. 3, pp. 1567–1602, 2017. doi: 10.1109/COMST.2017.2690823

[2] N. Gude, T. Koponen, J. Pettit, "Nox: Towards an Operating System for Networks," ACM SIGCOMM Computer Communication Review, vol. 38, no. 3, p. 105, 2008.

[3] L. Zhu, Md M. Karim, K. Sharif, Fan Li, X. Du, M. Guizani, "SDN Controllers: Benchmarking & Performance Evaluation", ACM Computing Surveys, Vol. 53, Issue 6, Article No. 133, pp. 1–40, 2020, https://doi.org/10.1145/3421764

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford: "OpenFlow: enabling innovation in campus networks", ACM SIGCOMM Comp. Communication Review, vol. 38, no. 2, pp. 69–74, 2008. doi: 10.1145/1355734.1355746

[5] ONF Foundation: "OpenFlow Switch Specification", Version 1.5.1 (Protocol version 0x06), 2015, https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[6] T. Zhang, F. Hu: "Controller architecture and performance in software-defined networks", in Network Innovation through OpenFlow and SDN, CRC Press, 1st edition, 2014, doi: https://doi.org/10.1201/b16521

[7] W. Zhou, Li Li, Min Luo, Wu Chou: "REST API Design Patterns for SDN Northbound API", 28th International Conference on Advanced Information Networking and Applications Workshops, 2014, doi:10.1109/WAINA.2014.153

[8] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O. Connor, P. Radoslavov, W. Snow: "ONOS: towards an open, distributed SDN OS", in Proceedings of the third workshop on Hot topics in software defined networking, pp. 1-6, 2014, https://doi.org/10.1145/2620728.2620744

[9] Md. T. Islam, N. Islam, Md. Al Refat: "Node to Node Performance Evaluation through RYU SDN Controller", Wireless Personal Comm., issue 1/2020, pp. 550-570, 2020, doi: 10.1007/s11277-020-07060-4

[10] P. Bispo, D. Corujo and R. L. Aguiar: "A Qualitative and Quantitative assessment of SDN Controllers", International Young Engineers Forum (YEF-ECE), pp. 6-11, 2017, doi: 10.1109/YEF-ECE.2017.7935632

[11] R. Khondoker, A. Zaalouk, R. Marx, K. Bayarou: "Feature-based comparison and selection of Software Defined Networking (SDN) controllers", WCCAIS, 2014, doi: 10.1109/WCCAIS.2014.6916572

[12] P. Berde, M. Gerola, J. Hart, Yuta Higuchi, M. Kobayashi, T. Koide, Bob Lantz, B. O'Connor, P. Radoslavov, "ONOS: Towards an Open, Distributed SDN OS", HotSDN '14: Proceedings of the third workshop on Hot topics in software defined networking, pp.1–6, 2014, https://doi.org/10.1145/2620728.2620744

[13] H. M. Noman, M. N. Jasim: "POX Controller and Open Flow Performance Evaluation in Software Defined Networks (SDN) Using Mininet Emulator," 3rd International Conference on Sustainable Engineering Techniques (ICSET 2020), vol. 881, 2020, doi:10.1088/1757-899X/881/1/012102

[14] M. P. Fernandez: "Comparing openflow controller paradigms scalability: reactive and proactive", IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), 2013, doi: 10.1109/AINA.2013.113

[15] A. Shalimov, D. Zuikov, D. Zimarina, V. Pahskov, R. Smeliansky: "Advanced Study of SDN/OpenFlow Controllers," Proceedings of the Central Eastern European Software Engineering Conference CEE-SECR '13, no. 1, pp. 1-6, 2013, doi:10.1145/2556610.2556621

[16] S. H. Haji1, S. R. M. Zeebaree, R. H. Saeed, S. Y. Ameen et. all: "Comparison of Software Defined Networking with Traditional Networking," Asian Journal of Research in Computer Science, 9(2), 1-18. https://doi.org/10.9734/ajrcos/2021/v9i230216

[17] Wireshark User's Guide, Version 3.5.0, https://www.wireshark.org/docs/wsug_html_chunked

[18] R. Barrett, A. Facey: "Dynamic traffic diversion in SDN: test bed vs Mininet," in International Conference on Computing, Networking and Communications (ICNC): Network Algorithms and Performance Evaluation (2017). https://doi.org/10.1109/iccnc.2017.7876121

[19] S. H. Yeganeh, A. Tootoonchian and Y. Ganjali, "On scalability of software-defined networking," in IEEE Communications Magazine, vol. 51, no. 2, pp. 136-141, 2013, doi: 10.1109/MCOM.2013.6461198.