

Performance evaluation of software-defined networking controllers in wired and wireless networks

Salar Jamal Rashid^{1,2}, Ahmed Maamoon Alkababji², Abdul Sattar Mohammed Khidhir³

¹Department of Technical Training, Computer Center, Northern Technical University, Mosul, Iraq

²Department of Computer Engineering, College of Engineering, Mosul University, Mosul, Iraq

³Department of Computer Systems, Mosul Technical Institute, Northern Technical University, Mosul, Iraq

Article Info

Article history:

Received Feb 28, 2022

Revised Oct 27, 2022

Accepted Nov 06, 2022

Keywords:

Bitrate

Delay

Jitter

Mininet-WiFi

POX

Ryu

SDN

ABSTRACT

Traditional networking solutions are unable to meet modern computing needs due to the expanding popularity of the internet, which requires increased agility and flexibility. To meet these objectives, software-defined networking (SDN) arises. A controller is a major element that will determine if SDN succeeds or fails. Various current SDN controllers in many sectors must be evaluated and compared. The performance of two well-known SDN controllers, POX and Ryu, is evaluated in this research. We used the Mininet-WiFi emulator to implement our work and the distributed internet traffic generator (D-ITG) to assess the aforementioned controllers using delay, jitter, packet loss, and throughput metrics. What is new in our research is the study of network performance in two different types of transmission media: wired and wireless. The speed of the wired medium was chosen to be fast ethernet, which was not previously studied. In addition, the size of the packet was varied among 128, 256, 512, and 1,024 bytes. The comparison was performed on three topologies (single, linear, and tree). The experimental results showed that Ryu offers significantly lower latency, jitter, and packet loss than POX in most scenarios. Also, the Ryu controller has higher throughput than POX, especially on wireless networks.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Salar Jamal Rashid

Department of Technical Training, Computer Center, Northern Technical University

Mosul, Iraq

Email: salar.jamal@ntu.edu.iq

1. INTRODUCTION

It's becoming more common for data-forwarding devices, such as routers and switches, to be full of control needs and rules, which has caused a lot of people to worry about the current hardware-based network infrastructure. This problem derives from the reality that traditional networks have become not only excessively complicated to install and run properly, but also resistive to new service revolution. The control planes, which serve like the brains of networks, and the data planes, which handle the forwarding of data, are both tightly located within these proprietary and heterogeneous forwarding devices in a typical network [1], [2].

The control plane is in charge of configuring the node and programming the paths that will be used for flowing data. Once these pathways have been defined, they are transported down to the data plane, where this control information is used to decide data plane forwarding at the hardware level. Because there is no control plane abstraction of the entire network in the typical network system, a distributed technique of network management is used to manage the network. As a result, networks have gotten more complex and difficult to manage and configure when something goes wrong. The software-defined networking (SDN) idea has been presented as a solution to the difficulties [3]-[5].

SDN is a novel network paradigm that allows existing networks' control and data plane functions to be separated. The SDN architecture allows for centrally controlled capabilities as well as a global view of the entire underlying physical network, making network management much easier. This centralized entity provides programmable control of the whole network and enables real-time control of all the underlying devices [6], [7]. Figure 1 depicts the differences in functionality planes between a legacy and an SDN network. SDN keeps the data plane logic within the network elements, while the control layer handles flow control and maintains a global perspective of the network via a centralized controller [8].

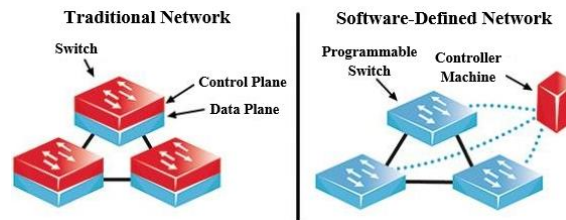


Figure 1. Comparison between SDN and traditional network [9]

The controller is at the heart of the SDN network, providing instructions to switches and other physical layer network devices. This particular controller will be responsible for offering forwarding decision-making and programmable interfaces, enabling user-written applications to manage the role of network devices based on a set of high-level rules. These rules are load balancing, switching, firewalling, routing, and network address translation [10]-[12].

Because the controller serves as the brain in the SDN architecture, the performance of the SDN is largely calculated by the controller's performance. Nowadays, there are a plethora of controllers to choose from, both free and commercial. As a result, a comprehensive evaluation methodology is required to select the optimum controller for each scenario based on quality of service (QoS) requirements, routing protocols, workload, and topology, all of which have a significant impact on the controller's performance [13]. There are some well-known criteria that can be used to measure the performance of the SDN controller, like delay, jitter, packet loss, and throughput. In this paper, the last aforementioned performance criteria of POX and Ryu controllers will be compared in different network topologies and workloads. The SDN controller connects with the data plane switches via open, standardized interfaces referred to as southbound interface (SBI) protocols such as OpenFlow.

OpenFlow was introduced by open networking foundation (ONF) in 2008 as the first protocol to separate the data and control planes [14]. OpenFlow is the most commonly used standardized SBI application programming interface (API) and follows the basic SDN principle of separation between the data and control planes. It specifies how the controller should adapt the network and communicate with data plane devices [15]. Traditional switches are vendor-specific making it less flexible in managing and programming so it was being replaced by OpenFlow switches. Each OpenFlow switch keeps a flow table, which is made up of flow rules that define how network packets are handled [16].

2. LITERATURE REVIEW

The positive impact of SDN on changing traditional network principles has resulted in a number of works, Islam *et al.* [17] evaluated the performance of different controllers in wireless networks. They compared between POX, Ryu, open network operating system (ONOS) and floodlight controllers using Mininet-WiFi emulator. The evaluation was done on a network of three hosts and three access points connected in a linear topology. The authors concluded that floodlight performs the best among the four SDN controllers in terms of jitter and delay. They also showed that Ryu and ONOS displayed the worst performance in terms of jitter and delay respectively, while the Pythonic network operating system (POX) exhibits an average performance in all terms.

Numan *et al.* [18] calculated the latency and jitter of SDN against traditional IoT communication networks. They showed that the average SDN jitter and latency per packet are three times smaller, which translate into better efficiency under different traffic conditions. The results showed that SDN improves network efficiency by reducing network overhead created from frequent communication attempts between the control and data planes each time a packet is received. It is also recommended that the distributed control plane architecture be used, although this also has its specific challenges, including how controllers can be positioned to ensure the best results.

Bhardwaj and Panda [19], the performance of the Ryu controller in a wired network was analyzed. The authors used Ping and iPerf tools to calculate the roundtrip time and throughput of a single topology made of four hosts. Their proposed work evaluated various node-to-node performances under transmission control protocol (TCP) and user datagram protocol (UDP) protocols. They came to the conclusion that the RYU SDN controller is one of the most powerful controller for traffic engineering.

Kumar and Sood [20], compared the characteristics of traditional networks and SDN. They analyzed the performance of POX controller in four topologies (reserved, single, linear and tree) based on delay, throughput, and packet loss. They used Wireshark for measuring performance metrics. They concluded that the linear topology is best in term of throughput and delay while the tree topology is the worst.

On the other hand, Babbar and Rani [21] evaluated the performance of Ryu in tree topology of two depth and three fanout (i.e. 9 hosts and 4 switches). They summarized the performance of Ryu using different criteria such as delay, bandwidth, jitter, and packet loss. They used different measurement tools, including Ping, iPerf, and Wireshark. They also advised that the work can be further extended to different network topologies like a single, linear, and ring using the different SDN controllers.

Mamushiane and Shozi [22] provided a comparison of ONOS and OpenDaylight (ODL) controllers in different network topologies. They analyzed the performance of these controllers in single, linear, and tree topologies. The comparison was made based on the distributed internet traffic generator (D-ITG) tool. They showed that ODL was drastically poor while the ONOS controller displayed the best performance for all metrics and scenarios.

Lastly, Salih [23] evaluated the performance of the Ryu controller in a wired network. The author used Cbench and Wireshark tools to calculate the latency and throughput of a linear topology with variable number of switches (2, 4, 8, 16, 32, and 64). He also advised that the work can be extended to measure different performance metrics like a jitter, packet drop and round-trip time. Our research will compare the performance of Ryu and POX controllers in different topologies with varying workloads in both wired and wireless networks.

3. THEORETICAL BACKGROUND

3.1. Software-defined networking (SDN)

The SDN architecture has a number of advantages over traditional network architectures, including traffic engineering policy change, programmability, a centralized view of the network, and high efficiency [24]. At present, SDN is viewed as a tool for delivering a dynamic and scalable architecture, which is the method to significantly improve future networks. With the help of SDN, network orchestration, and network virtualization, network architects will be able to create a really innovative and smarter network solution that is more efficient and cost effective [25].

Due to its specific architecture, SDN is not dependent on any proprietary software or hardware. It provides centralized control, programmability of the network, less capital cost, less operational cost, less agility, higher flexibility, easy customization and neutrality. Due to its popularity and scope in the field of networking, it is being adopted by academia and giant IT organizations [27]. Also, network modification is less error-prone and easier to carry out. The process of designing network servers, services, and applications is simplified, and high-level rules may be maintained by dynamically reacting to changes in the network state [15]. The architecture of the SDN network can be divided into three layers: the application layer, control layer, and infrastructure (data) layer. The relationships between SDN modules can be seen in Figure 2, which shows a basic overview of a typical SDN architecture.

- a) Application layer: this layer contains a range of programs that belong to various users. It includes network features and applications that are routinely used by enterprises, such as load balancing, firewalls, and security systems [28].
- b) Control layer: the main component of the SDN design is a logically centralized controller, which is present at this layer. This layer's job is to regulate the general behavior of network devices. It has comprehensive knowledge about all of the network's devices and can issue commands to all of them [29]. The control layer gets instructions from the application layer through Northbound APIs and transmits them to the data layer. It also gathers information's (issues, host tracks, and statistics) from the infrastructure layer and sends them back to the application layer [30].
- c) Data layer: it consists of all the networking devices that are concerned with forwarding the packets on the network. No device in this layer consists of any decision-making logic, but it just performs the actions on the packets coming to it according to the instructions given by the control layer. Communication between the control layer and data plane is carried out by the southbound APIs. For communication, OpenFlow is used [29], [31].

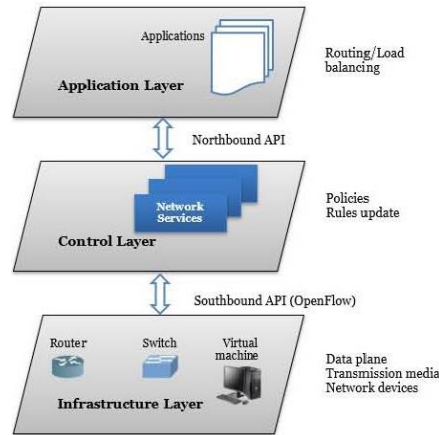


Figure 2. Software-defined network architecture [26]

3.2. SDN controllers

Several open-source SDN controllers are being used to deploy network designs, including Ryu, POX, floodlight, ONOS, ODL, and OpenDaylight [32]. The architecture of the SDN controller is shown in Figure 3. The diagram displays the modules that offer the controller’s main functionality, both northbound interface (NBI) and SBI, as well as a few applications that may use the controller. To connect with SDN devices, the southbound API is utilized. In the case of a commercial Open SDN controller, this API is OpenFlow, or an alternative in other SDN systems [33]. The controller’s primary duties are topology and device tracking and discovery, statistics, device administration, and flow management. These are all implemented by a set of modules internal to the controller [8].

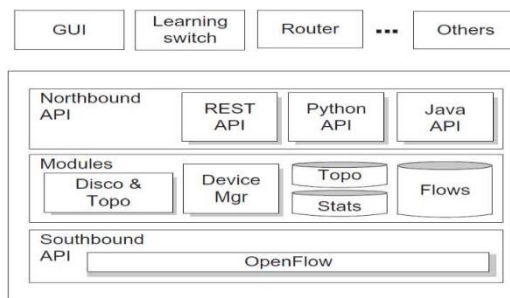


Figure 3. SDN controller architecture [34]

The importance of an SDN controller’s role deserves a short description of how it performs their primary tasks, packet forwarding and routing. In SDN environment, when node *X* starts communicating with node *Y*, the first switch receives the first flow from host *X* and sends it to its controller if the current flow table contains no information about host *Y*. The controller then completes the packet-in based on its applications and services, encapsulating it to create a packet-out. This, along with all other switches between the destination and the source, will be sent back to the switch. As a result, each switch flow table will be filled with the information needed to make the best routing decision [35]. Many SDN controllers have been developed, and their performance in various environments needs to be compared. Table 1 lists some of the most well-known SDN controllers, as well as the platforms that they support and the programming languages that were used to create them [6].

Table 1. SDN controllers comparison

Controller name	Programing language	Platform support	Developer
POX	Python	Linux, MAC and Windows	Nicira network
Ryu	Python	Linux	Nippon telegraph and telephone (NTT)
Floodlight	Java	Linux, MAC and Windows	BigSwitch network
OpenDaylight	Java	Linux	Cisco and OpenDaylight
ONOS	Java	Linux	ONF

3.2.1. POX

POX is a Python-based open-source controller that supports OpenFlow for developing SDN applications. A POX controller can also convert OpenFlow devices into switches, firewalls, load balancers, and other network devices. In the presence of the OpenFlow protocol, the POX controller can have straight access and manipulation capabilities to the forwarding devices. It is fast and very easy which makes it suitable for research purposes, demonstrations or experimentation. POX is based on a model that recognizes all SDN network activities and devices as separate components that can be separated and used at any time and in any location. It is responsible for achieving any type of communication between applications and SDN devices [8].

3.2.2. Ryu

The Ryu controller is an open-source and component-based SDN framework implemented entirely in Python. RYU term comes from a Japanese word meaning “flow,” which is a Japanese dragon, one of the water gods. it uses the OpenFlow protocol to associate with the switches to modify how the network will manage traffic flows and allows an event-driven programming model in which the flow of the program is established by events. The module called `ryu.controller.ofp_event` exports event classes that describe receptions of messages from connected switches. Ryu provides software components with well-defined APIs that make it simple to create control applications and SDN network management. It can also view the designed network in the GUI. Ryu provides a group of components such as OpenFlow representational API (OFREST), Firewall, OpenStack, and Quantum useful for SDN applications. The objective of these applications is to gathered network intelligence by using a controller, performed analytics by running algorithms, and then orchestrated the new rules by using the controller. In addition, Ryu supports various protocols for managing network infrastructure, such as OF-config, Netconf (RFC 6241), and OpenFlow. The Ryu is fully compatible with all OpenFlow versions (1.0 to 1.5) [36].

3.3. OpenFlow

OpenFlow is the most widely accepted and deployed SBI standard for SDN, and it is a protocol that is used to install data processing rules between the controller and forwarding devices. The SDN network is changed by OpenFlow in the sense that data plane elements are reduced to simple devices that forward packets according to the controller’s rules. OpenFlow switches are the most important components of a controller-based OpenFlow network. Each OpenFlow switch keeps a flow table, which is made up of flow rules (entries) that determine how packets are handled. Pattern fields (which match bits in the packet header), a list of actions (forward, flood, drop, modify, or send the packet to the controller), a set of counters (for tracking packets), and a priority field are all included in flow entries. Furthermore, as long as the OpenFlow enabled switch communicates with an OpenFlow controller, company vendors have a variety of options for implementing a data plane [8].

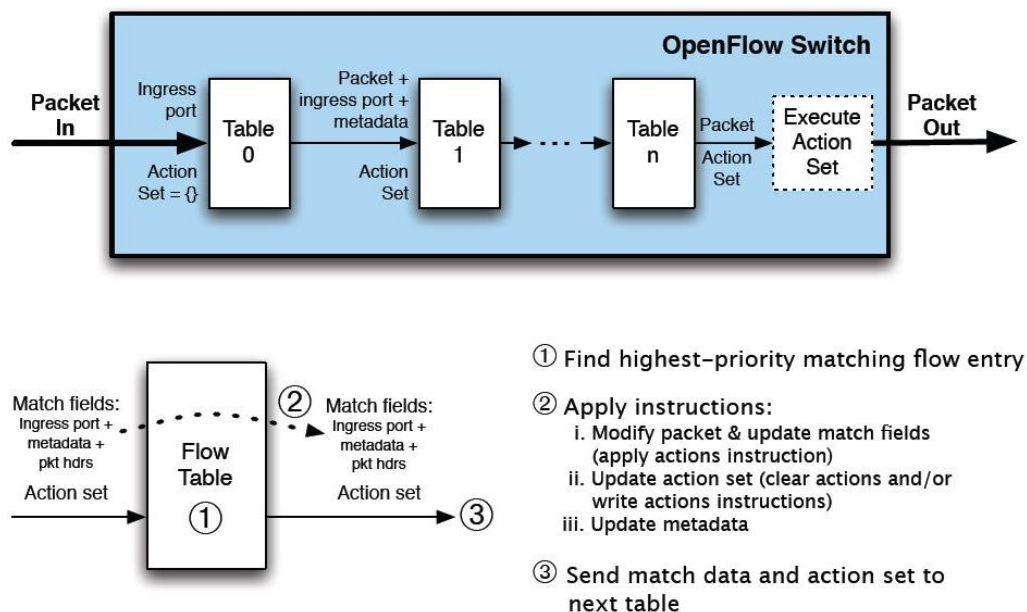


Figure 4. Working of SDN [28]

When the first packet of a flow is received by an SDN node (implemented using the OpenFlow protocol), the node queries the controller for the flow's forward path. The controller then updates the node firmware with the necessary rules. These rules, as shown in Figure 4, provide details about the activities required by the packet. The controller's job is to set up the necessary rules for each node in the forwarding path. An originating node that does not have a path installed related to the corresponding packet may send a route request message [28].

The structure of a flow entry in an OpenFlow switch is shown in Figure 5. When a new flow entry is received at a switch, the search begins. If no match is found, the switch sends a message to the controller requesting that action be taken on the unmatched flow. There are several possible actions for the flow, including sending the packet to an outgoing port, returning the packet to the controller, dropping the packet, and sending the packet to the next flow table or specific tables. The header of almost every new version of OpenFlow has been expanded to include more matching fields [13].

Matching Rule									Actions	Statistics
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP src	IP dst	TCP psrc	TCP pdst	1. Forward packet to port(s) 2. Encapsulate and forward to controller 3. Drop packet	Counters

Figure 5. OpenFlow v1.0 flow-table Fields [32]

4. METHODOLOGY

In this section, the simulation environment will be explained. Also, 3 different network topologies (single, linear, and tree) will be demonstrated. Lastly, the method for analyzing and comparing four different attributes (delay, jitter, packet loss, and bitrate) will be described.

4.1. Simulation environment

The test environment described in this section was created using virtualization technology on a single personal computer (PC) with an Intel® Core™ i7-7500U 2.70 GHz CPU with 2 cores, 4 logical processors, and 12 GB of RAM. VMware Workstation Pro installed on the PC, ubuntu was used to create virtual machine. the virtual machine was created with 1 core, 4 GB of RAM, and 20 GB of storage. The VM contains Mininet-WiFi emulator with implemented Ryu and POX controllers. Mininet-WiFi is used because it is open source and provides easy modeling of nodes, links, controllers and all network components. On the other hand, the reason for selecting Ryu and POX controllers because both controller uses python as the programming language. Python 3 is used as a scripting language to write the custom network topology.

4.2. Topology

The evaluation procedure comprises building an SDN topology with a Mininet-WiFi emulator and a number of hosts and switches. Three topologies are used to study their performances. The topologies adopted, namely; single, linear, and tree for wired and wireless are depicted in Figure 6.

- 1) Single: in a single topology, only one switch is used, and all the hosts are connected to that switch. Figure 6(a) and Figure 6(b) show the emulated single network of 8 hosts.
- 2) Linear: linear topology is similar to bus topology in that each OpenFlow enabled switch is connected in a straight line. Figure 6(c) and Figure 6(d) depict an emulated linear network with 7 switches and 8 hosts.
- 3) Tree: two terms define tree topology: depth and fanout. Depth specifies the number of switch levels, while fanouts reflect the number of output ports to which switches/hosts will connect. Figure 6(e) and Figure 6(f) depict an emulated tree network with a depth and fanout equal to 2 (7 switches and 8 hosts).

All the nodes have been assigned a unique IP address from the 10.0.0.0/24 address range and a unique MAC address. The IP/MAC addresses are (10.0.0.1/00:00:00:00:00:01) for node *h1*, (10.0.0.8/00:00:00:00:00:08) for node *h8*. The hosts were placed within a range of 100 m from the OpenFlow switches and access points. The wired connection was set to fast ethernet while the wireless was chosen to be 802.11g. To make switches/access points connect to Ryu or POX controller, the 127.0.0.1 virtual loopback IP address was used. The elements that are used to forward traffic from one host to another are switches and access points that support the OpenFlow v1.0 protocol. The Openflow v1.0 was used because POX controller support only this version and to make comparison fair.

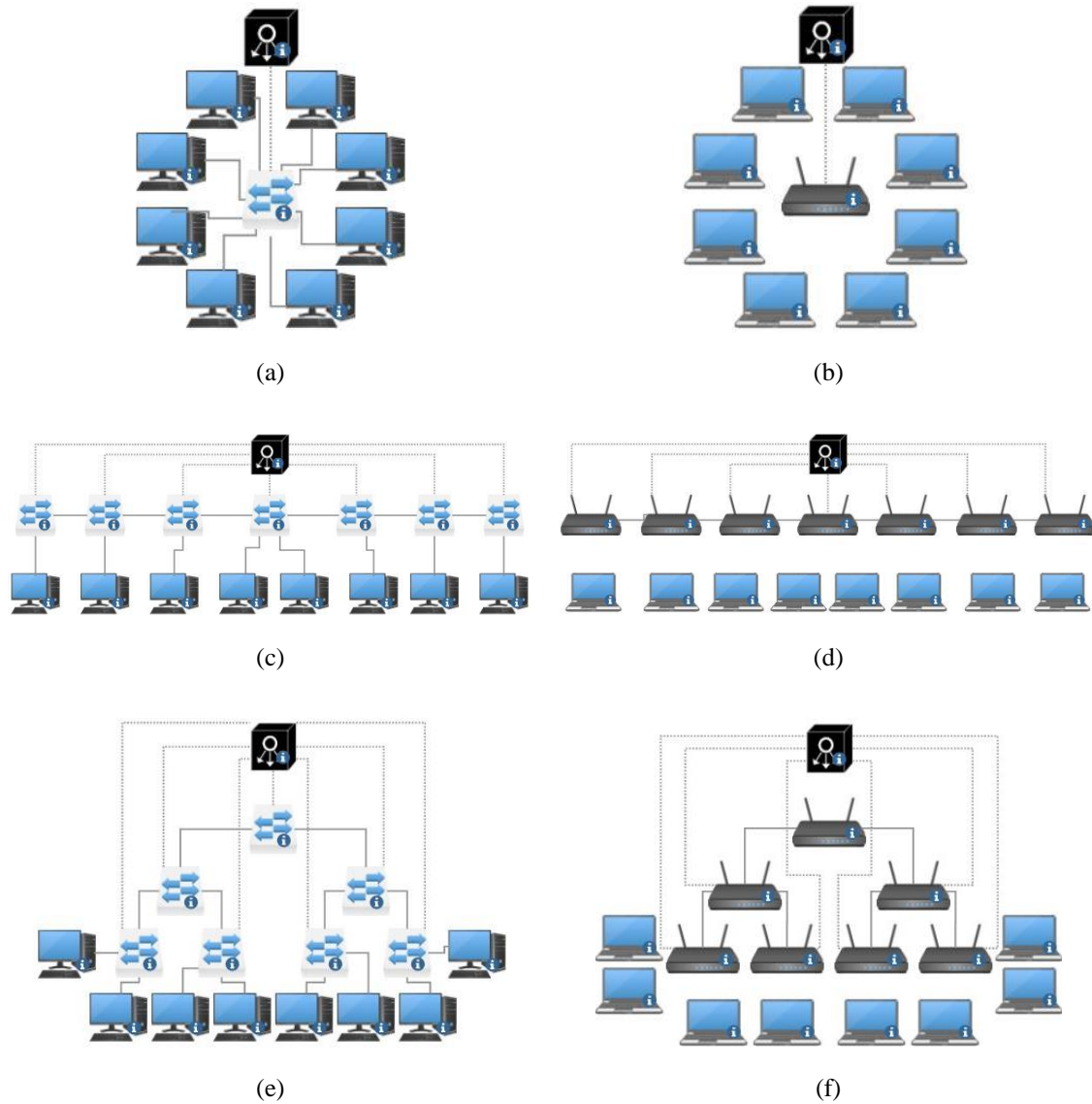


Figure 6. Network topologies in this research: (a) wired single topology, (b) wireless single topology, (c) wired linear topology, (d) wireless linear topology, (e) wired tree topology, and (f) wireless tree topology

4.3. Metrics

After the network design was implemented, its performance was tested by computing the main network's performance characteristics such as delay, jitter, packet loss, and bitrate using the D-ITG traffic generator. Following the completion of each topology in Mininet-WiFi, the first and last nodes will be picked and set using Xterm, a typical terminal emulator. In both trials, host B serves as the server, while host A serves as the client. To begin the performance analysis, host A is set up with the D-ITG command: `./ITGSend -T UDP -m rttm -a 10.0.0.8 -c 128 -C 1000000 -t 10000`. While host B is configured with: `./ITGRecv` command. Table 2 show the parameter of D-ITG command.

Table 2. Parameters of the D-ITG command

Variable	Description
-T <protocol>	Layer 4 protocol (default: UDP)
-m <meter>	Set the type of meter (default: owdm): - Owdm (one-way delay meter) - Rttm (round-trip time meter)
-a <dest_address>	Set the destination address (default: 127.0.0.1)
-c <pkt_size>	Constant (default: 512 bytes)
-C <rate>	Constant (default: 1000 pkts/s)
-t <duration>	Set the generation duration in ms (default: 10000 ms)

All procedures were carried out in two separate test sets for both controllers. The constant rate of the wired network was set to 10^6 pks/s, while the constant rate of the wireless network was set to 10^5 pks/s. To measure delay, jitter, and packet drop, the UDP protocol was employed. On the other hand, when measuring bitrate, UDP was replaced by TCP in the D-ITG command. We run tests ten times in each stage and take the average of the ten results as the final result. Furthermore, we gradually increase the packet size from 128 to 1,024 by a multiple of 2. Table 3 show the comparison between our research methodology and the papers reviewed in section 2.

Table 3. Comparison between different methodology

Reference	Controller	Topology	Performance	Tools
[17]	POX, Ryu, ONOS, floodlight	Linear	Delay, jitter, throughput	iPerf, Ping
[18]	Mininet controller	Single	Delay, jitter	iPerf, Ping
[19]	Ryu	Single	Delay, throughput	iPerf, Ping, Wireshark
[20]	POX	Single, linear, tree	Delay, throughput, packet loss	Wireshark
[21]	Ryu	Tree	Delay, jitter, throughput, packet loss	Wireshark, iPerf, Ping
[22]	ODL, ONOS	Single, linear, tree	Delay, jitter, throughput, packet loss	D-ITG
[23]	Ryu	Linear	Delay, throughput	Cbench, Wireshark
Our research	POX, Ryu	Single, linear, tree	Delay, jitter, throughput, packet loss	D-ITG

5. RESULTS

5.1. Average delay

In wired networks, Ryu outperforms POX in the three topologies (especially in linear and tree) as shown in Figure 7. It also showed that the delay in Ryu increases with increasing the size of the packet, in contrast with POX where the delay decreasing especially in linear and tree topologies. Figure 8 shows that both controllers get almost the same performance in wireless networks, with the best result for the packet size of 128 bytes and the worst for 512 bytes. So, in our scenarios SDN work better in small size packet applications. In addition, tree topology works better than linear in the POX controller.

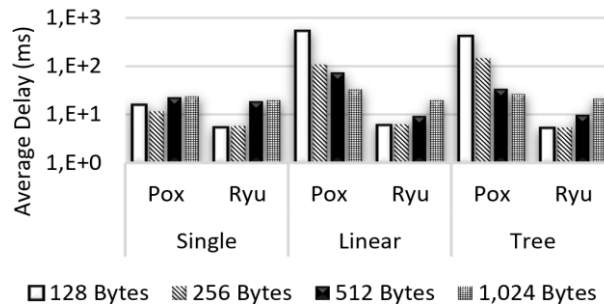


Figure 7. Average delay in wired network

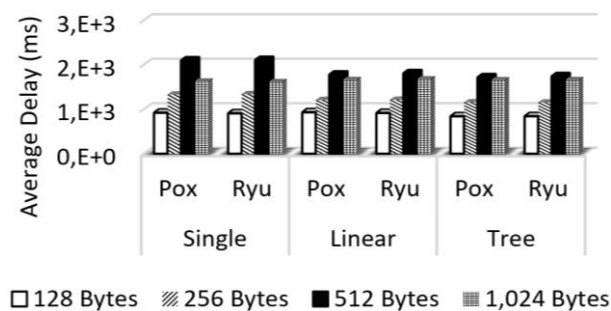


Figure 8. Average delay in wireless network

5.2. Average jitter

Figure 9 shows that the POX controller has higher jitter, especially when the packet size is small, on the other hand, Ryu gets lower jitter as the packet size decreases. The jitter in both controllers behave like delay's results. In wireless networks, the average jitter is the same in both controllers, as shown in Figure 10. Also, tree topology works better than linear with POX controller. The packet size of 128 bytes was shown to be suitable for applications that need less jitter, while 1,024 bytes is the worst.

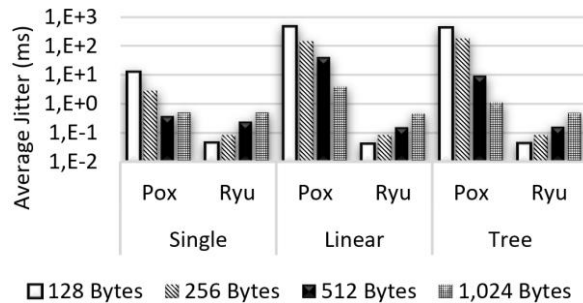


Figure 9. Average jitter in wired network

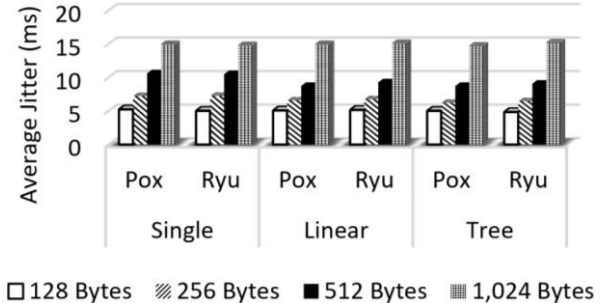


Figure 10. Average jitter in wireless network

5.3. Dropped packets

For the packet drop, Ryu performs better than POX in all topologies as shown in Figure 11. Also, tree get better results in POX controller than linear. Figure 12 shows that the worst packet drop is 13% in the POX with linear topology when the packet size is 1,024 bytes, while it is better in the packet size of 256 bytes with about 4.5% packet drop.

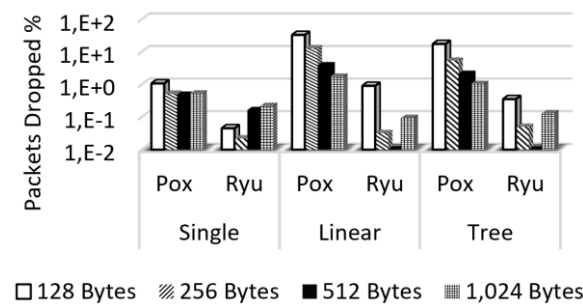


Figure 11. Packet drop in wired network

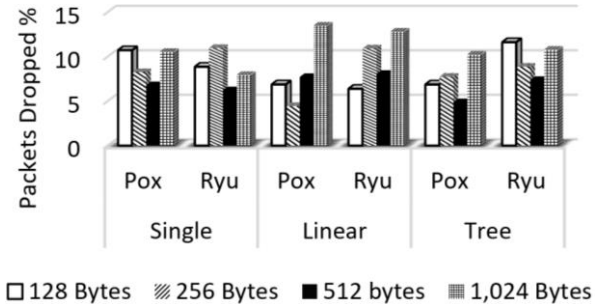


Figure 12. Packet drop in wireless network

5.4. Average bitrate

Figure 13 shows that Ryu and POX get almost the same average bitrate in all scenarios. It can be also shown that linear topology gets a better bitrate in comparison to the tree in POX scenarios. In wireless scenarios the average bitrate of Ryu is better than POX and the tree topology gets higher bitrate than linear in both Ryu and POX controllers as shown in Figure 14, the packet size of 128 Bytes is the worst in wireless network.

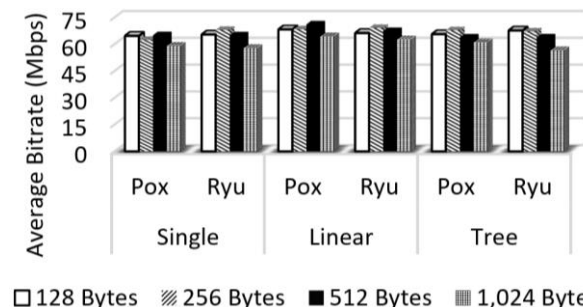


Figure 13. Average bitrate in wired network

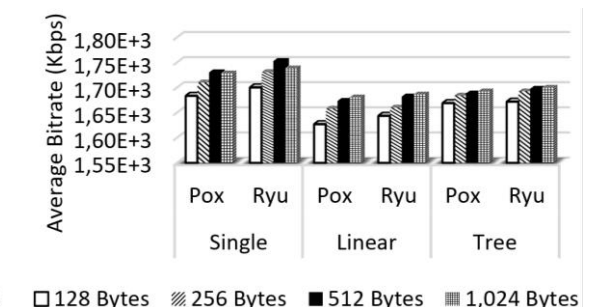


Figure 14. Average bitrate in wireless network

6. CONCLUSION

In the last few years, there has been a lot of progress in technology. Traffic analysis is one of the important parts of SDN's application field. The SDN controller serves as the network's central processing unit, analyzing and monitoring real-time data traffic. Monitoring and analyzing real-time data traffic is an important technique in any networking approach for observing the passage of data packets from the source host to the destination host. In this study, many comparisons were made. First, the performance of the two most prominent open source SDN controllers, POX and Ryu, is compared. Second, many topologies were implemented with comparing linear and tree topologies. Third, various packet sizes were compared, which provides the researchers with information about the suitable packet sizes in different applications. The D-ITG tool and the Mininet-WiFi emulator were used to conduct an impartial experimental analysis based on active measurement. Performance is measured using metrics including delay, jitter, packet loss, and throughput. From the evaluations, Ryu exhibited the best performance for most scenarios and metrics. While the performance of POX was drastically poor, the obtained results showed that the RYU seems to have less delay and jitter in wired networks as the size of packets decreases, in contrast with the POX controller. In the case of measuring packet loss, Ryu controller gets a 0% packet drop at a packet size of 512 bytes in both linear and tree topologies. In difference with the POX controller that recorded about 32% at 128 bytes in wired with linear topology, which is the worst case in all scenarios. The similar work may be done in the future to test the controller's security and robustness. The same operation can also be done on complicated networks, particularly those with multipath connections.




REFERENCES

- [1] B. Goswami and S. S. Asadollahi, "Enhancement of LAN infrastructure performance for data center in presence of network security," *Next-Generation Networks: Advances in Intelligent Systems and Computing*, Singapore: Springer, vol. 638, 2018, doi: 10.1007/978-981-10-6005-2_44.
- [2] S. Asadollahi and B. Goswami, "Experimenting with scalability of floodlight controller in software defined networks," in *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECOT)*, 2017, pp. 288-292, doi: 10.1109/ICEECOT.2017.8284684.
- [3] R. Masoudi and A. Ghaffari, "Software defined networks: A survey," *Journal of Network and Computer Applications*, vol. 67, pp. 1–25, 2016, doi: 10.1016/j.jnca.2016.03.016.
- [4] M. T. Islam, N. Islam, and M. Al Refat, "Node to Node Performance Evaluation through RYU SDN Controller," *Wireless Personal Communications*, vol. 112, pp. 555–570, 2020, doi: 10.1007/s11277-020-07060-4.
- [5] E. Amiri, E. Alizadeh, and M. H. Rezvani, "Controller selection in software defined networks using best-worst multi-criteria decision-making," *Bulletin of Electrical Engineering and Informatics*, vol. 9, no. 4, pp. 1506–1517, 2020, doi: 10.11591/eei.v9i4.2393.
- [6] S. Askar and F. Ketii, "Performance Evaluation of Different SDN Controllers: A Review," *International Journal of Science and Business*, vol. 5, no. 6, pp. 67-80, 2021, doi: 10.5281/zenodo.4742771.
- [7] B. A. A. Nunes, M. Mendonca, X. -N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014, doi: 10.1109/SURV.2014.012214.00180.
- [8] D. Cabarkapa and D. Rancic, "Performance Analysis of Ryu-POX Controller in Different Tree-Based SDN Topologies," *Advances in Electrical and Computer Engineering*, vol. 21, no. 3, pp. 31–38, 2021, doi: 10.4316/AECE.2021.03004.
- [9] A. Maleki, M. Hossain, J. Georges, E. Rondeau, and T. Divoux, "An SDN Perspective to Mitigate the Energy Consumption of Core Networks – GEANT2," in *International SEEDS conference 2017*, 2017. [Online]. Available: https://www.researchgate.net/publication/319876305_An_SDN_Perspective_to_Mitigate_the_Energy_Consumption_of_Core_Networks_-_GEANT2
- [10] S. Asadollahi and B. Goswami, "Revolution in Existing Network under the Influence of Software Defined Network," in *2017 4th International Conference on "Computing for Sustainable Global Development"*, 2017, pp. 1012–1017. [Online]. Available: https://www.researchgate.net/publication/319537055_Revolution_in_Existing_Network_under_the_Influence_of_Software_Defined_Networks_SDN
- [11] A. Mondal, S. Misra, and I. Maity, "AMOPE: Performance Analysis of OpenFlow Systems in Software-Defined Networks," *IEEE Systems Journal*, vol. 14, no. 1, pp. 124–131, 2020, doi: 10.1109/JSYST.2019.2912843.
- [12] G. Aziz and S. Askar, "Software Defined Network Based VANET," *International Journal of Science and Business*, vol. 5, no. 3, pp. 83–91, 2021, doi: 10.5281/zenodo.4497640.
- [13] S. Mostafavi, V. Hakami, and F. Paydar, "Performance Evaluation of Software-Defined Networking Controllers: A Comparative Study," *Journal of Computer and Knowledge Engineering*, vol. 2, no. 2, pp. 63–73, 2020, doi: 10.22067/cke.v2i2.84917.
- [14] O. Salman, I. H. Elhadj, A. Kayssi, and A. Chehab, "SDN controllers: A comparative study," *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, 2016, , pp. 1-6, doi: 10.1109/MELCON.2016.7495430.
- [15] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," in *Proc. of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015, doi: 10.1109/JPROC.2014.2371999.
- [16] T. Zhang and F. Hu, "Controller architecture and performance in software-defined networks," in *Network Innovation through OpenFlow and SDN Principles and Design*, CRC Press, Taylor and Francis Group, 2014, pp. 121–130.
- [17] S. Islam, M. A. I. Khan, S. T. Shorno, S. Sarker, and M. A. Siddik, "Performance Evaluation of SDN Controllers in Wireless Network," in *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, 2019, pp. 1-5, doi: 10.1109/ICASERT.2019.8934553.
- [18] P. E. Numan *et al.*, "On the latency and jitter evaluation of software defined networks," *Bulletin of Electrical Engineering and Informatics*, vol. 8, no. 4, pp. 1507–1516, 2019, doi: 10.11591/eei.v8i4.1578.
- [19] S. Bhardwaj and S. N. Panda, "Performance Evaluation Using RYU SDN Controller in Software-Defined Networking Environment," *Wireless Personal Communications*, vol. 122, pp. 701–723, 2022, doi: 10.1007/s11277-021-08920-3.
- [20] D. Kumar and M. Sood, "Performance Analysis of Impact of Network Topologies on Different Controllers in SDN," in *International Conference on Innovative Computing and Communications*, 2021, pp. 725-735, doi: 10.1007/978-981-15-5148-2_63.




- [21] H. Babbar and S. Rani, "Performance evaluation of QoS metrics in software defined networking using ryu controller," *IOP Conference Series: Materials Science and Engineering*, 2021, vol. 1022, doi: 10.1088/1757-899X/1022/1/012024.
- [22] L. Mamushiane and T. Shoji, "A QoS-based evaluation of SDN controllers: ONOS and OpenDayLight," *2021 IST-Africa Conference (IST-Africa)*, 2021, pp. 1-10. [Online]. Available: <https://ieeexplore.ieee.org/document/9577034>
- [23] A. T. A. -Salih, "Performance Evaluation of Ryu Controller in Software Defined Networks," *Journal of Al-Qadisiyah for Computer Science and Mathematics*, vol. 14, no. 1, pp. 1-7, 2022. [Online]. Available: https://www.researchgate.net/publication/359024814_Performance_Evaluation_of_Ryu_Controller_in_Software_Defined_Networks
- [24] L. Cominardi, C. J. Bernardos, P. Serrano, A. Banchs, and A. de la Oliva, "Experimental evaluation of SDN-based service provisioning in mobile networks," *Computer Standards and Interfaces*, vol. 58, pp. 158-166, 2018, doi: 10.1016/j.csi.2018.01.004.
- [25] D. Dholakiya, T. Kshirsagar, and A. Nayak, "Survey of Mininet Challenges, Opportunities, and Application in Software-Defined Network (SDN)," *Smart Innovation, Systems and Technologies*, vol. 196, pp. 213-221, 2021, doi: 10.1007/978-981-15-7062-9_21.
- [26] S. Badotra and S. N. Panda, "Software-defined networking: A novel approach to networks," *Handbook of Computer Networks and Cyber Security*, 2019, pp. 313-339, doi: 10.1007/978-3-030-22277-2_13.
- [27] U. Humayun, M. Hamdan, and M. N. Marsono, "Early Flow Table Eviction Impact on Delay and Throughput in Software-Defined Networks," in *2021 11th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, 2021, pp. 7-12, doi: 10.1109/ICCSCE52189.2021.9530933.
- [28] M. Bano, A. Qayyum, R. N. B. Rais, and S. S. A. Gilani, "Soft-Mesh: A Robust Routing Architecture for Hybrid SDN and Wireless Mesh Networks," *IEEE Access*, vol. 9, pp. 87715-87730, 2021, doi: 10.1109/ACCESS.2021.3089020.
- [29] P. Chauhan and M. Atulkar, "Achieving Enhanced Network Performance in UDP and TCP Traffic of Software Defined Networking by Selecting Java Based Controllers Decisively," *International Journal Innovative Technology Exploring Engineering*, vol. 9, no. 7, pp. 268-274, 2020, doi: 10.35940/ijitee.g4868.059720.
- [30] S. Badotra and S. N. Panda, "Experimental comparison and evaluation of various OpenFlow software defined networking controllers," *International Journal of Applied Science and Engineering*, vol. 17, no. 4, pp. 317-324, 2020. [Online]. Available: <https://gigvvy.com/journals/ijase/articles/ijase-202012-17-4-317.pdf>
- [31] H. M. Anitha and P. Jayarekha, "A Proposed SDN-Based Cloud Setup in the Virtualized Environment to Enhance Security," *CTIS 2020: Information and Communication Technology for Intelligent Systems*, vol. 196, pp. 453-461, 2020, doi: 10.1007/978-981-15-7062-9_45.
- [32] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using open flow: A survey," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 493-512, 2014, doi: 10.1109/SURV.2013.081313.00105.
- [33] M. Hamdan *et al.*, "Flow-Aware Elephant Flow Detection for Software-Defined Networks," *IEEE Access*, vol. 8, pp. 72585-72597, 2020, doi: 10.1109/ACCESS.2020.2987977.
- [34] S. I. Boucetta and Z. C. Johanyák, "Review on Networks Defined by Software," *Gradus*, vol. 7, no. 2, pp. 230-238, 2020, doi: 10.47833/2020.2.csc.001.
- [35] A. Shirvar and B. Goswami, "Performance comparison of software-defined network controllers," in *2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, 2021, pp. 1-13, doi: 10.1109/ICAECT49130.2021.9392559.
- [36] R. C. Meena, M. Bundeale, and M. Nawal, "RYU SDN Controller Testbed for Performance Testing of Source Address Validation Techniques," in *2020 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE)*, 2020, pp. 1-6, doi: 10.1109/ICETCE48199.2020.9091748.

BIOGRAPHIES OF AUTHORS






Salar Jamal Rashid    received the B.Sc. and M.Sc. degree in Computer Engineering Technology from Northern Technical University, Iraq in 2011 and 2016 respectively. His current research interests include Internet of Things, Computer Networks, and Information and communication Technology. He can be contacted at email: salar.jamal@ntu.edu.iq.



Ahmed Maamoon Alkababji    received the B.Sc., M.Sc. and PhD degree in Electrical Engineering from University of Mosul, Iraq in 1994, 1996 and 2007. Currently, a professor at computer engineering department in University of Mosul, Iraq. Research interest in signal processing, realtime systems and biometric engineering. He can be contacted at email: ahmedalkababji72@uomosul.edu.iq.



Abdul Sattar Mohammed Khidhir    received B.Sc. and M.Sc. in Electronics Engineering from University of Mosul / Iraq. He received Ph.D. in Communications Engineering from University of Mosul too. He is working at Mosul Technical Institute / Northern Technical University - Iraq. He supervised many Ph.D. and M.Sc. projects, and published many papers in many fields. He can be contacted at email: abdulsattarmk@ntu.edu.iq.