Journal of Advanced Research in Applied Sciences and Engineering Technology

Journal homepage:
https://semarakilmu.com.my/journals/index.php/applied_sciences_eng_tech/index
ISSN: 2462-1943

Toward Adaptive and Scalable Topology in Distributed SDN Controller

Virakwan Hai Kelian[1,*], Mohd Nazri Mohd Warip[1,3,*], R. Badlishah Ahmad[1,3], Phaklen Ehkan[1,3], Fazrul Faiz Zakaria[2,3], Mohd Zaizu Ilyas[1,3]

1 Advanced Computing, Centre of Excellence, Universiti Malaysia Perlis (UniMAP), Perlis, Malaysia
2 Micro System Technology, Centre of Excellence (CoE), Universiti Malaysia Perlis (UniMAP), Perlis, Malaysia
3 Faculty of Electronic Engineering Technology, Universiti Malaysia Perlis, Pauh Putra Campus, 02600 Arau, Perlis, Malaysia

ARTICLE INFO

Article history:
Received 6 October 2022
Received in revised form 14 December 2022
Accepted 16 February 2023
Available online 10 March 2023

Keywords:
Distributed SDN controller; OpenDaylight; Controller performance; Scalability

ABSTRACT

The increasing need for automated networking platforms like the Internet of Things, as well as network services like cloud computing, big data applications, wireless networks, mobile Internet, and virtualization, has driven existing networks to their limitations. Software-defined network (SDN) is a new modern programmable network architectural technology that allows network administrators to control the entire network consistently and logically centralized in software-based controllers and network devices become just simple packet forwarding devices. The controller that is the network's brain, is mostly based on the OpenFlow protocol and has distinct characteristics that vary depending on the programming language. Its function is to control network traffic and increase network resource efficiency. Therefore, selecting the right controllers and monitoring their performance to increase resource usage and enhance network performance metrics is required. For network performance metrics analysis, the study proposes an implementation of SDN architecture utilizing an open-source OpenDaylight (ODL) distributed SDN controller. The proposed work evaluates the deployment of distributed SDN controller performance on three distinct customized network topologies based on SDN architecture for node-to-node performance metrics such as delay, throughput, packet loss, and bandwidth use. The experiments are conducted using the Mininet emulation tool. Wireshark is used to collect and analyse packets in real-time. The results obtained from the comparison of networks are presented to provide useful guidelines for SDN research and deployment initiatives.

1. Introduction

The networking sector is being challenged with a new paradigm, which some consider extremely transformational. This new paradigm seeks to transform the way networks are developed, requiring networks to be flexible, secure, and keep the quality of service while still complying with policies and standards. Due to the network issues that need different solutions and intention to overcome the limitations in the conventional network, Software-defined networking (SDN) is presented. The main

* *Corresponding author.*
E-mail address: virakwan@studentmail.unimap.edu.my, nazriwarip@unimap.edu.my

https://doi.org/10.37934/araset.30.1.115131

idea behind the SDN is the "Stanford Clean Slate Project" in the year 2007 [1]. The project's primary objective is to develop a new architecture for business networks that is both simple to use and secure.

1.1 SDN Architecture

The architecture is the key component of the existence of SDN as, through the design, SDN is claimed to be able to overcome the constraint in the traditional networks [2]. SDN architecture separates the network into three (3) different layers data plane, control plane, and management plane [3] as shown in Figure 1.

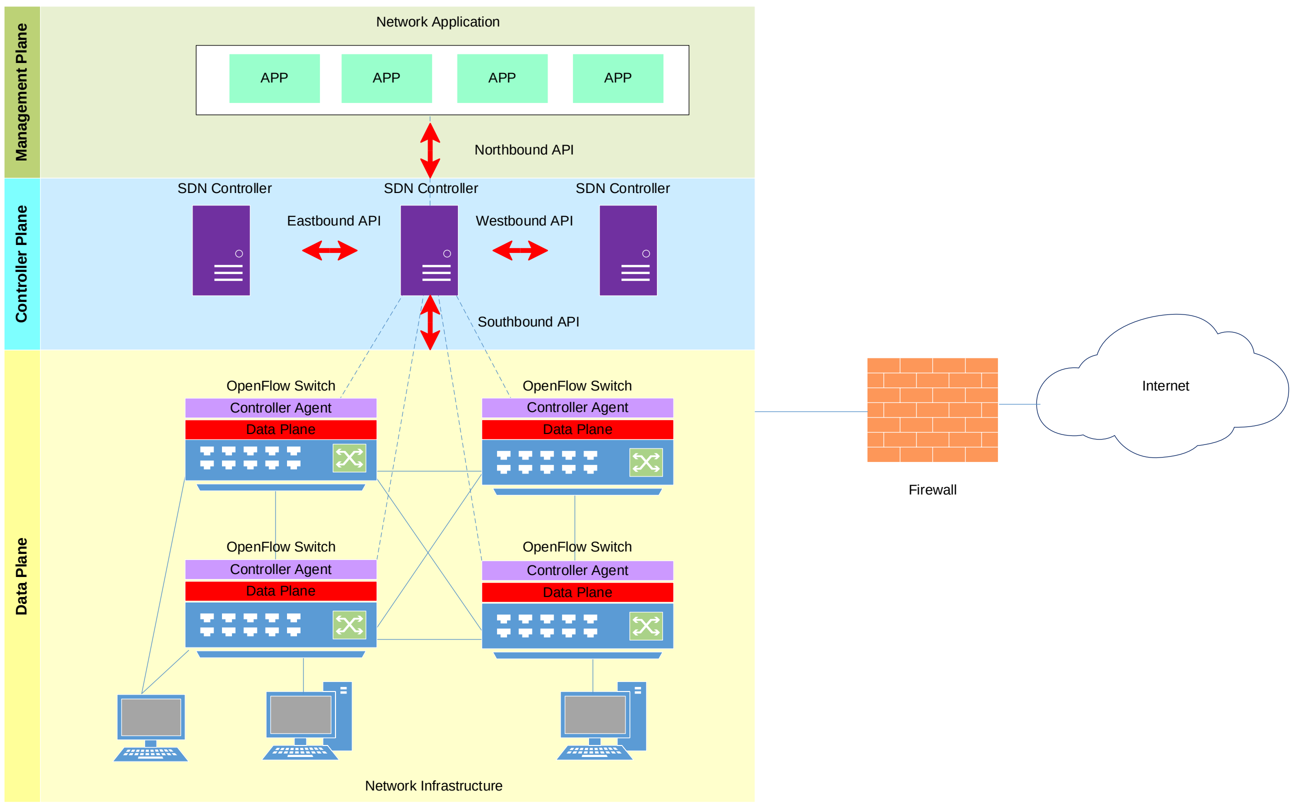


Fig. 1. SDN architecture

As indicated in Figure 1, there are two (2) basic characteristics of SDN architecture. The first is the isolation of data and control plane which is derived from the telephone network system, and the second is the integration of network intelligence in a centralized controller [4-5]. The network control is done by a controller that must have connectivity with all nodes in the network.

1.1.1 Data plane

The Data plane layer or also known as the infrastructure layer consists of network devices such as a physical switch or virtual switch, router, gateway, server, and access point [6]. Generally, this layer allows device connectivity and data transfer [7]. The data plane consists of hardware devices that are responsible to handle the traffic following the rules set by the control plane. It is responsible for the same functions as it is in the conventional network to forward the data, but routing decisions are excluded from this layer [8]. The connection between the data plane and the control plane can be performed through OpenFlow protocol and Southbound application programming interfaces (APIs) [9]. Southbound APIs are used to send the instructions and receive the information from the data plane to the controller [10]. As a result, SDN switches are composed primarily of three components, the OpenFlow protocol, a flow table, and a secure channel. For each OpenFlow switch securely connected to the controller, the interface is responsible to become a secure channel and a flow table is used to process the packets that are sent through them [11-12].

1.1.2 Control plane

The control plane consists of a controller that manages the data plane devices and establishes traffic flows according to network rules. All the decision routing in the OpenFlow switch will be controlled by the SDN controller. The controller serves as the network brain for decision routing at switches based on the OpenFlow protocol [13]. Meanwhile, the SDN controller is an application to the centralized control point. Due to the architecture of SDN consisting of three (3) layers and the control plane being the middle layer, the connection between the control plane and data plane is through Southbound API. While the connection between the control plane and management plane can be performed using Northbound API. The control plane can be implemented as a centralized controller, or distributed controller and integrated both centralized and distributed known as hybrid centralization only one controller manages the flow table of all SDN switches [14]. In contrast, for the distributed controller deployment, the control plane may consist of many controllers that may interact with one another through the Westbound and eastbound interfaces [15].

1.1.3 Management plane

The management plane or known as the application layer is at the top layer in the SDN architecture [16]. This layer covered the software-related operations and handle security applications such as network virtualization, mobility management, firewall, Intrusion Prevention Systems (IPS), and Intrusion Detection Systems (IDS) [17]. This layer interacts with the control layer using Northbound API [18-19].

1.2 Distributed Controllers

The first concepts of the controller were introduced to allow network administrators to set flow-based policies for their networks [1]. Further in the year 2008, the OpenFlow protocol and a program software were proposed which served as the beginning point for network programmability to accomplish a range of control applications [20]. OpenFlow is the most popular SDN protocol that implemented SDN communication standards for communication between the controller and other networking devices [21]. In recent years, numerous OpenFlow controllers have been created and made available for research and commercial use. These can be separated into two categories: centralized controllers and distributed controllers [22]. Ryu and Floodlight are the most well-known centralized SDN controllers [22]. However, a single physically or logically centralized controller to perform forwarding nodes presents a major bottleneck in a large-scale network [23]. It is a single point of failure that can cause the network to lose intelligence, become inefficient, experience unexpectedly long delays due to the controller's distance from the switches, lack scalability support for big SDN networks, and have limited controller processing power [24]. To overcome these issues, research has found that distributed SDN controllers can be used [25]. A variety of controllers has been developed, but OpenDayLight (ODL) has gained the most attention among the distributed controller platforms due to its excellent scalability, support for dependability, and ability to handle consistency [26]. Numerous OpenFlow controllers have been developed and made available for research and commercial usage in recent years. The following Figure 2 summarizes the various types of OpenFlow SDN controllers based on their control plane architecture. The majority of SDN controllers are based on the OpenFlow protocol that implemented SDN communication standards for interacting between the controller and other networking devices. An example, NOX was the initial release of an OpenFlow controller in early 2008 [37]. Afterward, various alternative OpenFlow controllers have been launched with distinct characteristics in terms of high-performance, multithreaded OpenFlow controllers and offer high availability.

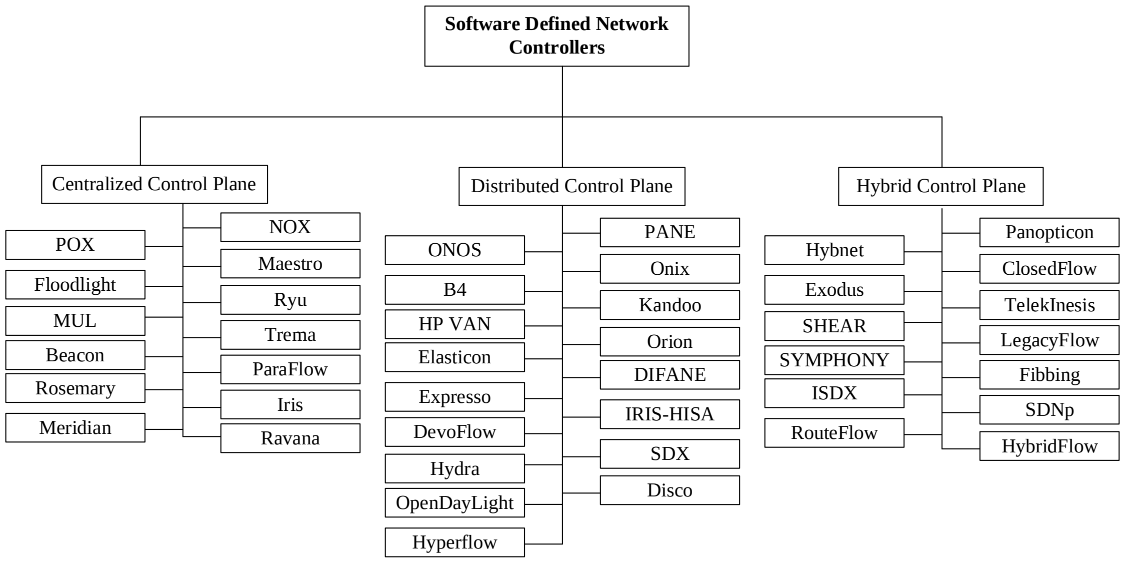


Fig. 2. SDN controllers

However, a physical centralized controller suffers from a single point of failure. Therefore, different SDN distributed controllers have been presented to give some amount of performance, security, availability, and scalability as indicated in Figure 2. Controllers such as Hyperflow [38], Kandoo [39], and, ONOS [22] provide a series of distributed controllers and each controller has an equivalent global view of network topology [40]. The distributed controller controls the entire network while preserving sophisticated requirements such as performance metrics, security, load balancing, efficiency, good features, stable architecture, availability, fault tolerance, and efficient convergence time [23, 33-41].

The summary of the primary features for the most prominent sophisticated distributed SDN controller platforms is given in Table 1. Each distributed controller has special features that differ according to the programming language and functionality employed.

Table 1
Main characteristics of distributed controllers [10, 26]

Controllers	Control Plane Design	Programming Language	Scalability	Reliability	Consistency
ONIX	Hierarchical	Python, C	Very Good	Good	Weak
HyperFlow	Hierarchical	C++	Good	Good	Moderate
Orion	Hierarchical	Java	Very Good	Very Good	Strong
ONOS	Hierarchical	Java	Very Good	Good	Weak
OpenDaylight	Hierarchical	Java	Very Good	Good	Strong
B4	Hierarchical	Python, C	Good	Good	N/A
Kandoo	Hierarchical	C, C++, Python	Very Good	Limited	N/A
DISCO	Flat	Java	Good	Limited	Strong (inter-domain)
SDX	Flat	Python	Limited	N/A	Strong
DevoFlow	N/A	Java	Good	N/A	N/A
DIFANE	N/A	N/A	Good	N/A	N/A

Some of the controllers met some performance requirements better than others but failed in some other aspects. Even though the distributed control architecture is considered a scalable solution when compared to the centralized control model, the capability of the SDN controller to assure service continuity while preserving high performance requires proper attention to any proposal or design [33].

1.3 Reliable Connection Protocol in SDN

The basic connection of an SDN network is formed on a data plane that is comprised of network devices such as OpenFlow switches that will execute regular forwarding that is controlled by a logically centralized SDN controller. Switches comprise essentially three components included flow table, secure channel, and OpenFlow protocol [10]. The interface is function as a secure route to link each OpenFlow switch to the controllers [11]. A flow table is used to process packets in switches [12]. The entire packet forwarding process in an OpenFlow switch is shown in Figure 3.

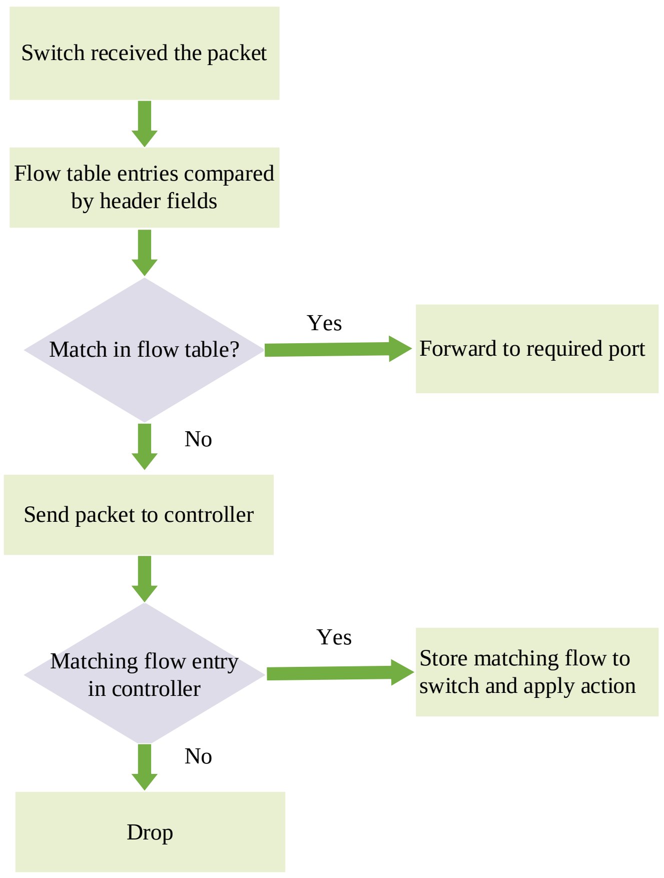

Switch received the packet
Flow table entries compared by header fields
Match in flow table?
Yes
Forward to required port
No
Send packet to controller
Matching flow entry in controller
Yes
Store matching flow to switch and apply action
No
Drop

Fig. 3. The flow of packets in switches [10, 42-43]

In Figure 3, when a packet arrives at a switch, the switch examines the flow table for an entry that matches the packet's header information. If the rule is matched, the packet is forwarded. In the missing of a match, the switch sends an asynchronous message to the controller. Based on the programmed policies, the controller transmits the message to the appropriate control applications as an event. The applications process the event and, if needed, return a message containing action instructions. Controllers configure network devices using Openflow. The SDN controller instructs the switch on what actions they should perform through southbound API. The Openflow protocol is the most efficient method of communicating between SDN controllers and switches using the Southbound API. It is a layer on top of the Transmission Control Protocol (TCP) and specifies the implementation of Transport Layer Security (TLS). Controllers used TCP port 6653 for switches that wanted to connect, and OpenFlow protocol unofficially used port 6633 [44].

Figure 4 illustrates the OpenFlow connection establishment process, in which switches start a secure TCP channel to the controller, allowing the controller to manage switches using the OpenFlow protocol [45]. The IP address of the controller is accessible through switch configuration. Moreover, the controller may also have recognized the switch and performed the connection setup. As seen in Figure 4, the requirement has been met with a three-ways-hand check. The controller is then able to identify all connected switches and initiate the connection. Upon establishing a secure connection, the controller, and switches exchange hello messages to determine the highest OpenFlow version supported by both entities. Second, if the OpenFlow versions in both nodes are compatible, the controller requests the characteristics of the connected switch using a features request message. Finally, after receiving the message of the requested features, the switches respond with a message informing the controller of supported features [30, 45-46].

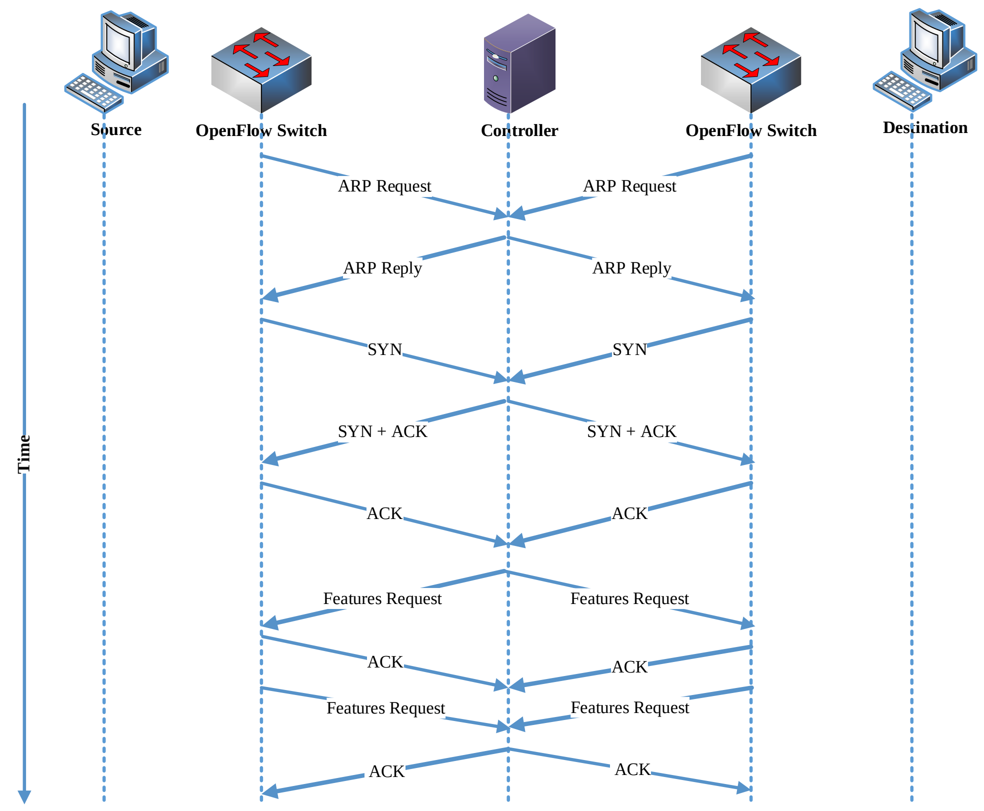


Fig. 4. OpenFlow connection establishment process [30, 45-46]

1.4 Distributed Controller Performance Evaluation

Much research focusing on distributed controller performance evaluation and comparison has been conducted in recent years, and this section reviews a few of them. Koponen *et al.,* [27] introduced and evaluated the performance of the first distributed SDN controller named ONIX. The evaluation was focused more on the reliability and scalability of ONIX in a large-scale network. However, because the ONIX figure provides a general API for the control plane and it is still a close source, more research is needed to examine the performance. An experiment was done in the year 2014 using the distributed SDN control platform Open Network Operating System (ONOS) to examine the performance, scalability, and availability needs of large operator networks [28]. The authors evaluated the scalability, fault tolerance, and performance measures including latency and throughput of two ONOS prototypes. According to the authors, ONOS still must be improved to accommodate use cases like core network traffic engineering and scheduling.

Mamushiane *et al.,* [29] provided a comparative study on the performance of popular open-source controllers such as ONOS, Ryu, Floodlight, and OpenDayLight in terms of latency and throughput using an OpenFlow benchmarking tool called Clench. The author recommends using OpenDayLight since it has a lot of APIs and vendor support. In terms of performance, ONOS had the best throughput while Ryu had the lowest latency. The same researchers compared the QoS performance of ONOS and OpenDayLight distributed SDN controllers [22]. Mininet is used to emulate three different topologies: single, linear, and tree. The purpose of the observation was to assess performance indicators such as one-way trip delay, jitter, and packet loss. In all topologies, the testing results reveal that OpenDayLight has much higher latency, jitter, and packet loss than ONOS.

Priya *et al.,* in Bhardwaj and Panda [30] compared the performance of well-known OpenFlow controllers such as NOX, POX, Ryu, and FloodLight by determining packet handling capacity and measuring performance in terms of delay, jitter, throughput, and packet loss using the Distributed internet traffic flow generator (D-ITG). According to the authors, FloodLight offers higher throughput and less delay than alternative controllers. Furthermore, the authors recommended for future work include a comparison of the OpenDayLight and OpenContrail controllers.

Rowshanrad *et al.,* [31] examined QoS metrics of Floodlight and OpenDaylight in terms of latency, packet loss, and network loads in single, linear, and tree topologies using Mininet. The authors determined that OpenDaylight had better latency in tree topology for a network with half of bandwidth traffic, but floodlight can outperform OpenDaylight in terms of packet loss in the heavily loaded network in a tree topology. The authors propose comparing these two controllers in more complex topologies with varying numbers of switches for future work.

Eftimie and Burcoci [32] present the implementation of the OpenDaylight controller in a small environment using Mininet to observe key functionality, stability, and resource usage, as well as analyze primary limitations. The authors concluded that OpenDaylight is a low-power distributed controller and easy-to-use tool. However, there are limits in terms of controller incompatibility with JAVA versions. Furthermore, the author recommends that the performance of this controller be assessed in the future.

Abdullah *et al.,* [33] provided the performance comparison of five SDN controllers libfluid, ONOS, OpenDaylight, POX, and Ryu. The authors develop custom linear topology in Mininet and observe end-to-end throughput and delay by using iPerf and Ping commands. Authors found that libfluid gives the best throughput performance and POX gives the best delay performance.

Ghalwash and Huang [34] suggested a framework for applying QoS in an SDN network. The suggested framework is examined in a fat-tree topology utilizing an OpenDayLight (ODL) controller to evaluate two QoS metrics which are port use and delay. The authors concluded that the proposed framework with the OpenDaylight controller can lower the average delay and reduce average port utilization.

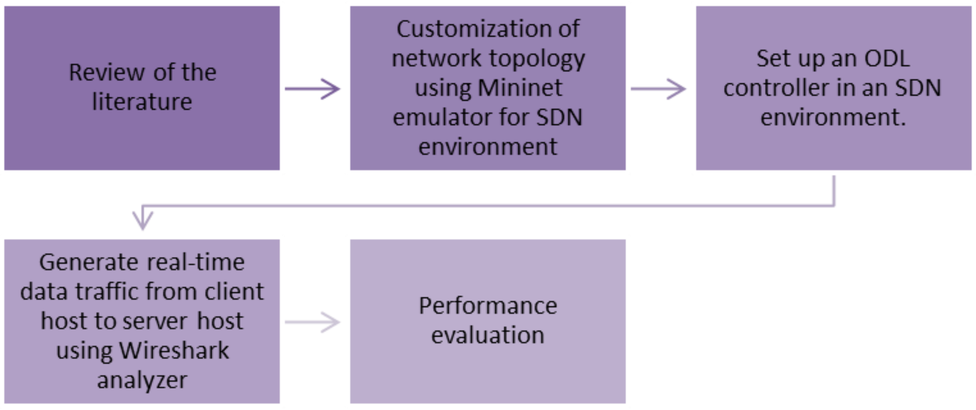
Vilchez and Samiento [35] experimented on ONOS and OpenDaylight controllers to evaluate the abilities of the controller to handle fault tolerance in various fault situations using the Mininet emulator. Authors claimed that the ONOS controller outperforms ODL in terms of switching over to other pathways to ensure service continuity. The author recommended future directions to examine the capabilities of ONOS and ODL to dynamically modify the intentions deployed to avoid the bandwidth reduction in data links.

Lastly, Ahmed Hassan *et al.,* [36] conducted a comparative study of Floodlight and OpenDaylight controllers examining parameters such as new flow generation, flow setup latency, open flow messages, flow misses to the controller, CPU unitization, and memory. The experiment was accomplished by constructing tree topology using the OfNet environment. The authors determined that the Floodlight controller is outperforming the OpenDaylight controller in small occupying memory space, less CPU use, and a smaller number of messages in packages, but the new flow generation is static. However, the OpenDaylight controller is outperforming the floodlight controller in average setup latency. The authors emphasized the weakness of the OfNet emulator that caused instability of the controllers during an experiment. Therefore, the authors propose to use a Mininet emulator with more complex topologies for future study.

The previous studies reveal that the architecture of single controllers is inefficient for network administration. Distributed SDN controllers have been implemented to solve scalability, reduce transmission delay, improve fault tolerance, and avoid packet loss concerns. Although distributed SDN controller platforms have been deployed to solve scalability, lowering transmission delay, improve fault tolerance, and minimize packet loss concerns but there is limited research existing literature based on distributed controllers [10]. To guarantee high quality of services (QoS) performance, the controller should be able to respond to packets in messages promptly. This means that the delay and packet loss must be minimal, and throughput must be maximum. In most studies based on a distributed SDN controller, the placement of the controller is the primary challenge that directly affects its performance. Consequently, there is a research need to construct the structure of distributed SDN controllers with custom topology for traffic engineering and to give an in-depth illustration of performance metrics required for exploring the future of SDN [10]. To the best of our knowledge, the majority of controller evaluation studies did not focus exclusively on distributed controllers, and there is still a gap in implementing SDN architecture with custom topology and providing a detailed representation of various network performance measurements utilizing distributed controllers [10]. As a result, the focus of this research will be on implementing the SDN architecture in the Mininet emulator that includes the OpenDaylight controller for three (3) different custom-designed topologies consisting of OpenFlow switches and network nodes. This research's primary objective is to examine the performance of SDN networks, with a focus on distributed controllers in three (3) various bespoke network designs. The proposed research intends to develop a framework for reporting the evaluation results of node-to-node performance measures such as delay, throughput, packet loss, and bandwidth utilization. Academics, application developers, and service providers can utilize this study to make educated controller selection decisions.

2. Methodology

The purpose of this study is to evaluate the performance of distributed SDN controller, which has the role of controller for a network emulated using Mininet. The methodology used to conduct the suggested research is shown in Figure 5. The first step of the approach is to conduct a literature evaluation on network traffic analysis using SDN controllers to identify research gaps and establish the goals of our study. The Mininet tool is then utilized to construct a custom network for the SDN environment. Mininet is a network simulator that implements the OpenFlow protocol and can construct any arbitrary network consisting of hosts, switches, and connections [33]. Even though network elements are formed by software, they are considered real-world features. Mininet's baseline design consists of an OpenFlow kernel switch connected to two hosts and an OpenFlow controller. Mininet hosts can run Linux and file system commands. The command "iPerf," for example, parses bandwidth between a client and server, whilst the "topo" command or MiniEdit GUI platform is used to design custom virtual networks [31].

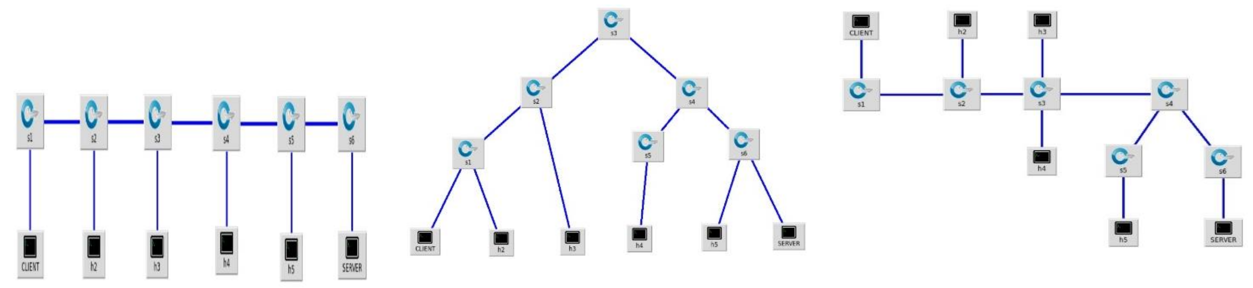

Review of the literature
Customization of network topology using Mininet emulator for SDN environment
Set up an ODL controller in an SDN environment.
Generate real-time data traffic from client host to server host using Wireshark analyzer
Performance evaluation
Fig. 5. Research methodology

Implementation of the controller in the SDN is the next significant step. Experiments were set up using the open-source controller Opendaylight (ODL) in the Mininet topology. ODL is a distributed multi-protocol controller system intended for highly available, adaptable, and scalable SDN implementations. It provides a framework service abstraction that allows users to create applications that are interoperable with a variety of hardware and Southbound protocols [47]. After the network topologies have been created, the Wireshark protocol analyzer is used to produce data traffic from the source to the destination node. Finally, the performance of the SDN network utilizing the ODL controller is measured by various metrics.

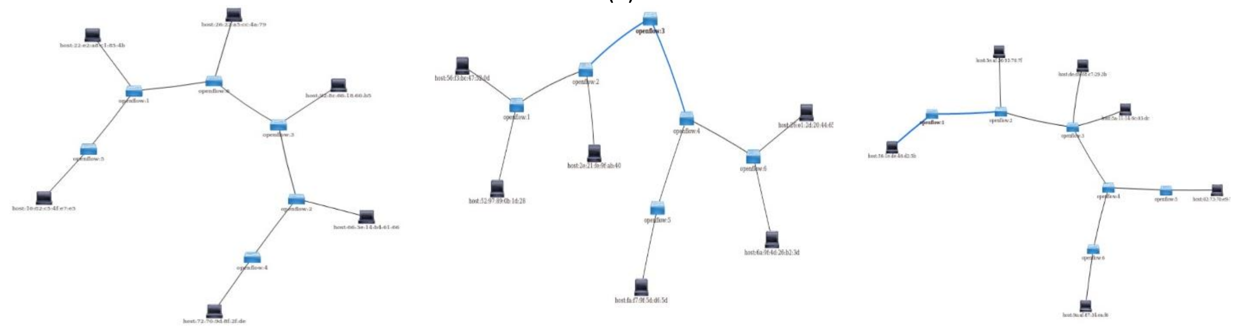
2.1 Experiment Setup

In this experiment, two virtual machines have been employed. One of these will run Mininet where the emulated network topology is located, and the second machine used to run the ODL controller. The two virtual machines must have connectivity to each other and execute services essential for the experiment are SSH, X Server software client, and Wireshark.

Three (3) custom networks are created consisting of a liner network, tree network, and hybrid network with a combination of linear and tree networks. Figure 6 exhibits the network topologies GUI implemented in MiniEdit and topologies presented through the web interfaces of the ODL controller. The topologies are made from software switches, named Open Vswitches (OVS) and OpenFlow version 1.3 is utilized as the Southbound protocol for control traffic. The forwarding path selection is based on the odl-L2switch feature of the ODL controller. The odl-L2switch was configured to operate reactively to a new flow [34]. The abstract view of the proposed SDN architecture is presented in Figure 7. The IP address range for all hosts and switches is 10.0.0.0/8, and the ODL controller is implemented in the control plane using port 6633 and the IP address 192.168.56.107/24.



Linear Topology Tree Topology Hybrid Topology
(a)



Linear Topology Tree Topology Hybrid Topology
(b)

Fig. 6. Network topologies (a) MiniEdit platform (b) OpenDaylight

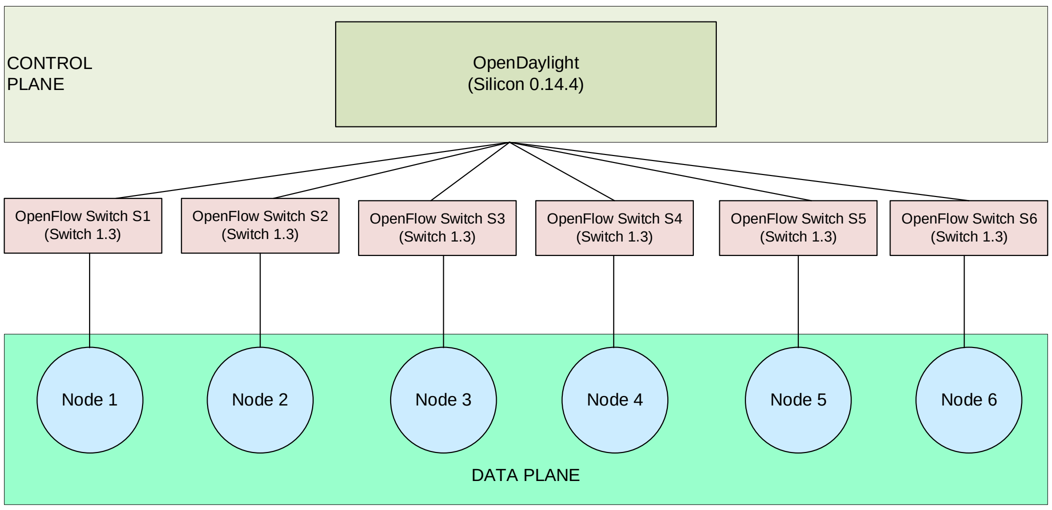

CONTROL PLANE
OpenDaylight (Silicon 0.14.4)
OpenFlow Switch S1 (Switch 1.3)
OpenFlow Switch S2 (Switch 1.3)
OpenFlow Switch S3 (Switch 1.3)
OpenFlow Switch S4 (Switch 1.3)
OpenFlow Switch S5 (Switch 1.3)
OpenFlow Switch S6 (Switch 1.3)
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
DATA PLANE

Fig. 7. Abstract view of the topology

To emulate network performance such as delay, throughput, packet loss, and bandwidth utilization under TCP data flow, tools like ping as well iPerf are employed. By implementing iPerf, one side runs in a "server" mode, listening for requests; the other end runs in "client" mode, sending data. iPerf can test in real time with any number of TCP packet size settings. Therefore, to study in-depth the performance of each network topologies, the size of a data packet for the traffic flow was configured differently with sizes 356 Kbytes, 675 Kbytes, 1090 Kbytes, 1550Kbytes, and 2020Kbyt. The data traffic is transmitted from the client to the server node using Wireshark. The performance of the SDN network utilizing the distributed controller is evaluated.

3. Results

The delay, throughput, and packet loss for linear, tree, and hybrid topologies are shown in Table 2. The time takes packets to travel from client to server is measured in delay. The controller is evaluated for the number of data successfully delivered per unit time during the throughput test [48]. The percentage of packets that fail to reach their destination is known as packet loss [31]. Referring to Table 2, the simulation result for the topology tree seemed to have the highest average delay and packet loss, as well as the lowest throughput average.

Table 2
Simulation result

Topologies	Min Delay (s)	Max Delay (s)	Average Delay (s)	Average Throughput (Kbps)	Average Packet Loss (%)
Linear	0.00001	0.00060	0.00004	219614	37
Tree	0.00003	0.00149	0.00009	111099	67
Hybrid	0.00021	0.00052	0.00003	201118	39

3.1 Network Delay

In a network, during the process of data communication, the delay also known as latency is defined as the total time taken for a complete message to arrive at the destination, starting with the time when the first bit of the message is sent out from the source and ending with the time when the last bit of the message is delivered at the destination. Figure 8 illustrates the delay comparison between three different topologies with different size of packet sizes.

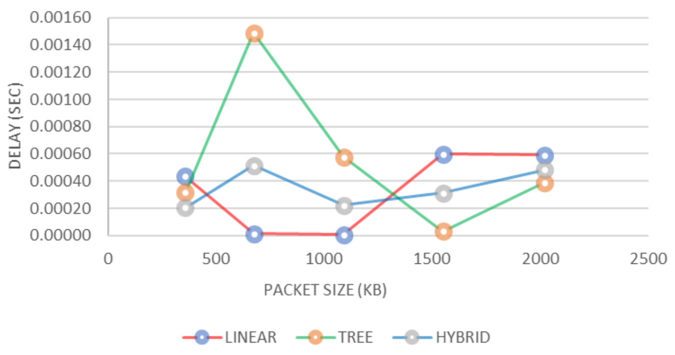


Fig. 8. Comparison of delay result

The tree topology has the largest delay in all three topologies, as seen in Figure 8. As shown in Table 2, tree topology has the longest delay when the packet size is small. Linear topologies have lower delay than other topologies, implying that tree topologies take longer to select a route and send a decision for newly arriving flows. The effect of the delay, however, is just transient and diminishes as the number of packet sizes grows. In a more complicated topology (hybrid topology), the network has less delay compared to linear and tree topology. In contrast, the delay is increased in linear topology when the packet size increased.

3.2 Throughput

The controller is examined in throughput mode tests to see how many packets it can process in a second. The amount of data transferred per time is used to calculate network throughput. The throughput evaluation findings in Figure 9 show that increasing the number of packet sizes has a minor impact on tree topology. This is because larger packet sizes generate congestion at the data layer, requiring more processing resources. The throughput performance of the linear topology is the best. However, a rise in the number of packet sizes has a significant impact.

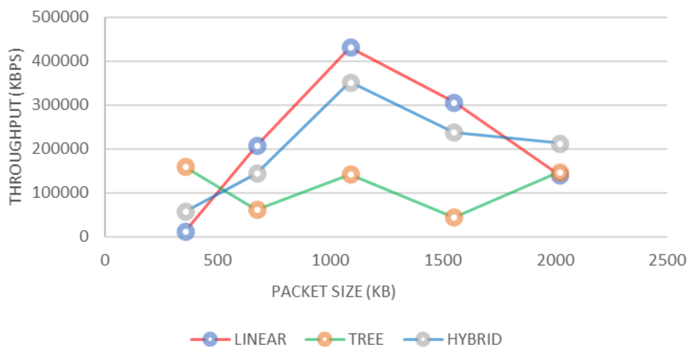


Fig. 9. Comparison of throughput result

3.3 Packet Loss

The number of packets that fail to reach their destination is referred to as packet loss. The packet loss ratio is calculated as a percentage of total packet loss divided by the total number of packets delivered. The comparison of packet loss results is shown in Figure 10.

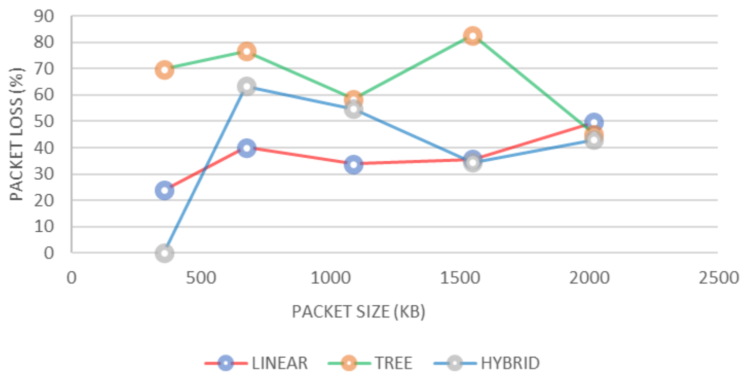


Fig. 10. Comparison of packet loss result

It is noticeable that hybrid topology exhibited no packet loss under small packet size. However, it is drastically high when packet size increased. There is no significant difference in packet loss ratio in a linear topology. Tree topology faced the highest packet loss rate even in the small size of the packet. The experimental results showed that the packet size had different effects on the packet loss rate of the TCP stream.

3.4 Bandwidth

Performance evaluation of bandwidth used in SDN networks can be accomplished using iPerf to simulate the TCP data flow. In TCP traffic, the source node sends a request packet TCP SYN to the destination node for the establishing of connection via the SDN switch. TCP examines the number of packets forwarded to the target host via a different number of processes. This test is regulated to measure the bandwidth of TCP traffic among the nodes. The consumption of bandwidth in transferring packets is also determined. The average bandwidth utilization is displayed in Figure 11. As demonstrated in Figure 11, higher average bandwidth utilization during small packet size is transmitted and somewhat reduced when packet size is bigger.

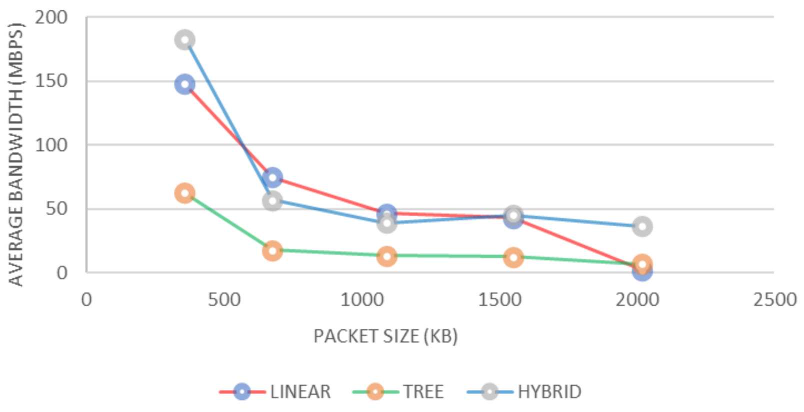


Fig. 11. Comparison of bandwidth average usage result

Packet size will affect bandwidth depending on the performance of the network sources utilized in the transfer. Each packet has a header that has the destination address for that packet. Every switch needs to look up and match that destination address to a flow table. That lookup takes a specified amount of time and hence a latency in sending the packet to the proper “port” on the switch. As the lookup time or latency is pretty much the same for small packets as well as large packets, then impact larger packets will have better bandwidth performance. Small packets will have more lookups per byte of payload [49].

4. Conclusions

The effectiveness of controllers directly guarantees the quality of service in SDN. Therefore, controller performance is one of the most significant design parameters. To assure good quality of service, the controller should be able to respond to packets in messages immediately. This means that the average delay and packet loss must be minimal, and throughput must be maximal. This study intends to analyse the performance of the distributed controller to explore if these controllers are ready for prime-time deployment. The proposed work provides the traffic analysis via performance evaluation on one of the well-known distributed controllers named OpenDaylight.

In this research, the results of the linear, tree, and hybrid topologies have been compared. The result shows that the controller's load increases as the network architecture more complicated. The performance of the SDN network thus becomes incompetent. The packet drop ratio rises as the delay increases. The delay in tree topology is high, making a high packet loss ratio and throughput low. We have also found that the throughput and transmission delay are much better for simple network topologies such as linear topology.

This report would be helpful for all the researchers working in SDN and controller traffic evaluation. The experimentations revealed that the distributed SDN controller may potentially be considered one of the powerful controllers for traffic engineering. The study work revealed beneficial findings for various performance indicators in the SDN environment using the distributed controller. It gives the traffic analysis via performance evaluation of the distributed controller in the SDN environment to optimize the consumption of resources for the enhanced performance of the network, and management of data traffic in the network.

Acknowledgement
This paper is a preliminary study of the first author's ongoing research work under the main supervision of Dr. Mohd Nazri Mohd Warip, Associate Professor, University Malaysia Perlis for her Ph.D. program. We would like to thank the reviewers of this manuscript for their valuable comments throughout the process.

References
[1] Casado, Martin, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. "Ethane: Taking control of the enterprise." *ACM SIGCOMM computer communication review* 37, no. 4 (2007): 1-12. https://doi.org/10.1145/1282427.1282382

[2] Blenk, Andreas, Arsany Basta, Martin Reisslein, and Wolfgang Kellerer. "Survey on network virtualization hypervisors for software defined networking." *IEEE Communications Surveys & Tutorials* 18, no. 1 (2015): 655-685. https://doi.org/10.1109/COMST.2015.2489183

[3] Tanyingyong, Voravit, Markus Hidell, and Peter Sjödin. "Improving pc-based openflow switching performance." In *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pp. 1-2. 2010. https://doi.org/10.1145/1872007.1872023

[4] Wijethilaka, Shalitha, and Madhusanka Liyanage. "Survey on network slicing for Internet of Things realization in 5G networks." *IEEE Communications Surveys & Tutorials* 23, no. 2 (2021): 957-994. https://doi.org/10.1109/COMST.2021.3067807

[5] Ahmad, Suhail, and Ajaz Hussain Mir. "Scalability, consistency, reliability and security in SDN controllers: a survey of diverse SDN controllers." *Journal of Network and Systems Management* 29 (2021): 1-59. https://doi.org/10.1007/s10922-020-09575-4

[6] Benzekki, Kamal, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. "Software-defined networking (SDN): a survey." *Security and communication networks* 9, no. 18 (2016): 5803-5833. https://doi.org/10.1002/sec.1737

[7] Josbert, Nteziriza Nkerabahizi, Wang Ping, Min Wei, and Ahsan Rafiq. "Solution for Industrial Networks: Resilience-based SDN Technology." In *2021 IEEE 2nd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*, pp. 392-400. IEEE, 2021. https://doi.org/10.1109/ICBAIE52039.2021.9390019

[8] Wang, Tao, Fangming Liu, Jian Guo, and Hong Xu. "Dynamic SDN controller assignment in data center networks: Stable matching with transfers." In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pp. 1-9. IEEE, 2016. https://doi.org/10.1109/INFOCOM.2016.7524357

[9] Nisar, Kashif, Emilia Rosa Jimson, Mohd Hanafi Ahmad Hijazi, Ian Welch, Rosilah Hassan, Azana Hafizah Mohd Aman, Ali Hassan Sodhro, Sandeep Pirbhulal, and Sohrab Khan. "A survey on the architecture, application, and security of software defined networking: Challenges and open issues." *Internet of Things* 12 (2020): 100289. https://doi.org/10.1016/j.iot.2020.100289

[10] Rajoriya, Manisha Kumari, and Chandra Prakash Gupta. "A Taxonomy on Distributed Controllers in Software Defined Networking." In *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, pp. 120-126. IEEE, 2021. https://doi.org/10.1109/ICCMC51019.2021.9418048

[11] Haji, Saad H., S. R. Zeebaree, Rezgar Hasan Saeed, Siddeeq Y. Ameen, Hanan M. Shukur, Naaman Omar, Mohammed AM Sadeeq, Zainab Salih Ageed, Ibrahim Mahmood Ibrahim, and Hajar Maseeh Yasin. "Comparison of software defined networking with traditional networking." *Asian Journal of Research in Computer Science* 9, no. 2 (2021): 1-18. https://doi.org/10.9734/ajrcos/2021/v9i230216

[12] Naseer, Muhammd Zeshan. "Modeling Control Traffic in Distributed Software Defined Networks." (2016).

[13] Fazea, Yousef, and Fathey Mohammed. "Software defined networking based information centric networking: An overview of approaches and challenges." In *2021 International Congress of Advanced Technology and Engineering (ICOTEN)*, pp. 1-8. IEEE, 2021. https://doi.org/10.1109/ICOTEN52080.2021.9493541

[14] Shailly, Ms. "A critical review based on Fault Tolerance in Software Defined Networks." *Turkish Journal of Computer and Mathematics Education (TURCOMAT)* 12, no. 2 (2021): 456-461. https://doi.org/10.17762/turcomat.v12i2.849

[15] Awais, Muhammad, Muhammad Asif, Maaz Bin Ahmad, Toqeer Mahmood, and Sundus Munir. "Comparative Analysis of Traditional and Software Defined Networks." In *2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC)*, pp. 1-6. IEEE, 2021. https://doi.org/10.1109/MAJICC53071.2021.9526236

[16] Perera, Kosala, Udesh Gunarathne, Binal Chathuranga, Chamika Ramanayake, and Ajith Pasqual. "Hybrid Software Defined Networking Controller." In *DCNET*, pp. 77-84. 2017. https://doi.org/10.5220/0006423800770084

[17] Liu, Jiyang, Liang Zhu, Weiqiang Sun, and Weisheng Hu. "Scalable application-aware resource management in software defined networking." In *2015 17th International Conference on Transparent Optical Networks (ICTON)*, pp. 1-5. IEEE, 2015. https://doi.org/10.1109/ICTON.2015.7193522

[18] Hakiri, Akram, Aniruddha Gokhale, Pascal Berthou, Douglas C. Schmidt, and Thierry Gayraud. "Software-defined networking: Challenges and research opportunities for future internet." *Computer Networks* 75 (2014): 453-471. https://doi.org/10.1016/j.comnet.2014.10.015

[19] Killi, Bala Prakasa Rao, and Seela Veerabhadreswara Rao. "Controller placement with planning for failures in software defined networks." In *2016 IEEE international conference on advanced networks and telecommunications systems (ANTS)*, pp. 1-6. IEEE, 2016. https://doi.org/10.1109/ANTS.2016.7947795

[20] McKeown, Nick, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. "OpenFlow: enabling innovation in campus networks." *ACM SIGCOMM computer communication review* 38, no. 2 (2008): 69-74. https://doi.org/10.1145/1355734.1355746

[21] Costa, Leonardo C., Alex B. Vieira, Erik de Britto e Silva, Daniel F. Macedo, Luiz FM Vieira, Marcos AM Vieira, Manoel da Rocha Miranda Junior et al. "OpenFlow data planes performance evaluation." *Performance Evaluation* 147 (2021): 102194. https://doi.org/10.1016/j.peva.2021.102194

[22] Mamushiane, Lusani, and Themba Shozi. "A QoS-based evaluation of SDN controllers: ONOS and OpenDayLight." In *2021 IST-Africa Conference (IST-Africa)*, pp. 1-10. IEEE, 2021.

[23] Sarmiento, David Espinel, Adrien Lebre, Lucas Nussbaum, and Abdelhadi Chari. "Decentralized SDN control plane for a distributed cloud-edge infrastructure: A survey." *IEEE Communications Surveys & Tutorials* 23, no. 1 (2021): 256-281. https://doi.org/10.1109/COMST.2021.3050297

[24] Isong, Bassey, Reorapetse Ramoliti Samuel Molose, Adnan M. Abu-Mahfouz, and Nosipho Dladlu. "Comprehensive review of SDN controller placement strategies." *IEEE Access* 8 (2020): 170070-170092. https://doi.org/10.1109/ACCESS.2020.3023974

[25] Abdelaziz, Ahmed, Ang Tan Fong, Abdullah Gani, Usman Garba, Suleman Khan, Adnan Akhunzada, Hamid Talebian, and Kim-Kwang Raymond Choo. "Distributed controller clustering in software defined networks." *PloS one* 12, no. 4 (2017): e0174715. https://doi.org/10.1371/journal.pone.0174715

[26] Bannour, Fetia, Sami Souihi, and Abdelhamid Mellouk. "Distributed SDN control: Survey, taxonomy, and challenges." *IEEE Communications Surveys & Tutorials* 20, no. 1 (2017): 333-354. https://doi.org/10.1109/COMST.2017.2782482

[27] Koponen, Teemu, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan et al. "Onix: A distributed control platform for large-scale production networks." In *OSDI*, vol. 10, no. 1, p. 6. 2010.
[28] Berde, Pankaj, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz et al. "ONOS: towards an open, distributed SDN OS." In *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1-6. 2014.
[29] Mamushiane, Lusani, Albert Lysko, and Sabelo Dlamini. "A comparative evaluation of the performance of popular SDN controllers." In *2018 Wireless Days (WD)*, pp. 54-59. IEEE, 2018. https://doi.org/10.1109/WD.2018.8361694
[30] Bhardwaj, Shanu, and Surya Narayan Panda. "Performance evaluation using ryu sdn controller in software-defined networking environment." *Wireless Personal Communications* 122, no. 1 (2022): 701-723. https://doi.org/10.1007/s11277-021-08920-3
[31] Rowshanrad, Shiva, Vajihe Abdi, and Manijeh Keshtgari. "Performance evaluation of SDN controllers: Floodlight and OpenDaylight." *IIUM Engineering Journal* 17, no. 2 (2016): 47-57. https://doi.org/10.31436/iiumej.v17i2.615
[32] Eftimie, Alexandra, and Eugen Borcoci. "SDN controller implementation using OpenDaylight: experiments." In *2020 13th International Conference on Communications (COMM)*, pp. 477-481. IEEE, 2020. https://doi.org/10.1109/COMM48946.2020.9142044
[33] Abdullah, Mahmood Z., Nasir A. Al-Awad, and Fatima W. Hussein. "Performance Comparison and Evaluation of Different Software Defined Networks Controllers." *International Journal of Computing and Network Technology* 6, no. 2 (2018).
[34] Ghalwash, Haitham, and Chun-Hsi Huang. "A QoS framework for SDN-based networks." In *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, pp. 98-105. IEEE, 2018. https://doi.org/10.1109/CIC.2018.00024
[35] Vilchez, José Manuel Sanchez, and David Espinel Sarmiento. "Fault tolerance comparison of onos and opendaylight sdn controllers." In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pp. 277-282. IEEE, 2018. https://doi.org/10.1109/NETSOFT.2018.8460099
[36] Hassan, Ahmed Hassan M., Ahmed M. Alhassan, and Fathia Izzeldean. "Performance evaluation of sdn controllers in ofnet emulation environment." In *2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, pp. 1-6. IEEE, 2019. https://doi.org/10.1109/ICCCEEE46830.2019.9071007
[37] Islam, Md Tariqul, Nazrul Islam, and Md Al Refat. "Node to node performance evaluation through RYU SDN controller." *Wireless Personal Communications* 112 (2020): 555-570. https://doi.org/10.1007/s11277-020-07060-4
[38] Tootoonchian, Amin, and Yashar Ganjali. "Hyperflow: A distributed control plane for openflow." In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, vol. 3, pp. 10-5555. 2010.
[39] Ahmed, Hatim Gasmelseed, and R. Ramalakshmi. "Performance analysis of centralized and distributed SDN controllers for load balancing application." In *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 758-764. IEEE, 2018. https://doi.org/10.1109/ICOEI.2018.8553946
[40] Kazemian, Mohammad Mahdi, and Meghdad Mirabi. "Controller placement in software defined networks using multi-objective antlion algorithm." *The Journal of Supercomputing* (2022): 1-24. https://doi.org/10.1007/s11227-021-04109-4
[41] Shalimov, Alexander, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov, and Ruslan Smeliansky. "Advanced study of SDN/OpenFlow controllers." In *Proceedings of the 9th central & eastern european software engineering conference in russia*, pp. 1-6. 2013. https://doi.org/10.1145/2556610.2556621
[42] Gong, Yili, Wei Huang, Wenjie Wang, and Yingchun Lei. "A survey on software defined networking and its applications." *Frontiers of Computer Science* 9 (2015): 827-845. https://doi.org/10.1007/s11704-015-3448-z
[43] Khatri, Vikramajeet. "Analysis of OpenFlow protocol in local area networks." Master's thesis, 2013.
[44] Specification, OpenFlow Switch. "Version 1.0. 0 (Wire Protocol 0x01)." *Open Networking Foundation* (2009).
[45] Klein, Dominik, and Michael Jarschel. "An OpenFlow extension for the OMNeT++ INET framework." In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, pp. 322-329. 2013. https://doi.org/10.4108/icst.simutools.2013.251722
[46] Banjar, Ameen, Pakawat Pupatwibul, and Robin Braun. "Comparison of TCP/IP routing versus openflow table and implementation of intelligent computational model to provide autonomous behavior." *Computational Intelligence and Efficiency in Engineering Systems* (2015): 121-142. https://doi.org/10.1007/978-3-319-15720-7_9
[47] Parhandhito, Nasikh, Ridha Muldina Negara, and Favian Dewanta. "Comparison of High Availability Performance on OpenDaylight with Corosync Pacemaker and OpenDaylight SDN Controller Platform Clustering." In *2020 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)*, pp. 66-71. IEEE, 2021. https://doi.org/10.1109/IoTaIS50849.2021.9359696

[48] Khattak, Zuhran Khan, Muhammad Awais, and Adnan Iqbal. "Performance evaluation of OpenDaylight SDN controller." In *2014 20th IEEE international conference on parallel and distributed systems (ICPADS)*, pp. 671-676. IEEE, 2014. https://doi.org/10.1109/PADSW.2014.7097868

[49] Banchuen, Teerawut, Kiattikun Kawila, and Kultida Rojviboonchai. "An SDN framework for video conference in inter-domain network." In *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pp. 600-605. IEEE, 2018. https://doi.org/10.23919/ICACT.2018.8323848