EAI Endorsed Transactions
on Internet of Things
Research Article EAI.EU

POX and RYU Controller Performance Analysis on Software Defined Network

Naimullah Naim[1], Muhammad Imad[1], Muhammad Abul Hassan [2,*], Muhammad Bilal Afzal [3], Shabir Khan [1] and Amir Ullah Khan [3]

[1]Department of Computing and Technology, Abasyn University Peshawar Pakistan
[2]Department of Information Engineering and Computer Science, University of Trento Italy
[3]Pakistan Council of Scientific and Industrial Research, Peshawar, Pakistan

Abstract

From the last decades different types of network schemes are pitched to enhance the user performance. Software Defined Networks (SDN) is also considered as important factor for different network schemes and its proper administration or management. Due to major deployment in today's networking era SDN are further sub divided in to commercial and open-source controllers. Commercial and open-source controllers are utilized in different type of businesses. According to our knowledge considerable amount of literature is available on these controllers but did not provide or analyse performance of these controllers on different network parameters. This paper evaluates and compares the performance of two well-known SDN open-source controllers POX and RYU with two performance assessments. The first assessment is the implementation of optimal path by using Dijkstra's algorithm from source to destination. Second assessment is the creation of a custom topology in our desired tool (MiniNet emulator). Then, the performance in terms of QoS parameters such as Jitter, throughput, packet loss, and packet delivery ratio are computed by two end hosts in each network. After the assessments, the performance of POX are optimal as compare to the RYU and best suited to be deployed in any scenario.

Keywords: SDN, POX, RYU, Packet Loss, Packet Delivery Ratio, Jitter, Throughput, MiniNet

Received on 30 October 2023, accepted on 20 July 2023, published on 05 September 2023

doi: 10.4108/eetiot.v9i3.2821

*Corresponding author. Email: muhammadabul.hassan@unitn.it

1. Introduction

Tremendous volumes of data have contributed to massive data centres. Large-scale Processing and storage across these data centres have resulted in new market models. More complex computer networks have also increased the complexity of computing and storage. Well-developed conventional network infrastructure is a good candidate for domain networking for network engineers. They have limited access to customize new network policies by converting high-level dynamic firmware into a new network policy. Low-level configuration commands that require manual intervention. Because of this limitation has become extremely difficult for the internet to evolve from advanced protocols and efficient performance. This challenge could result in network management and control capacities confinement, requiring manual improvements to lead to error-prone tasks. This challenge is referred to as 'Internet ossification' by researchers and network engineers [1]. Thus, the current Internet situation is limited to addressing emerging technological trends in networking [2] [36] [37].

To facilitate new network developments, the concept of programmable networks has been proposed [3]. Software-Defined Networks (SDN) are a relatively new programmable network organization technique. Software-Defined Networking (SDN) facilitates control and data plane isolation [4]. This paradigm shift in networking architecture promises to solve several large-scale networks, particularly data centers. SDN has three essential features: first, it can isolate core networking devices' data and control functions using a well-defined application programming interface (API) (generally referred to as Southbound APIs) [5]. Second, SDN offers a single control plane feature, such that a specific software application can conveniently manage different elements of the data plane (usually referred to as Network Operating System) (NOS) [6]. Thirdly, SDN facilitates hierarchical controls, and this architecture offers a global vision of an entire underlying physical network to network engineers or network operators who can improve globally. A well-defined open API (usually called Northbound APIs between control planes and network running applications) enables innovation and efficient network control. Different types of SDN controllers provide various services with different criteria for quality of service (QoS) and scalability. The controller can also alter the network's configuration and facilities at runtime [7]. SDN controllers are mainly implemented in large-scale environments where performance is a key concern. Therefore, an extensive assessment framework is required to select the well-liked controller in each scenario based on the quality of service (QoS) requirement, custom topology, routing protocols, and workload, impacting the controller performance tremendously [39] [40].

This research paper has analysed two well-known open-source SDN controllers with the help of MiniNet command-line interface tools for simulation. We implemented Dijkstra's shortest route algorithm in the POX and RYU SDN controllers and compared these controllers' performance based on various metrics like delay, throughput, packet delivery ratio, jitter, and bitrate to identify the best controllers.

This research article is organized such that the next section defines different terms and definition related to SDN. In section III, we study some related work done in software-defined networking controller assessment.

1.1 Software-Defined Network Architecture

The primary idea of the software-defined network is elementary. The SDN architecture has three key components described in the below section.

The first is the management plan, which consists of a collection of network programs that handle the software-defined network control logic. SDN-enabled networks use programmability rather than command-line interfaces to offer simplicity and ease in deploying new software and facilities, including routing, policy enforcement, load balancing, or custom service provider applications. The network's automation and orchestration are also possible through the current API [8].

The control plane is the second and most clever and critical aspect of the fundamental SDN architecture. This layer contains a controller that manages the packet transmission through the Southbound interface, forwarding various rules and policies to the infrastructure layer [8].

The infrastructure layer, also known as the data plane, is the third level, and it represents network communication devices such as switches, routers, and load balancers.). The southbound APIs link to the control plane by gathering policies, forwarding rules, and applying them to the relevant equipment [9].

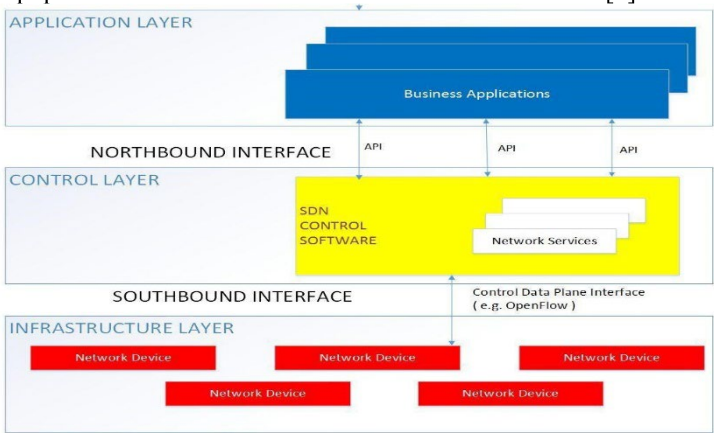

Figure 1: SDN Architecture

1.2 SDN Controller

In a software-defined network, SDN controllers' function as the network's "brain." It is the network's operating system (os). It is an intermediate SDN architecture layer, as seen in Figure.1. It is a strategic control point that uses northbound APIs to handle networking appliances or business applications and uses southbound APIs to transfer control information to the underlying routers or switches. An SDN controller (also referred to as an OpenFlow controller) uses the OpenFlow Protocol (OFP) in the SDN architecture to configure the underlying core network equipment and select the right route for data traffic forwarding. Since the control plane is typically a centrally controllable software program, it can handle network traffic dynamically. Several open-source SDN controller programs (NOS) are currently used to deploy the architecture; these are Floodlight [10], POX [11], Trema [12], Beacon [13], Ryu [14], Maestro [15], etc.

1.3 POX Controller

The POX is an SDN controller built on a python inherited from the NOX controller. The POX controller may be easily implemented using the OpenFlow protocol, which is the de facto communication protocol between controllers and switches. Using the POX control, you can run multiple programs, such as a switch, hub, load balancer, and firewall. The POX controller and switches communicate through a communication protocol such as OpenFlow. POX is built into MiniNet and is also available for download from GitHub [5, 16].

1.4 RYU Controller

RYU is an SDN networking framework built on components. Ryu provides well-defined API platform modules that allow developers to create new network management and control applications. Ryu follows multiple network interface control protocols, such as Netconf, OpenFlow, OF-config, etc. Ryu completely supports 1.0, 1.2, 1.3, 1.4, 1.5, and Nicira Plugins for OpenFlow. The complete code is available free of charge under the Apache 2.0 license [17].

2. Literature Review

The POX and Floodlight controller's performance was compared based on delay and throughput. Experiments were carried out in MiniNet for various network topologies. This investigation was also narrowed to two controllers, rather than any other java or python developed controllers; just a few network variables were computed [18].

The performance compression of ONOS, Open Daylight, POX, and RYU controllers was done based on bandwidth and end-to-end delay parameters. A fixed four-level tree topology with 16 hosts was employed to test the controllers' performance. The analysis concluded that RYU had the least end-to-end delay of these four controllers, and ONOS had the maximum bandwidth. Depending on the aims or needs of the outcome, the best-qualified controllers were frequently chosen. The POX controller was adopted as the most acceptable set of configuration simplicity as the highest priority. Still, performance is not equivalent to Open Daylight, RYU, and ONOS controllers [19].

They differentiated the performance of well common OpenFlow controllers such as POX, NOX, RYU, FloodLight, and OpenFlow reference controllers based on their packet handling capability by changing the packet size and coming pattern in the IP traffic flows. The distributed internet traffic flow generator tool has been used to compute throughput, jitter, packet loss, and delay. Their experimentation outcomes display that Floodlight has better throughput and less delay when differentiated from other controllers [20].

The performance of five controllers (POX, ONOS, Open Daylight, RYU, libfluid) is evaluated using the linear topology in a MiniNet emulator with different switches. Ping and Iperf commands perform the performance assessment. This paper provides a new contribution to measuring and comparing the delay in and throughput responses of the five controllers while increasing the load on the linear topology and stopping responding to the network load (number of switches). Finally, the findings demonstrate that libfluid provides the best throughput performance, and POX offers the best delay performance [21].

The performance of SDN controllers such as Floodlight, Beacon, Open-MUL, and Open-IRIS was assessed. The assessment used three types of traffic: TCP, ICMP, and UDP using Iperf and Ping commands. A method to improve the network's performance by using the QoS technique was the Floodlight controller [22].

The exploration and comparison analysis of POX, RYU, and Open Daylight on network performance parameters such as packet loss, throughput, and jitter are done in this article. Although, using an open-source simulation tool called MiniNet to create different topologies. The data's assessment clearly shows that Ryu has higher throughput relative to Floodlight in all topologies. In all topologies, except Torus, Ryu performs best in cases of latency and jitter [23].

This paper discusses and analyzes POX characteristics, and Floodlight controllers and contrasts their performance parameters to select the popularly known SDN open-source controllers. The parameters are evaluated in various topologies. When varying the number of measurements and the data rate, it is found that, in terms of the packet transmitting time, Floodlight results are much quicker (31 times faster) in all topologies when compared to POX [24].

In terms of available delay and packet forwarding capacity, the implementation of two well-known SDN controllers, Open Daylight and Floodlight, was compared. The simulation modelling was based on a network flow, and the shortest or lowest path technique was also applied. The load-balancing algorithm's introduction has made it possible to optimize the Software-Defined Networking's QoS activity, reduce response times, and optimally spread the load from the connections. Consequently, the proposed load-balancing algorithm dramatically improves the performance of the Open Daylight-based controller in terms of QoS given [25].

3. Proposed Approach

This section describes SDN open-source controllers, Dijkstra's shortest route algorithm, and the simulation tool used for experimentation.

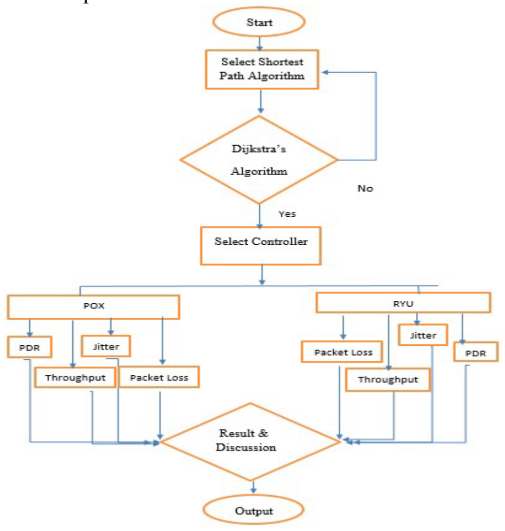

Figure 2: Flow Chart

3.1Dijkstra'sAlgorithm

The majority of the issues that contemporary networks that prohibit adequate load balancing are connected to the routing algorithm itself. The present routing technique is based on the shortest path algorithm. Each packet seeks a route that can cover the fewest number of hops, and this is the same for all packets, even though alternative routes are higher but considerably faster. To simulate the traffic behavior and evaluate the network performance based on the regular algorithm Dijkstra's shortest routing in SDN [26].

The Dijkstra algorithm is called the shortest single-source route. It calculates the length of the shortest route from the source to each of the vertices remaining in the graph. The shortest route problem for a single source can be defined as follows: Let G= be a weighted graph directed with V having the vertices set. The special vertex in V, where s is the source and can be used for any edge in E, Edge Cost (e) is the length of edge e. It should be non-negative for all weights in the graph.

Algorithm: DIG-RYU-POX
Problem: shortest route finding
Input: Number of (k), all possible paths, S, D
Output: The best path from S to D using the controller

```
1   Start
2   Connectivity matrix G (I, J)
3   Network matrix C (S, D)
4   INF= all possible path
5   IF (Failure>0)
6           Remove spam node between S and D
7           G (I, j) = possible, matrix=0
8   End if
9   Dijkstra (K, S, D, G, C)
10  While (n<k)
11          Dijkstra (S, D, G, C)
12          Save all possible path
13          Subtract connectivity matrix
14          n++

15  End while
16  For n pass to controllers
17          For possible path in all paths
18                  For S, D, all paths
19                          Adjacency [S], [D]
20                  End For
21          End For
22  End For
```

3.2 Experimental Setup

The POX and RYU controllers are run in a virtual environment created by VMware Workstation Pro. Ubuntu 18.04 is installed in Virtual Box to construct the simulation's operating environment. The network simulation employs a MiniNet simulator, which can create the network environment and the accompanying simulation within the scope of the virtual environment. For comparison, we build a Python-based RYU, POX controller. The available throughput and PDR of ICMP queries are measured for a situation in which the shortest path technique is used [27].

3.2.1 MiniNet

MiniNet is the open-source network emulator for the SDN in a virtual environment to simulate an extensive network. The most important reason for using the MiniNet is supporting the Open Flow Protocol, a better environment to simulate software Defined Networking controller and test custom network topologies.

3.2.2 Iperf

The iperf network testing program is widely used for measuring bandwidth and network connections. The program can create TCP and UDP data streams as well as assess network throughput, bandwidth, and network quality for these streams [28]. The iperf utility may assess uni-directional or bi-directional throughput between the two end-hosts and perform client and server functions. It enhances the tuning of multiple buffers, protocols, and timing parameters. It calculates the failure of packets, latency jitters, etc., and supports several simultaneous links.

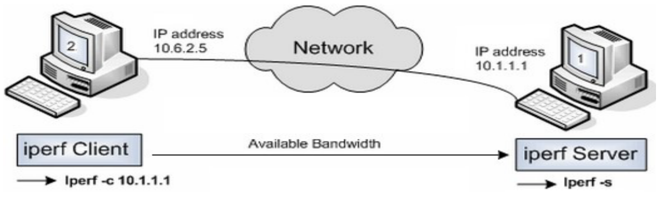

Figure 3: Iperf Flow

4. Results and Discussion

In this chapter, we simulate and show the simulation outcomes in graphs. We evaluate the network's performance with a custom network topology to measure the algorithm's nature with the simulation results. The parameters used to evaluate the network's performance were throughput, packet delivery ratio, jitter, and packet loss. The parameters have been graphically displayed concerning the number of times.

4.1 Comparison Parameters

Comparison parameters played a vital role in evaluating routing protocol algorithms in different network scenarios.

In this research study network performance evaluation of POX and RYU SDN controllers are carried out in terms of:

- i. Throughput
- ii. Packet Delivery Ratio
- iii. Jitter iv.
- iv. Packet Loss

4.2 Performance Evaluation

This section describes the results obtained by RYU and POX controllers when the shortest path algorithms mentioned in the preceding section are used. It is worth noting that both SDN controllers operate on the same customized network topology.

4.2.1 Throughput

The amount of data transferred in a unit of time is measured in bites per second (kbps). The throughput can be calculated mathematically by processing several bits per unit of time. Average throughput is calculated according to the following formula [29]:

$$Throughput = (\Sigma\ received\ packet\ size)/time\ (Gbps) \qquad (1)$$

The iperf utility is a well-known network testing tool for measuring bandwidth and network connections. The program can create TCP and UDP data streams and calculate network performance for these streams. The iperf tool performs both server and client features and can measure uni-directionally or bidirectionally throughput between the two end-hosts. The network throughput is computed in bits per second or data packets per second using the iperf real-time technique between source and destination nodes with and without Dijkstra's algorithm implementation and various topologies in the RYU SDN controller. The iperf program was used to assess the controller throughput performance by creating a TCP node-to-node connection where one node acts as the server and the other as the client. To figure out TCP throughput, iperf has carried out in 10 s on the client side, and data have been obtained every 1 s on the server side.

4.2.2 RYU Controller Throughput

The throughput result between source and destination nodes of the RYU SDN controller in Dijkstra's algorithm and normal flow is measured and tabulated in Table 1 and is displayed graphically in Figure. 5.1. The throughput graph helps in discovering end-to-end performance.

Table 1: RYU Controller Throughput

S. No	Time Interval(sec)	Throughput (Gbp/S)	
		Dijkstra Algorithm	Normal Flow
1	0.0-1.0	7.16	5.01
2	1.0-2.0	7.2	5.56
3	2.0-3.0	7.14	5.56
4	3.0-4.0	7.21	5.3
5	4.0-5.0	7.16	4.62
6	5.0-6.0	7.16	4.76
7	6.0-7.0	6.81	4.39
8	7.0-8.0	7.34	4.69
9	8.0-9.0	7.23	4.73
10	9.0-10.0	7.25	5.17
Average Throughput		7.166	4.979

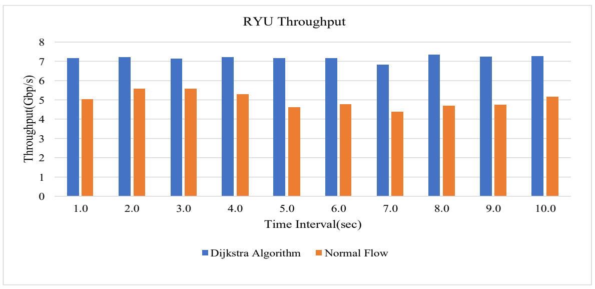

Figure 4: Throughput of SDN RYU Controller.

4.2.3 POX Controller Throughput

The throughput test between source and destination nodes of POX SDN controller in Dijkstra's algorithm and normal flow is measured and tabulated in Table 2 and is displayed graphically in Figure. 5. The throughput graph helps in discovering end-to-end performance.

Table2: POX Controller Throughput

S. No	Time Interval(sec)	Throughput (Gbp/S)	
		Dijkstra's Algorithm	Normal Flow
1	0.0-1.0	13	8.22
2	1.0-2.0	13.3	7.54
3	2.0-3.0	12.9	8.31
4	3.0-4.0	13	8.02
5	4.0-5.0	12.7	8.08
6	5.0-6.0	11.9	8.12
7	6.0-7.0	12.9	8.12
8	7.0-8.0	12.6	8.02
9	8.0-9.0	13	7.99
10	9.0-10.0	12.9	7.49
Average Throughput		12.82	7.991

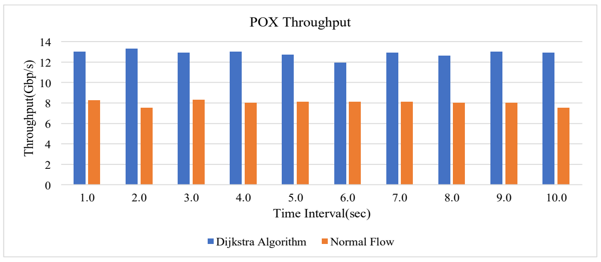

Figure 5: Throughput of SDN POX Controller

4.2.4 POX and RYU Controller Throughput

The throughput test between source and destination nodes of the RYU SDN controller is measured and tabulated in Table 3 and is displayed graphically in Figure. 6. The throughput graph helps in discovering end-to-end performance.

Table 3: RYU & POX Controller Throughput

S.No	Time Interval(sec)	RYU-Throughput (Gbp/S)		POX-Throughput (Gbp/S)	
		Dijkstra's Algorithm	Normal Flow	Dijkstra's Algorithm	Normal Flow
1	0.0-1.0	7.16	5.01	13	8.22
2	1.0-2.0	7.2	5.56	13.3	7.54
3	2.0-3.0	7.14	5.56	12.9	8.31
4	3.0-4.0	7.21	5.3	13	8.02
5	4.0-5.0	7.16	4.62	12.7	8.08
6	5.0-6.0	7.16	4.76	11.9	8.12
7	6.0-7.0	6.81	4.39	12.9	8.12
8	7.0-8.0	7.34	4.69	12.6	8.02
9	8.0-9.0	7.23	4.73	13	7.99
10	9.0-10.0	7.25	5.17	12.9	7.49
Average Throughput		7.166	4.979	12.82	7.991

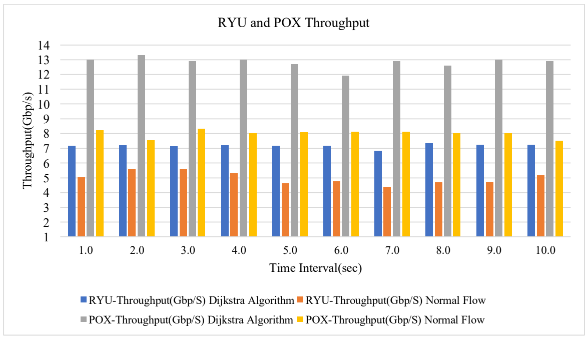

Figure 6: Throughput of RYU vs. POX Controllers.

The results of the observations are displayed; the test was performed on the open-source controllers RYU and POX SDN. Figures 4 to 6 display the throughput values RYU and POX for each controller in (Gbp/s). Figure 6 shows a comparison of the average network throughput of various controllers. Consequently, when compared to RYU, a controller across customized and conventional network topologies, the POX controller has the greatest throughput value. When evaluating the effects of network overload on various numbers of switches, the POX controller outperforms RYU. In comparison, the RYU controller demonstrates the lowest throughput value.

4.2.5 Packet Delivery Ratio

The packet delivery ratio (PDR) is s major indicator for evaluating the efficiency of a routing mechanism in any network. The Packet Delivery Ratio is an essential characteristic for measuring the performance of a routing system in any network. The protocol's performance is determined by the simulation settings chosen. The packet delivery ratio is determined by dividing the total number of data packets arriving at destinations by the total number of data packets transmitted from sources. When there is a high PDR, performance improves. In this research study packet received ratio is derived from the following formula [30].

$$Packet\ Delivery\ Ratio = 100 * \frac{No.of\ packets\ Delivered}{Total\ packets\ sent} \quad (2)$$

4.2.6 RYU Controller PDR (%)

Figure 5.4. illustrate the packet delivery ratio (PDR) of the RYU controller in Dijkstra's and normal flow. The X-axis represents the number of times for each experiment, and the y-axis represents Packet Delivery Ratio (%).

Table 4: RYU Controller PDR%

S.No	Time Interval (sec)	RYU PDR in Dijkstra's				RYU PDR in Normal Flow			
		Total Packet	Packet Lost	Received	PDR(%)	Total Packet	Packet Lost	Received	PDR(%)
1	0.0-1.0	89208	1514	87694	98.30	43647	1566	42081	96.41
2	1.0-2.0	89166	557	88609	99.37	20660	321	20339	98.44
3	2.0-3.0	89145	315	88830	99.64	38220	47	38173	99.87
4	3.0-4.0	89185	406	88779	99.54	41521	365	41156	99.12
5	4.0-5.0	89156	251	88905	99.71	39176	646	38530	98.35
6	5.0-6.0	89173	327	88846	99.63	36327	965	35362	97.34
7	6.0-7.0	89164	178	88986	99.80	43565	315	43250	99.27
8	7.0-8.0	88525	72	88453	99.91	43839	834	43005	98.09
9	8.0-9.0	89797	207	89590	99.76	45491	434	45057	99.04
10	9.0-10.0	89130	146	88984	99.83	43320	278	43042	99.35
Average PDR		89164.9	397.3	88767.6	99.54	39576.6	577.1	38999.5	98.52

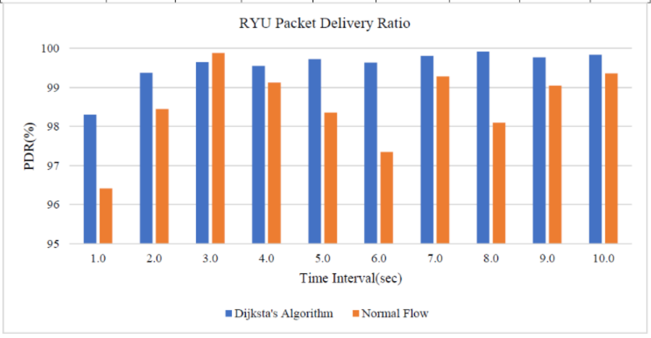

Figure 7: RYU Controller Packet Delivery Ratio

4.2.7 POX Controller PDR (%)

Figure 8, illustrates the packet delivery ratio (PDR) of the POX controller in Dijkstra's algorithm and normal flow. The X-axis represents the number of times for each experiment, and the y-axis represents Packet Delivery Ratio (%).

Table 5: POX Controller PDR (%)

S. No	Time Interval sec)	POX-PDR in Dijkstra's				POX-PDR in Normal Flow			
		Total Packet	Packet Lost	Received	PDR (%)	Total Packet	Packet Lost	Received	PDR (%)
1	0.0-1.0	450936	22989	427947	94.90	23981	1726	22255	92.80
2	1.0-2.0	400557	18188	382369	95.45	26629	1280	25349	95.19
3	2.0-3.0	489888	15437	474451	96.84	85326	1703	83623	98.01
4	3.0-4.0	1281242	11401	1269841	99.11	75525	4774	70751	93.67
5	4.0-5.0	567057	12468	554589	97.80	82945	3460	79485	95.82
6	5.0-6.0	1471233	10979	1460254	99.25	85998	3430	82568	96.01
7	6.0-7.0	656060	13354	642706	97.96	82485	1210	81275	98.53
8	7.0-8.0	1335629	11310	1324319	99.15	84643	5630	79013	93.34
9	8.0-9.0	529291	15494	513797	97.07	84910	1105	83805	98.69
10	9.0-10.0	483453	20118	463335	95.83	82493	1700	80793	97.93
Average PDR		766534.6	15173.8	751360.8	97.33	71493.5	2601.8	68891.7	95.99

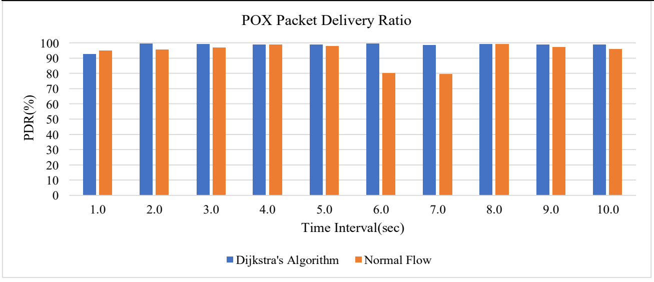

Figure 8: POX Controller Packet Delivery Ratio.

4.2.8 RYU and POX Controller PDR (%)

Figure 9, illustrates the comparison packet delivery ratio of RYU and POX controller. The X-axis represents the number of times for each experiment, and the y-axis displays Packet Delivery Ratio (%).

Table 6: POX & RYU Controller PDR%

S.No	Time Interval(sec)	RYU Packet Delivery Ratio(%)		POX Packet Delivery Ratio(%)	
		Dijkstra's algorithm	Normal Flow	Dijkstra's Algorithm	Normal Flow
1	0.0-1.0	98.30	94.9	94.9	92.8
2	1.0-2.0	99.37	95.45	95.45	95.19
3	2.0-3.0	99.64	96.84	96.84	98.01
4	3.0-4.0	99.54	99.11	99.11	93.67
5	4.0-5.0	99.71	97.8	97.8	95.82
6	5.0-6.0	99.63	99.25	99.25	96.01
7	6.0-7.0	99.80	97.96	97.96	98.53
8	7.0-8.0	99.91	99.15	99.15	93.34
9	8.0-9.0	99.76	97.07	97.07	98.69
10	9.0-10.0	99.83	95.83	95.83	97.93
Average PDR		99.549	98.528	97.336	5.99

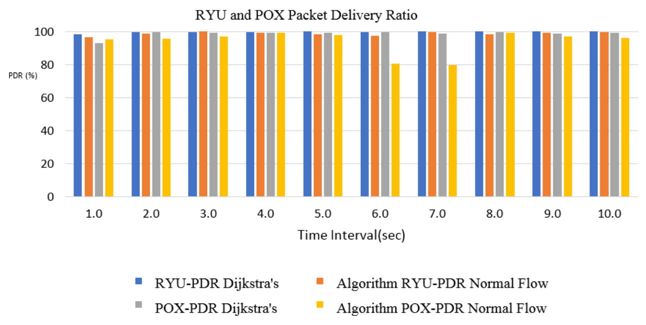

Figure 9: RYU PDR vs. POX PDR

Figures 7 to 9 show the Packet Delivery Ratio (PDR) as a percentage. Figure 9 depicts a comparison of these two controllers. Finally, the DRR performance of the RYU controller is better than a POX controller.

4.2.9 Jitter

In the last experiment, the jitter is the variance in the time delay or the packet delay between when a packet is transmitted and when it is received, the measuring of jitter by making UDP connection between server and client of POX and RYU controller for the various number of times in standard and custom MiniNet topology [31].

4.2.10 RYU Jitter

Below is the jitter table for the value collected for different packets of the RYU controller in Dijkstra's algorithm and normal flow. The X-axis displays the number of times in seconds for each experiment and the y-axis displays the average jitter calculated for different packets.

Table 7: RYU Controller Jitter

S.No	Time Interval(sec)	RYU-Jitter in Dijkstra Algorithm		RYU-Jitter in Normal Flow	
		Total Packet	Jitter(ms)	Total Packet	Jitter(ms)
1	0.0-1.0	89208	0.001	43647	0.065
2	1.0-2.0	89166	0.002	20660	0.015
3	2.0-3.0	89145	0.002	38220	0.014
4	3.0-4.0	89185	0.001	41521	0.019
5	4.0-5.0	89156	0.002	39176	0.021
6	5.0-6.0	89173	0.001	36327	0.025
7	6.0-7.0	89164	0.001	43565	0.019
8	7.0-8.0	88525	0.001	43839	0.011
9	8.0-9.0	89797	0.003	45491	0.026
10	9.0-10.0	89130	0.001	43320	0.013
Average Jitter		89164.9	0.0015	39576.6	0.0228

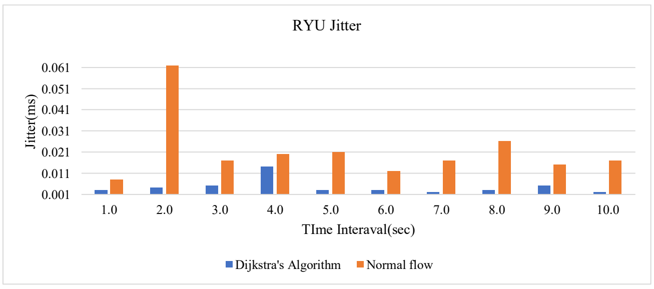

Figure 10. RYU Controller Jitter

4.2.11 POX Jitter

Below is the table of jitter for the value collected for different packets of the POX controller in Dijkstra's algorithm and normal flow. The X-axis displays the number of times in seconds for each experiment and the y-axis represents the average jitter calculated for different packets.

Table 8:POX Controller Jitter

S.No	Time Interval(sec)	POX-Jitter in Dijkstra Algorithm		POX-Jitter in Normal Flow	
		Total Packet	Jitter(ms)	Total Packet	Jitter(ms)
1	0.0-1.0	450936	0.003	23981	0.008
2	1.0-2.0	400557	0.004	26629	0.062
3	2.0-3.0	489888	0.005	85326	0.017
4	3.0-4.0	1281242	0.014	75525	0.02
5	4.0-5.0	567057	0.003	82945	0.021
6	5.0-6.0	1471233	0.003	85998	0.012
7	6.0-7.0	656060	0.002	82485	0.017
8	7.0-8.0	1335629	0.003	84643	0.026
9	8.0-9.0	529291	0.005	84910	0.015
10	9.0-10.0	483453	0.002	82493	0.017
Average Jitter		766534.6	0.0044	71493.5	0.0215

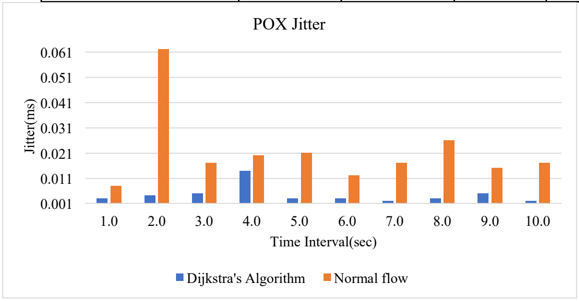

Figure 11: POX Controller Jitter

4.2.12 RYU and POX Controller Jitter

Figure 12, illustrates the comparison jitter of RYU and POX controller. The X-axis displays the number of average jitter times for each experiment, and the y-axis represents the number of packets.

Table 9: RYU and POX Controller Jitter

S.N o	Time Interval(sec)	RYU Jitter(ms)		POX-Jitter(ms)	
		Dijkstra's Algorithm	Normal Flow	Dijkstra's Algorithm	Normal Flow
1	0.0-1.0	0.001	0.065	0.003	0.008
2	1.0-2.0	0.002	0.015	0.004	0.062
3	2.0-3.0	0.002	0.014	0.005	0.017
4	3.0-4.0	0.001	0.019	0.014	0.02
5	4.0-5.0	0.002	0.021	0.003	0.021
6	5.0-6.0	0.001	0.025	0.003	0.012
7	6.0-7.0	0.001	0.019	0.002	0.017
8	7.0-8.0	0.001	0.011	0.003	0.026
9	8.0-9.0	0.003	0.026	0.005	0.015
10	9.0-10.0	0.001	0.013	0.002	0.017
Average Jitter		0.0015	0.0228	0.0044	0.0215

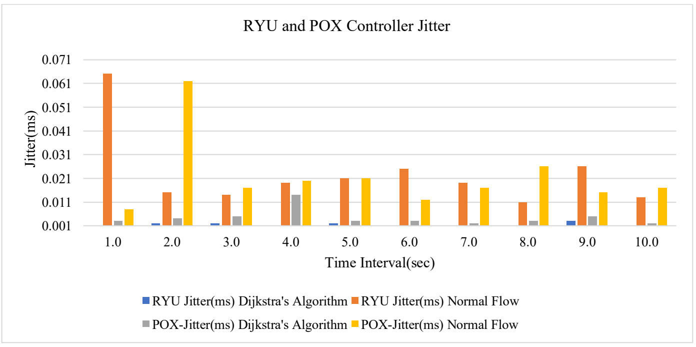

Figure 12: RYU vs POX Controller Jitter

Figures 10 to 12 demonstrate the jitter. Figure 5.9 depicts a comparison of these two controllers. Finally, the Jitter performance of the RYU controller is better than a POX controller.

4.2.13 Packet Loss

Data is sent and retrieved in small units known as packets in every network system. Packet loss refers to data packets that do not arrive at their destination after being transmitted through a computer network and the number of packets lost or discarded during their journey through a computer network [32] [33].

4.2.14 RYU Controller Packet Loss

The Bellow table is of packet loss ratio variability of RYU SDN controller in Dijkstra's algorithm and normal flow (without Dijkstra's), measuring packet loss by making UDP connection between server and client of RYU for the different number of times in custom MiniNet topology [34] [35].

Table 10: RYU Controller Packet Loss

S.No	Time Interval(sec)	RYU Packet-Lost in Dikstra's algorithm			RYU Packet-Lost in Normal Flow		
		Total Packet	Packet Lost	Packet-Lost(%)	Total Packet	Packet Lost	Packet-Lost(%)
1	0.0-1.0	89208	1514	1.69	43647	1566	3.58
2	1.0-2.0	89166	557	0.62	20660	321	1.55
3	2.0-3.0	89145	315	0.35	38220	47	0.12
4	3.0-4.0	89185	406	0.45	41521	365	0.87
5	4.0-5.0	89156	251	0.28	39176	646	1.64
6	5.0-6.0	89173	327	0.36	36327	965	2.65
7	6.0-7.0	89164	178	0.19	43565	315	0.72
8	7.0-8.0	88525	72	0.08	43839	834	1.90
9	8.0-9.0	89797	207	0.23	45491	434	0.95
10	9.0-10.0	89130	146	0.16	43320	278	0.64
Average Packet Lost		89164.9	397.3	0.441	39576.6	577.1	1.462

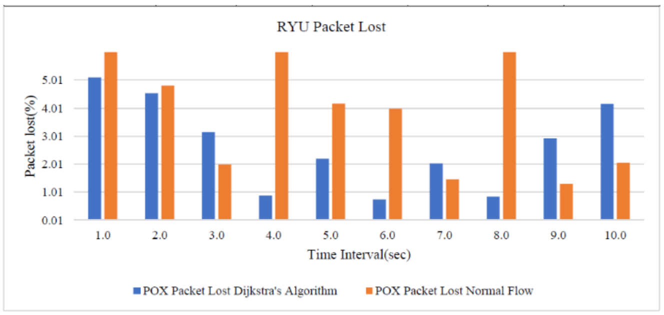

Figure 13: RYU Controller Packet Loss

4.2.15 POX Controller Packet Loss

The Bellow table is of packet loss ratio variability of POX SDN controller in Dijkstra's algorithm and normal flow, the measuring of packet loss by making UDP connection between server and client of POX for the different number of times in custom MiniNet topology.

Table 11:POX Controller Packet Loss

S.No	Time Interval(sec)	POX Packet Lost in Dijkstra's Algorithm			POX Packet Lost in Normal Flow		
		Total Packet	Packet lost	Packet-Lost(%)	Total Packet	Packet Lost	Packet-Lost(%)
1	0.0-1.0	450936	22989	5.09	23981	1726	7.19
2	1.0-2.0	400557	18188	4.54	26629	1280	4.80
3	2.0-3.0	489888	15437	3.15	85326	1703	1.99
4	3.0-4.0	1281242	11401	0.88	75525	4774	6.32
5	4.0-5.0	567057	12468	2.19	82945	3460	4.17
6	5.0-6.0	1471233	10979	0.74	85998	3430	3.98
7	6.0-7.0	656060	13354	2.03	82485	1210	1.46
8	7.0-8.0	1335629	11310	0.84	84643	5630	6.65
9	8.0-9.0	529291	15494	2.92	84910	1105	1.30
10	9.0-10.0	483453	20118	4.16	82493	1700	2.06
Average Packet Lost		766534.6	15173.8	2.654	71493.5	2601.8	3.992

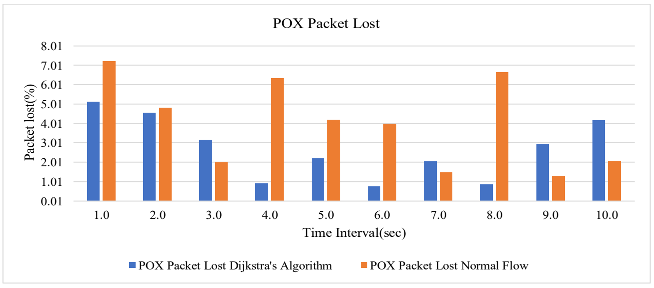

Figure 14: POX Controller Packet Lost

4.2.16 RYU and POX Controller Packet Loss

Figure 15, illustrates the comparison packet loss of RYU and POX controller. The X-axis represents the number of times of packet losses for each experiment, and the y-axis represents the number of packets.

Table 12: RYU vs POX Controller Packet Loss

S.No	Time Interval(sec)	RYU Packet Lost(%)		POX Packet Lost(%)	
		Dijkstra's Algorithm	Normal Flow	Dijkstra's Algorithm	Normal Flow
1	0.0-1.0	1.69	3.58	5.09	7.19
2	1.0-2.0	0.62	1.55	4.54	4.80
3	2.0-3.0	0.35	0.12	3.15	1.99
4	3.0-4.0	0.45	0.87	0.88	6.32
5	4.0-5.0	0.28	1.64	2.19	4.17
6	5.0-6.0	0.36	2.65	0.74	3.98
7	6.0-7.0	0.19	0.72	2.03	1.46
8	7.0-8.0	0.08	1.90	0.84	6.65
9	8.0-9.0	0.23	0.95	2.92	1.30
10	9.0-10.0	0.16	0.64	4.16	2.06
Average Packet Lost		0.441	1.462	2.654	3.992

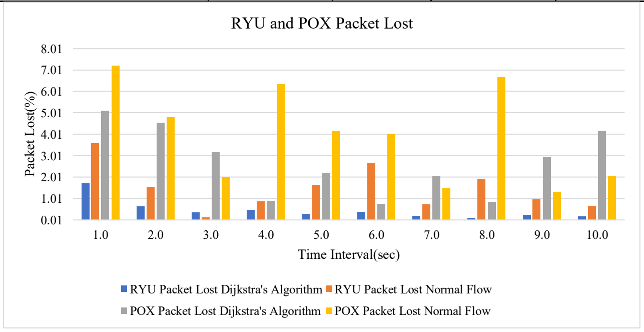

Figure 15: RYU vs POX Controller Packet Loss

Figures 13 to 15 demonstrate the Packet loss. Figure 15 depicts a comparison of these two controllers. Finally, the RYU controller outperforms the POX controller in packet loss.

5. Conclusion

Software-Defined Networking is a new concept which exists for about 20 years. However, it has become relevant in the network area in the last few years. This is due to the increasing necessities in network programmability and traffic that have been driven by the development of other areas such as network virtualization, mobile devices, and others. A controller is the principal construction of SDN. In this paper, the performance evaluation of two open-source controllers (POX, and RYU) was compared based on jitter, packet loss, throughput, and packet delivery ratio for custom topology in the MiniNet emulator. The performance evaluation of POX and RYU shows that the POX controller provides a better result in terms of throughput. In the packet delivery ratio, jitter, and packet loss the RYU controller provides better performance.

References

[1] E. Rojas, "From software-defined to human-defined networking: Challenges and opportunities," IEEE Network, vol. 32, pp. 179-185, 2017.

[2] N.Ullah, S. I. Ullah, A. W. Ullah, A. Salam, M. Imad, and F. Ullah, "Performance Analysis of POX and RYU Based on Dijkstra's Algorithm for Software Defined Networking," in European, Asian, Middle Eastern, North African Conference on Management & Information Systems, 2021: Springer, pp. 24-35.

[3] D. S. Rana, S. A. Dhondiyal, and S. K. Chamoli, "Software defined networking (SDN) challenges, issues and solution," Int J Comput Sci Eng, vol. 7, pp. 884-889, 2019.

[4] S. Barguil, V. Lopez, and J. P. F.-P. Gimenez, "Towards an open networking architecture," in 2020 International Conference on Optical Network Design and Modeling (ONDM), 2020, pp. 1-3.

[5] I. Z. Bholebawa and U. D. Dalal, "Performance analysis of SDN/OpenFlow controllers: POX versus floodlight," Wireless Personal Communications, vol. 98, pp. 1679-1699, 2018.

[6] B. Pandya, S. Parmar, Z. Saquib, and A. Saxena, "Framework for securing SDN southbound communication," in 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), 2017, pp. 1-5.

[7] M. A. Hassan, S. I. Ullah, A. Salam, A. W. Ullah, M. Imad, and F. Ullah, "Energy efficient hierarchical based fish eye state routing protocol for flying ad-hoc networks," Indonesian Journal of Electrical Engineering and Computer Science, vol. 21, no. 1, pp. 465-471, 2021.

[8] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, et al., "Advancing software-defined networks: A survey," IEEE Access, vol. 5, pp. 25487-25526, 2017.

[9] T. G. Robertazzi, "Software-defined networking," in Introduction to Computer Networking, ed: Springer, 2017, pp. 81-87.

[10] S. Asadollahi, B. Goswami, A. S. Raoufy, and H. G. J. Domingos, "Scalability of software defined network on floodlight controller using OFNet," in 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), 2017.

[11] M. A. Hassan, M. Imad, T. Hassan, F. Ullah, and S. Ahmad, "Impact of Routing Techniques and Mobility Models on Flying Ad Hoc Networks," in Computational Intelligence for Unmanned Aerial Vehicles Communication Networks: Springer, 2022, pp. 111-129.

[12] A. Hussain, M. Imad, A. Khan, and B. Ullah, "Multi-class Classification for the Identification of COVID-19 in X-Ray Images Using Customized Efficient Neural Network," in AI and IoT for Sustainable Development in Emerging Countries: Springer, 2022, pp. 473-486.

[13] M. Vahlenkamp, F. Schneider, D. Kutscher, and J. Seedorf, "Enabling ICN in IP networks using SDN," in 2013 21st IEEE International Conference on Network Protocols (ICNP), 2013, pp. 1-2.

[14] D. Erickson, "The beacon openflow controller," in Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, 2013, pp. 13-18.

[15] S. Lateef, M. Rizwan, and M. A. Hassan, "Security Threats in Flying Ad Hoc Network (FANET)," Computational Intelligence for Unmanned Aerial Vehicles Communication Networks, pp. 73-96, 2022.

[16] Z. M. Imad, S. I. Ullah, A. Salam, W. U. Khan, F. Ullah, and M. A. Hassan, "Automatic Detection of Bullet in Human Body Based on X-Ray Images Using Machine Learning Techniques," International Journal of Computer Science and Information Security (IJCSIS), vol. 18, no. 6, 2020.

[17] S. Kaur, J. Singh, and N. S. Ghumman, "Network programmability using POX controller," in Proc.

[18] M. Imad, A. Hussain, M. A. Hassan, Z. Butt, and N. U. Sahar, "IoT Based Machine Learning and Deep Learning Platform for COVID-19 Prevention and Control: A Systematic Review," AI and IoT for Sustainable Development in Emerging Countries, pp. 523-536, 2022.

[19] M. A. Hassan, A. R. Javed, T. Hassan, S. S. Band, R. Sitharthan, and M. Rizwan, "Reinforcing Communication on the Internet of Aerial Vehicles," IEEE Transactions on Green Communications and Networking, 2022.

[20] M. Darianian, C. Williamson, and I. Haque, "Experimental evaluation of two openflow controllers," in 2017 IEEE 25th International Conference on Network Protocols (ICNP), 2017, pp. 1-6.

[21] Y. Zhang, L. Cui, W. Wang, and Y. Zhang, "A survey on software defined networking with multiple controllers," Journal of Network and Computer Applications, vol. 103, pp. 101-118, 2018.

[22] A. V. Priya and N. Radhika, "Performance comparison of SDN OpenFlow controllers,"

[23] International Journal of Computer Aided Engineering and Technology, vol. 11, pp. 467-479, 2019.

[24] M. Z. Abdullah, N. A. Al-awad, and F. W. Hussein, "Performance Comparison and Evaluation of Different Software Defined Networks Controllers," International Journal of Computing and Network Technology, vol. 6, 2018.

[25] A. Jasim and D. Hamid, "Enhancing the performance of OpenFlow network by using QoS," International Journal of Scientific & Engineering Research (IJSER), vol. 7, pp. 950-955, 2016.

[26] R. K. Chouhan, M. Atulkar, and N. K. Nagwani, "Performance Comparison of Ryu and Floodlight Controllers in Different SDN Topologies," in 2019 1st International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE), 2019, pp. 188-191.

[27] C. Fancy and M. Pushpalatha, "Performance evaluation of SDN controllers POX and floodlight in MiniNet emulation environment," in 2017 International Conference on Intelligent Sustainable Systems (ICISS), 2017, pp. 695-699.

[28] J. P. Duque, D. D. Beltrán, and G. P. Leguizamón, "OpenDaylight vs. Floodlight: Comparative Analysis of a Load Balancing Algorithm for Software Defined Networking,"

International Journal of Communication Networks and Information Security, vol. 10, pp. 348-357, 2018.
[29] H. Sufiev and Y. Haddad, "A dynamic load balancing architecture for SDN," in 2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE), 2016, pp. 1-3.
[30] L. Zhu, M. M. Karim, K. Sharif, F. Li, X. Du, and M. Guizani, "Sdn controllers: Benchmarking & performance evaluation," arXiv preprint arXiv:1902.04491, 2019.
[31] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, and K. Prabhu, "iPerf-The ultimate speed test tool for TCP, UDP and SCTP," línea]. Available: https://iperf. fr.[Último acceso: 23 Mayo 2018], 2014.
[32] S. I. Ullah, A. Salam, W. Ullah, and M. Imad, "COVID-19 lung image classification based on logistic regression and support vector machine," in European, Asian, Middle Eastern, North African Conference on Management & Information Systems, 2021: Springer, pp. 13-23.
[33] M.Imad, N. Khan, F. Ullah, M. A. Hassan, and A. Hussain, "COVID-19 classification based on Chest X-Ray images using machine learning techniques," Journal of Computer Science and Technology Studies, vol. 2, no. 2, pp. 01-11, 2020.
[34] A. Salam, F. Ullah, M. Imad, and M. A. Hassan, "Diagnosing of Dermoscopic Images using Machine Learning approaches for Melanoma Detection," in 2020 IEEE 23rd International Multitopic Conference (INMIC), 2020: IEEE, pp. 1-5.
[35] M. Imad, F. Ullah, and M. A. Hassan, "Pakistani Currency Recognition to Assist Blind Person Based on Convolutional Neural Network," Journal of Computer Science and Technology Studies, vol. 2, no. 2, pp. 12-19, 2020.
[36] M. Rizwan et al., "Risk monitoring strategy for confidentiality of healthcare information," Computers and Electrical Engineering, vol. 100, p. 107833, 2022.
[37] R. V Boppana, R. Chaganti, and V. Vedula. "Analyzing the vulnerabilities introduced by ddos mitigation techniques for software-defined networks." National Cyber Summit. Springer, Cham, 2019.
[38] V. Ravi, R. Chaganti and M. Alazab, "Deep Learning Feature Fusion Approach for an Intrusion Detection System in SDN-Based IoT Networks", IEEE Internet of Things Magazine, vol. 5, no. 2, pp. 24-29, 2022. Available: 10.1109/iotm.003.2200001.
[39] M. A. Hassan, S. Ali, M. Imad and S. Bibi, “New Advancements in Cybersecurity: A Comprehensive Survey” Big Data Analytics and Computational Intelligence for Cybersecurity,pp. 3-17, 2022.
[40] M. Imad, M. A. Hassan, S. H Bangash, “A Comparative Analysis of Intrusion Detection in IoT Network Using Machine Learning” In Big Data Analytics and Computational Intelligence for Cybersecurity, pp. 149-163, 2022. Springer, Cham.