Multipath Routing Algorithm to find Optimal Path in SDN with POX Controller

Original Scientific Paper

Deepthi Goteti*
Department of Computer Science & Engineering,
Koneru Lakshmaiah Education Foundation,
Green Fields, Vaddeswaram, A.P, India
2102031088@kluniversity.in

Imran Rasheed
Department of Computer Science & Engineering,
Koneru Lakshmaiah Education Foundation,
Green Fields, Vaddeswaram, A.P, India
imran.rasheedamu@kluniversity.in

*Corresponding author

Abstract *– The past decade witnessed a tremendous increase in network usage, and traditional network architecture is needed to sustain modern requirements with high throughput and minute delay. This leads to the introduction of software-defined networks. Congestion is a critical problem that needs attention, so identifying the optimal path is required to eliminate the congestion. Researchers introduce rigorous studies to identify optimal paths, some resulting in less data loss and delay. Identifying multiple paths between nodes may eliminate congestion. When the first best path is congested, selecting the second best path between nodes can solve the congestion problem. With this ideology, the multipath routing algorithm is developed and tested on Fat Tree, Custom, and Tree topologies, and performance is measured using quality of service factors. Considering Throuhput, Fat Tree produced 27.15% better throughput than the tree topology and 17.57% better than the custom topology, Whereas in the case of jitter, fat tree topology reduces jitter by about 90.36%compared to the tree topology, but custom topology reduces jitter by about 12.24% compared to the fat-tree topology. In the packet delivery ratio, fat tree topology reduces packet loss by about 77.87% compared to the tree topology. Fat tree topology reduces packet loss by about 55.62% compared to the custom topology. Fat tree performs best overall, with the highest throughput, lowest packet loss, and significantly reduced jitter compared to the tree and custom topologies.MiniNet is used to perform simulations. TCP and UDP flows are calculated with the iperf tool and tested on the POX Controller.*

Keywords*: SDN, Congestion, POX, Multipath, MiniNet*

Received: September 6, 2024; Received in revised form: October 28, 2024; Accepted: November 26, 2024

1. INTRODUCTION

The network's infrastructure is based on hardware, and a multitude of devices, like switches and routers, work for data forwarding and operate with rules and requirements. This is resisting convolution networks' upgrades to services [1]. Network usage has immensely increased over the past decade, and traditional networks are unable to withstand due to the unadoptable nature of their architectures [2].

To meet modern requirements, layered architecture is unsuitable, so the architecture is designed into planes in software-defined networks. Every plane is dedicated to a particular responsibility and can add programmability to SDN, which enhances the chances of opting for the SDN. The data plane is tightly packed with various data forwarding devices, and the control plane works as intelligence to network [3].

Data planes are designed to work with switches and to create topologies. POX controller is used for routing decisions, traffic monitoring, and identifying the optimal path between two nodes and computing the shortest route; multipath algorithm is implemented with the help of POX controller on Fat Tree and Custom topologies.

The main aim of separating planes is to use resources efficiently and control and secure them. Due to that,

numerous controllers like Floodlight, RYU, POX, OpenDayLight, and POX have been developed. SDN has a wide range of Controllers, and selecting the optimum controller depends on the application. Gupta et al. assess and contradict various SDN controllers, and simulations are conducted with Mininet [4].

Control planes are responsible for routing decisions, using open flow protocol as an interface. Using the open-flow protocol, the controller will identify paths across switches for data packets [5]. Centralized and distributed path computations are well-known approaches with prominent results due to their dynamic resource handling. The author analyzed these techniques and performed simulations to test critical factors such as latency, throughput, and fault tolerance in various traffic patterns with varying loads. The research provided valuable insights into optimized path computation with an extensive network and low latency. Results reveal that the centralized approach is superior but needs more scalability. Whereas distributed approaches face challenges with higher latency, the author concludes that hybrid models perform best in scalability and fault tolerance, but security issues must be addressed [6].

Caria et al. proposed a model that combines distributed routing with a centralized control plane, which allows centralized decision-making. Results are noted on parameters like control overhead and scalability. The author concluded that this approach could manage large-scale features [7].

Denar et al. mentioned that handling massive data by switches will downgrade the performance, so the author suggested threading and multiprocessing, which are parallel programming methods that improve controller performance on CPU time consumption, memory usage, and execution time and it concludes that Ryu produced superior results over POX [8]. The application plane uses logical programs handled by APIs and deploys software, routing, and policies. Planes are communicated with two sets of interfaces: southern and northern [9]. SDN architecture, which includes various planes, is shown in Fig. 1.

If data transmission needs to occur between two dedicated nodes over a network, then we need to identify the optimal path between nodes. Then transmission will happen. The main drawback of selecting the same optimal path every time is that it leads to congestion, which leads to network performance degradation. We address this problem by identifying multiple paths between nodes, regardless of the cost and length of the paths. We measured network performance with Throughput, jitter, and packet loss. The controller used in the control plane influences these metrics and the overall network design. The multipath algorithm meticulously tests three distinct topologies using the POX controller within the innovative Software-Defined Networking (SDN) framework. We carry out simulations using a MiniNet simulator.

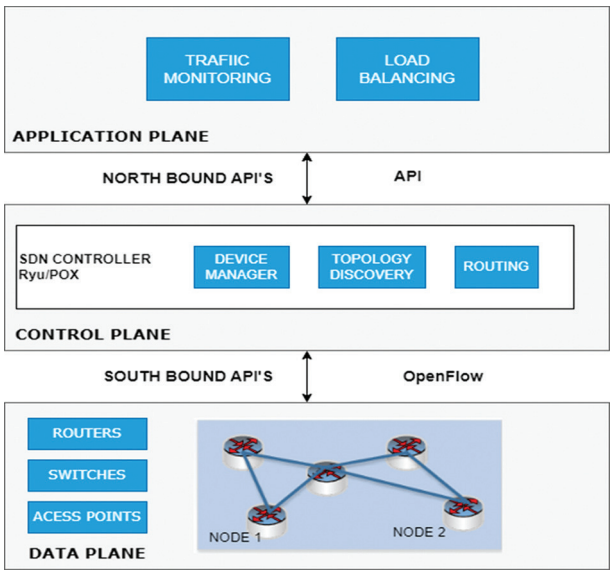

Fig. 1. SDN architecture

The Fat Tree topology performed well over custom and Tree topologies in Throughput and packet loss. On the other hand, a custom topology also demonstrated slightly improved performance, with less jitter than the Fat Tree topology.

The remaining sections explain the methodology followed by the simulations taken over topologies with the POX controller. The result analysis showcases the QOS factor over topologies through extensive benchmarking using the iPerf tool.

2. LITERATURE REVIEW

2.1. SDN CONTROLLERS

This research mainly concentrates on increasing the throughput of network with optimal path algorithm, Optimal path will allow packet loss routing from source to given destination without congestion. static routers are not suitable if there is a drastic increase of load in network also fail to maintain if nodes are added routing tables are fail to control this may be result link failure and congestion in SDN. SDN controller computes path and maintains network information. So, it is called a central part of the SDN. The network is more efficiently managed with the help of OpenFlow protocol. SDN controller forwards the packets from source to destination to establish the path we are using. Multi-path routing is tested with a POX controller, and performance is measured with QoS factors.

2.2. EXPLORING THE FUNCTIONALITY OF SDN CONTROLLER OVER DIFFERENT APPROACHES

POX is inherited from NOX Controller, written in python code and implemented in OpenFlow protocol. It is mainly used for academic research due to its ease of use and flexibility to develop network system and

control applications. MiniNet is used to build POX controller code available from GitHub [10, 11]. This can run multiple programs like switch, load balance and hub.

Mohammadi et al. compared conventional and SDN based on throughput, packet loss, and delay on three typologies with Wire shark and concluded that linear topology performed better than tree topology in delay and throughput [12]. According to Salman et.al., with OpenFlow protocol, POX can directly access forwarding devices which is easy and perfect for experiments and demonstrations [13]. In this paper, we are going to test the shortest path computing algorithm on custom and Fat Tree typologies with QoS as measuring factors.

Many researchers analyzed the performance of Ryu, POX, ONOS, OpenDaylight, and NOX on various typologies and with common factors like throughput, jitter, and packet loss.

Table 1. shows a literature review

Author	Year	Topology/Approach	Controller	QoS Parameters	Simulators
Patel et al. [14]	2023	Comparative evaluation of SDN controllers	POX, Ryu, OpenDaylight	Response time, resource management	Mininet-WiFi
Smith et al. [15]	2023	Enhancements in POX SDN topologies	POX	Latency, throughput, reliability	Mininet
Koulouras et.al. [16]	2022	Abilene Network, GEANT Network	ONOS, Ryu OpenDaylight	Throughput, RTT latency, packet loss, and jitter	Mininet
Wilson et al. [17]	2022	Ryu SDN controller for scalable topologies	Ryu	Scalability, resource efficiency	Mininet
Liehuang Zhu et al. [18]	2020	IoT and VANETS	POX, Ryu,Nox,ONOS, Floodlight, and OD	Latency and throughput	CBench, PktBlaster, and OFNet
Lee et al. [19]	2020	Ryu SDN framework	Ryu	Latency, throughput, resource management	Mininet
Numan et al. [20]	2019	Single	POX	RTT, Jitter, Delay	Ping, iperf, MiniNet
Sajid et al. [21]	2018	Round Robin & Random Algorithm	POX	response time and transaction per second	MiniNet
Duque et al. [22]	2018	Custom	POX Floodlight	QoS Metrics	Mininet
Farrugia et al. [23]	2018	Geant, Butterfly,Optimized peer- Multipath	OpenDaylight	throughput, jitter	NS-3.26
Abdul-hafiz et al. [24]	2017	Abilene network/Improved Dijkstra's	Ryu	throughput and latency	MinNet,Iperf
Bholebawa et al.[25]	2016	OpenFlow-enabled network topologies	POX	Latency, throughput	Mininet
Yahya et al. [26]	2015	Abilene network/Extended Dijkstra's	Ryu	end-to-end latency, and throughput.	MinNet,Iperf
Zhang et al. [27]	2015	Tree-based topology design	POX	Throughput, delay	Mininet
Stancu et al. [28]	2015	Comparison of SDN controllers	Various	Performance metrics (response time, resource utilization)	Mininet

Table 1 shows a literature review of different articles. A few authors have worked on Dijkstra's performance evaluation on multiple controllers, as noted in the table below. Numerous authors worked on finding optimal paths and analyzing the performance of various topologies and controllers observed from the last decade; however, identifying multiple paths and testing them is presented in this work with the help of the POX controller on topologies like Custom and Fat Tree. The multipath algorithm will be tested on various controllers with further available topologies.

Ram et al. analyze SDN performance in wireless and wired networks over single, linear, and tree topologies. They measure POX and RYU controllers' metrics, such as jitter, Bitrate, and packet loss, with D-ITG, Mininet, and Mininet-WiFi simulators. The research found that wireless networks showed performance inconsistency through SDN-optimized resource management. However, this research has limited application in real-world applications, and more dynamic topologies need to be tested.

N. Ullah et al. worked on evaluating the performance of Dijkstra's algorithm on POX and Ryu controllers. They measured Jitter, throughput, packet loss, and packet delivery ratio with iperf, Wireshark, and the MiniNet simulation tool. Where RYU outpaced POX in terms of witnessing low Jitter and higher throughput, which is well suited for dynamic networks. In the case of packet loss, both have near results. Research can be done on more dynamic and hybrid topologies to know the adoptability of this approach [29].

Several distributed but logically centralized controllers are available, including POX, Ryu, Floodlight, and OpenDayLight; the author compared these with Ryu. Ryu performed well due to its scalability and modularity; when testing the MniNet emulator, the author exhibited the controller's scalability and reliability and studied network performance under heavy loads over linear topology. The study briefly touches on operational styles but suggests further research is needed to fully understand how different modes impact controller efficiency in various environments [30]. Therefore, we must test any algorithm that needs more in-depth testing with various topologies.

Koulouras et al. worked on the evaluation of various SDN controllers customized for wireless networks. To assess controllers, the authors specially used an analytic hierarchy process, and they found Ryu and ONOS to be the best among other controllers. They concluded that selecting the controller plays a crucial role in evaluating the performance of any approach. Still, studies must fully explore wireless protocols like 5G and their adaptability to massive networks and dynamic conditions. The type of topology also makes a difference in various quality of service factors [31].

This is proved by exploring optimized tree-based network topology in SDN and applying various optimization strategies like routing to address inefficiencies in traditional topologies and manage traffic in a Dynamic way. Data centers use tree topology because it performs better under higher loads. Author prospered teaching produced a 10% increase in throughput and a 15 %reduction in latency. However, the author stuck to only tree topology and did not do rigorous testing on dynamic networks [32]. Identifying the optimal path is achieved by Dijkstra's algorithm, and with the help of the POX controller, path computation and network performance are enhanced. Real-time network conditions drive dynamic decisions on routing. Algorithm efficacy is measure with QOS factors. It also addressed critical issues in traditional routing and achieved noticeable throughput and less jitter. However, it still lacks energy efficiency and is stuck to general topologies like mesh, star, and ring topologies where the structure is straightforward. Calculating the optimal path is simple [33].

Cabarkapa et al. conducted a performance analysis of the Ryu-POX controller in various tree-based Software-Defined Networking (SDN) topologies. The methodology involves simulations in a controlled environment where different tree-based topologies, such as binary and balanced trees, are evaluated [34]. The quality-of-service factors measures the effectiveness of the Ryu and POX controllers in handling the load over these topologies and performance. The quality-of-service factors measures the effectiveness of the Ryu and POX controllers in handling the load over these topologies and performance. This work proves that selecting topology depends on the traffic type and network demand. Most of the researchers' work concluded that selecting a proper controller and testing with multiple topologies will help them evaluate any approach's performance. Also, for finding an optimal path, Dijkstra is one role model, though congestion is a significant issue. So, in this paper, we have used multipath routing tested on multiple topologies on a pox controller.

3. PROPOSED MECHANISM

Congestion will occur when two dedicated nodes select the same optimal path for every transmission and must identify alternative paths to avoid network traffic. Calculating multiple paths between all available nodes in the network is necessary. The algorithm will choose an alternative path if the load floods the selected path. This proposed mechanism discusses selecting multiple paths based on bandwidth, link cost, and hop number between dedicated nodes. Multipath routing algorithm start with finding the routes with the help of the Depth First Search algorithm, whose working functionality, is to go deep from other adjacent nodes of the last visited node of a graph [35]. DFS algorithm is showed in table. With the DFS algorithm, every node is visited once to compute paths from each node to every other node. Pseudocode shows the Multipath routing algorithm. It starts with executing topology at one end; on another end, The system initializes the POX controller and executes the Multipath algorithm. After initializing the get path function, we will use DFS to acquire paths between dedicated nodes, as we expect paths between the source and destination. The routing table stores all paths, and Get-link-cost will be used to calculate path cost.

Algorithm:

RecurDFS(G_i, root):
Traversed <- set all nodes false initially
DFS(root)
function **DFS**(u):
if Traversed [u_i] = true:
return
print(u_i)
Traversed [u_i] <- true
for each vi in $G[u_i]$. neighbors ():
DFS(v_i)
Input: G is graph in Adjacency list where root is starting node
Output: DFS order nodes in that graph are printed

At the start, bandwidth is initialized. B1 is the minimum bandwidth requirement between si and sj, ewi is the bandwidth capacity of the edge between si and sj, and reference bw/bl refers to the baseline value. It ensures B1, the minimum bandwidth requirement met by the bandwidth available between si and sj, compared to the reference BW value.

We identify all possible paths and calculate the cost. The shortest Path will return a less costly option. Get Path will display several available paths and sort them in order. Upon selecting the Path,add_ports_to paths functions add ports for communication between hosts, which involves various link and switch handling functions. In the data plane, topology is created with a set of nodes and switches from the control plane. The Multipath algorithm computes paths between source to destination, and performance is measured with iPerf and wire shark with Mini-Net as the simulation environment. The following sections present the simulations, network setup, and results.

Pseudocode

1. Define function $get\_paths(self, s_i, dj)$:
   return $paths$
   DFS algorithm to find $paths$

2. Define function *get_link_cost*(*self*, s_i, s_j):
   set e_r1 to *self.adjacency*[s_i][s_j]
   set e_r2 to *self.adjacency*[s_i][s_j]
   set *bl* to *min*(*self.bandwidths*[s_i][s_j], *self.bandwidths*[s_i][s_j])
   set ew_i to *reference_bw/bl*
   return ew_i
3. Define function *get_path_cost*(*self*, *path*):
   return *cost*
4. Define function *get_optimal_paths*(*self*, s_r, d_j):
   # s_r is switch, d_j is switch
   set paths to *self.get_paths*(sr, dj)
   set *paths_count* to *length*(*paths*)
   if *length*(*paths*) < *maxi_paths*
      else *maxi_paths*
   return *sorted*(*all_set_of_paths*)
5. Define function *add_ports_to_paths*(*self*, *paths*, *first_port*, *last_port*):
      # assigning *ports* to *path*
6. return *paths_p*

Fig. 2 represents the flow of work. This includes Topologies applied and simulations taken on multipath routing algorithm with implementation on POX controller.

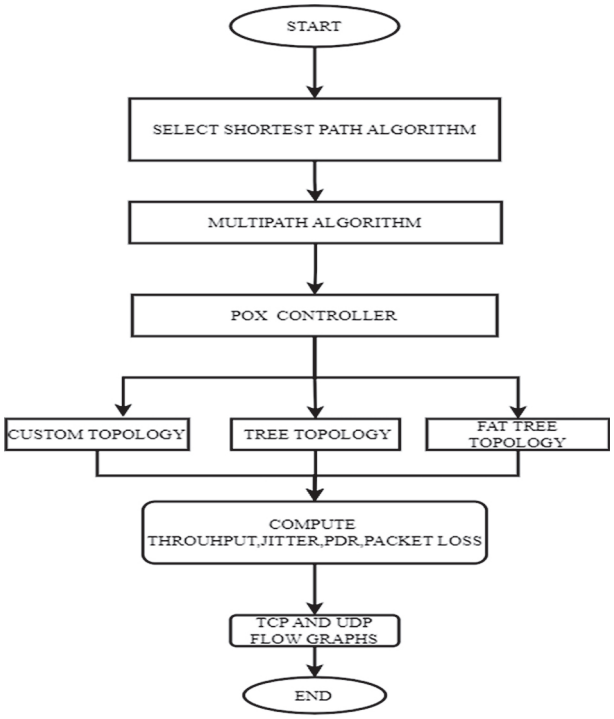

Fig. 2. Methodology

4. SIMULATION SETUP

We conduct simulations using MiniNet [36]. Stanford University develops it. It includes routers, switches, end-hosts, and SDN controllers, which are of OpenFlow and allow users to create networks with the MiniEdit tool via a graphical interface. Users can test different topologies, and it has a feature of CLI support to create and test switches and routers. Mininet supports wide-ranging controllers for flexible simulations. It gears up its usage in various sectors like education, research, and development by allowing Operating system virtualization with hundreds of nodes [37, 38].

We worked with Mininet on the 2.3.0 version of the default operating system with ubuntu-20.04.4. controller installed with POX-2.0, switch is over 2.5.4, SBI is OpenFlow 1.3 with iperf 2.0.13 also used Wireshark with 4.0.6 to analyze the network.

4.1. TOPOLOGIES

Topologies can be created either by using commands or through MiniEdit. MiniNet also supports creating topology with Python code. Here, we have used all three ways to create topologies, and this work presents a performance evaluation of the Multipath routing algorithm with Fat tree, Tree, and Custom Topologies. Python code creates a Fat Tree topology, Tree Topology with MiniEdit, and Custom topology through commands, but we have drawn all topologies in MiniEdit for better visual clarity. Fig. 3 shows the Fat Tree topology. It has three levels, where the core level is to create redundant paths, connect to aggregate switches, and ensure multiple paths between pairs of edge switches. The middle level is the aggregation level. It distributes traffic to the core level received from edge-level switches. The edge level has end host devices that directly connect to the network. In fat tree topology, each end node connects to the top of the rack switch. Fat Tree was chosen because of its identical bandwidth for bisections; each layer has the same aggregated bandwidth. Also, each port has the same speed at the end host.

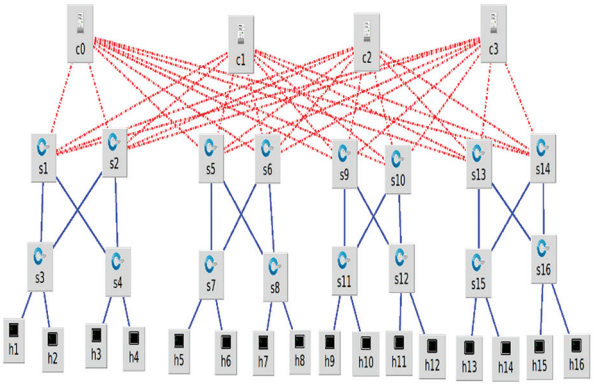

Fig. 3. Tree Topology

Observations are also taken on custom topology with superuser privileges with a remote controller located at 127.0.0.1. This means the controller can run on the same machine as MiniNet. Open vSwitch is a switch type, and the system sets the OpenFlow protocol version to 1.3. A custom tree topology creator designs it with a fan-out of 3, which means each node has three child nodes, and depth means several levels. Fig. 4 shows the custom topology. The controller creates the tree topology with nine switches, connecting them to 10 nodes in various ways.

The system generates these three topologies at the data plane while testing occurs at the control plane using a POX controller. A multipath routing algorithm calculates a path between dedicated nodes, as shown in Pseudocode.

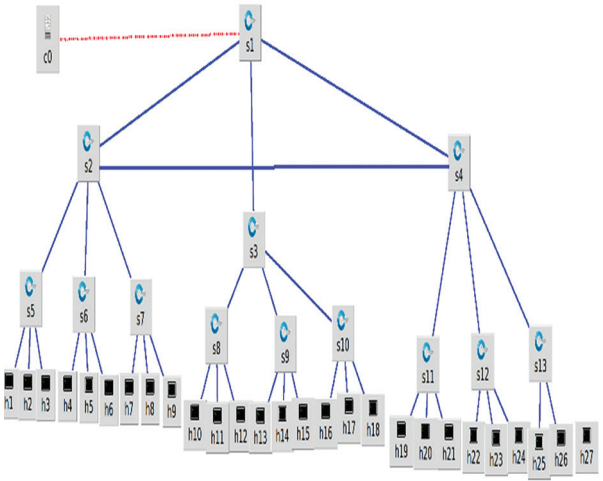

Fig. 4. Custom Topology

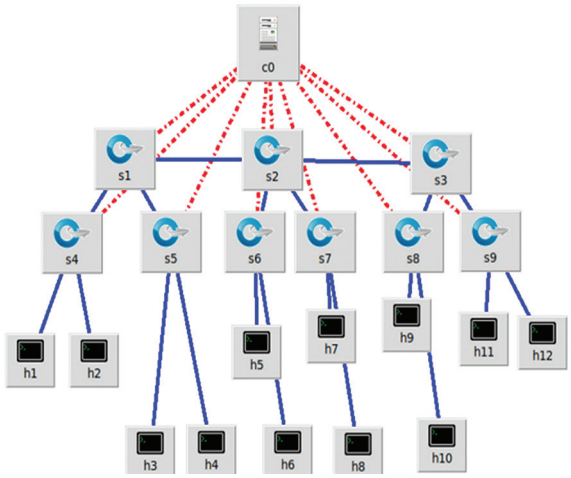

Fig. 5. Tree Topology

5. SIMULATION SETUP

Simulations are carried out with a POX controller on three topologies under three test cases. In every test case, the common factor is finding the multiple paths between node one and node two and calculating throughput, jitter, and packet loss ratio with the help of a network analyzing tool. Measured TCP and UDP flows. Test case 1 includes Testing with Fat Tree topology, Test case 2 is with custom topology, and Test case 3 is with Tree topology.

5.1. TEST CASE 1: FAT-TREE

POX implements the multipath algorithm on fat-tree topologies, and We use the iPerf benchmark utility to obtain TCP and UDP flows and observe bandwidth performance. When we execute "Fat Tree.py" in the terminal, it creates topologies according to the SDN architecture in what can be called a data plane. In the POX environment, the system opens another terminal to execute the Multipath algorithm. Links start observed between dedicated nodes after executing h1 ping h16. or we use Xtrem h1, h16, and ping from h1 to h16.with set of perf commands and note TCP and UDP flows.

Table 2. TCP and UDP Flow-Fat Tree with Multipath Routing Algorithm

C.T	TCP FLOW	UDP FLOW		
Interval (sec)	Bandwidth (Gbits/sec)	Bandwidth (Mbits)	Jitter (ms)	Lost/Total Datagram
0.0- 1.0	17.2	10.7	0.014	56/ 984
1.0- 2.0	18.7	10.6	0.084	8/ 894
2.0- 3.0	19.7	10.5	0.099	5/ 983
3.0- 4.0	14.5	10.4	0.104	9/ 898
4.0- 5.0	18.6	10.6	0.184	1/ 882
5.0- 6.0	17.3	10.5	0.211	4/ 888
6.0- 7.0	17.8	10.7	0.245	5/ 862
7.0- 8.0	17.6	10.2	0.017	31/ 898
8.0- 9.0	18.6	10.5	0.011	26/ 899
9.0-10.0	17.18	10.6	0.039	3/898
0.0-10.0	17.71	10.53	0.098	148/9085

We measure throughput from TCP and UDP flow and observe jitter and packet loss from the UDP flow. Table 2 shows the Fat Tree TCP flow and UDP flow.

5.2. TEST CASE 2: CUSTOM TOPOLOGY

Switch connected to 3 more switches. Each switch at the third level connects to three hosts (Hosts "h1" to "h8"). In the sudo command, we specified the controller as remote and IP 127.0.0.1; this specifies the controller for the MiniNet network. Fig. 4 shows the custom topology when Ping is executed between the H1 and H27 respective switches, making the path from source to destination. The multipath algorithm is applied, and TCP and UDP flow are noted in Table 3. Congestion can be avoided by carefully designing the custom topology.

Table 3. TCP and UDP Flow-Custom topology with Multipath Routing Algorithm

C.T	TCP FLOW	UDP FLOW		
Interval (sec)	Bandwidth (Gbits/sec)	Bandwidth (Mbits)	Jitter (ms)	Lost/Total Datagram
0.0- 1.0	18.98	10.7	0.106	123/ 983
1.0- 2.0	16.3	10.2	0.084	26/ 867
2.0- 3.0	17.5	10.3	0.029	44/ 895
3.0- 4.0	16.5	10.2	0.208	66/ 899
4.0- 5.0	16.6	10.2	0.134	52/ 865
5.0- 6.0	18.6	10.2	0.101	2/ 892
6.0- 7.0	18.5	10.1	0.035	2/ 899
7.0- 8.0	18.3	10.2	0.019	5/ 883
8.0- 9.0	19.4	10.2	0.099	6/ 898
9.0-10.0	19.5	10.1	0.102	2/ 899
0.0-10.0	18.01	10.24	0.086	328/8980

Table 4. TCP and UDP Flow-Tree topology with Multipath Routing Algorithm

C.T	TCP FLOW	UDP FLOW		
Interval (sec)	Bandwidth (Gbits/sec)	Bandwidth (Mbits)	Jitter (ms)	Lost/Total Datagram
0.0- 1.0	18.98	10.7	0.965	449/889
1.0- 2.0	16.3	10.2	3.208	21/ 867
2.0- 3.0	17.5	10.3	2.252	39/ 895
3.0- 4.0	16.5	10.2	0.803	75/ 899
4.0- 5.0	16.6	10.2	0.542	49/ 865
5.0- 6.0	18.6	10.2	0.412	22/ 892
6.0- 7.0	18.5	10.1	0.619	2/ 899
7.0- 8.0	18.3	10.2	1.369	12/ 883
8.0- 9.0	19.4	10.2	0.446	82/ 898
9.0-10.0	19.5	10.1	0.505	86/ 899
0.0-10.0	18.01	10.24	1.019	651/8886

5.3. TEST CASE 3: TREE TOPOLOGY

Controller, The system connects controller c0 to switches s1 through s9, facilitating node connections. The nodes communicate with each other via the switches and controllers. Fig. 5 displays the structure of the tree topology. Table 4 lists TCP and UDP flows after executing the tree topology in the data plane and running the multipath algorithm at the control plane with the POX controller. When the system executes the ping command from node 1 to node eight, it computes multiple paths, including one optimal path selected for transmission. Tree topology experiences congestion due to its architecture, and most packets queue at the root node. Tree topology experiences congestion due to its architecture, and most packets queue at the root node.

6. RESULT ANALYSIS

The simulation section notes the results of executing the multipath algorithm on Fat Tree, Custom, and Tree topologies. In this section, the quality-of-service parameters, which are throughput, jitter, and packet loss, will be calculated from the results.

6.1. THROUGHPUT

Throughput is measured in the context of the data rate at which data is successfully transmitted between source and destination or how many packets are delivered per second. This section discusses the Throughput of Fat Tree, Custom, and Tree topologies and their performances upon applying the multipath routing algorithm. The choice of topology plays a crucial role in identifying network performance.

6.1.1 Fat Tree Topology

This solution suits multipath connections between nodes best and organizations mainly use it in data centers requiring the highest throughput and crucial load balancing. Fat Tree is designed to produce high bandwidth by allowing traffic to distribute across multiple paths between dedicated nodes, reducing congestion and improving the service's overall quality. POX serves as a controller that effectively leverages multipath routing to maximize throughput. Table 4 shows the transfer rate of Fat Tree topology with ten intervals produced an average of 2.81GBytes of throughput with TCP flow. In the case of UDP, a throughput rate of 1.256 Mbytes of data is noted.

6.1.2 Custom Topology

Custom topology performance depends on the layout, how it supports multipath, and how it balances the load in various paths. Due to its adequate redundancy with path multiplicity, it achieves a high throughput, though less than Fat Tree's. If multiple paths are available between dedicated nodes, optimal performance is achieved by carefully designing configurations to eliminate bottlenecks and enhance throughput. Table 5 shows TCP flow and UDP flows noted over custom topology. TCP flow noted an average of 2.39 GBytes of throughput, whereas, in the case of UDP flow, it is 1.39 Mbytes

6.1.3 Tree Topology

Tree topology is more prone to bottlenecks and limited redundancy, so throughput is low compared to well-designed custom and fat-tree topologies

Near the root, congestion may occur. However, multipath routing will distribute traffic across available paths. However, there needs to be more path diversity, and it cannot utilize the benefits of multipath routing. There is a higher chance of congestion at the root, which limits the throughput. Table 6 displays the transfer rates of TCP and UDP flows, observing rates of 2.21 GBytes and 1.21 MBytes. Ultimately, we can conclude that the POX controller achieves better throughput with Fat Tree than with custom and tree topologies when applying multipath routing. The design of the Fat Tree itself supports multiple paths that are sometimes redundant and have high bandwidth. Suppose we design a custom topology with redundancy and multiple paths. In that case, a tree topology with minimal bottlenecks and careful path handling can also enable these topologies to produce high throughput. The TCP flow of the three topologies is compared in Fig. 6, while Fig. 7 displays the UDP flow.

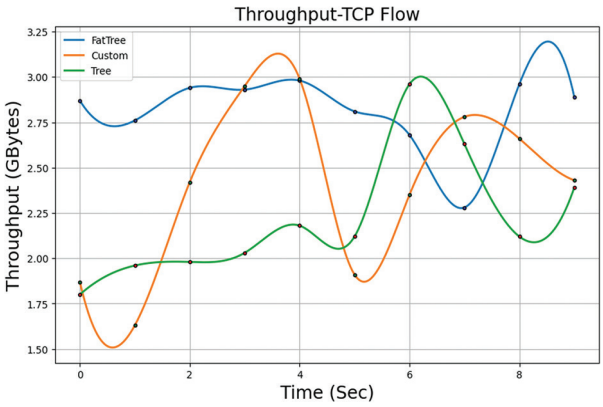

Fig 6. Throughput -TCP flow over various topologies

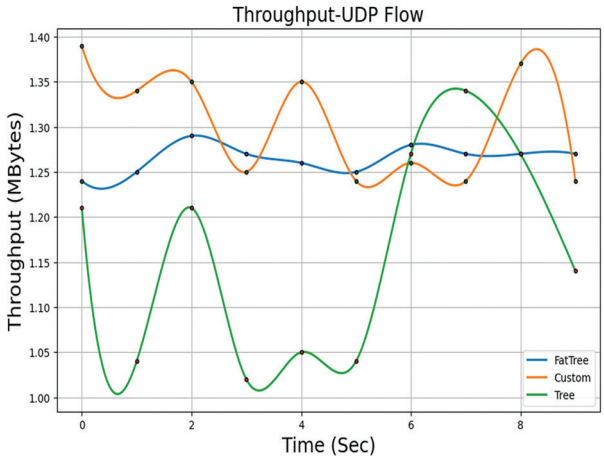

Fig 7. Throughput -UDP flow over various topologies

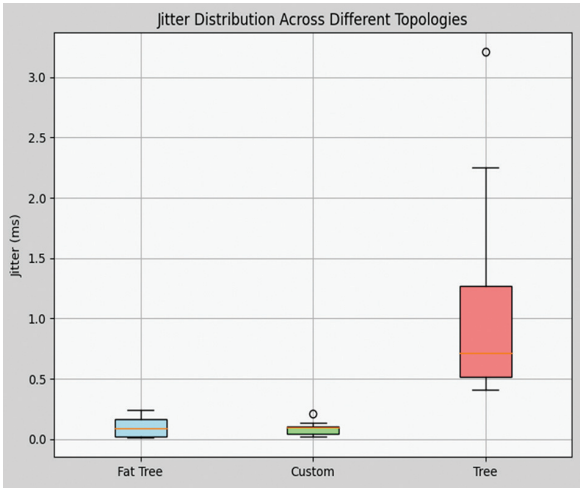

Fig 8. Jitter over various topologies

6.2. JITTER

Inconsistency of arrival time at the destination is known as jitter. Here, packets have irregular interval times. This variability we will denote as jitter. The main reasons for jitter may include congestion in the route, which may be due to load packets that may take different routes to reach the destination or a controller that introduces delays to process packets when it encounters enormous traffic. The POX controller is programmed and configured to manage the jitter. Path selection, dynamic path adjustment, and monitoring traffic influence the process. POX includes a real-time module that monitors traffic by maintaining a threshold for jitter. If it encounters massive traffic, the module reroutes the traffic to a less congested route to avoid congestion.

6.2.1 Fat-Tree Topology

This topology balances traffic by allowing it to flow on multiple paths in a balanced way, which reduces congestion on a single path. So, it produces less jitter than tree topology and custom topology. It also maintains consistent latency across paths, which minimizes jitter. The main reason is that packets are routed to the destination in a similar path length to synchronize packet delivery. POX controller avoids paths with high inconsistency. The fat tree structure maintains consistency in fluctuating traffic conditions, ensuring less jitter. Up on applying multipath routing on fat-tree, noted an average of 0.09ms of jitter.

6.2.3 Custom Topology

The design of a custom topology affects jitter. Careful topology design may yield less jitter. Path lengths and capacities produce varying jitters. Paths are balanced in terms of capacity and latency, which achieves low jitter. The POX controller performs traffic management and path selection to control traffic across paths to minimize jitter. We note an average of 0.86ms of jitter over ten intervals.

6.2.3 Tree Topology

As Tree topology suffers from limited path redundancy, the route node gets congested which leads to delays. Limited number of paths makes the queues for packets. Potential length differences can introduce inconsistency in packet delivery times, resulting in higher jitter. Managing jitter is more challenging due to the limited path range and latent congestion. The POX controller may struggle to maintain consistent path performance, leading to inconstant latency and increased jitter. Table V shows an average of 1.09 ms of jitter. Jitter is managed if multiple paths between source and destination are the same length. Among all topologies, the custom topology achieves less jitter, though the fat tree is designed well. Figure 8 shows a comparison of fat tree, custom, and tree topology.

6.3. PACKET DELIVERY

The packet delivery ratio is calculated from the number of packets lost to send. To find the number of packets delivered, subtract the lost packets from the sent packets. Measuring the network performance will be helped by a smaller number of lost packets. Traffic congestion and link failures are a few reasons for packet loss. Buffering also may lead to packet loss. To achieve less packet loss with the POX controller, the critical factor is to choose a topology with redundancy, and the controller should able to manage traffic over multiple paths

6.3.1 Fat-Tree Topology

As it has high redundancy and availability of multiple paths between any pair of nodes, it helps to distribute traffic more evenly across paths, which reduces congestion. In case of link failure, redundancy will help to minimize packet loss. Multipath routing prevents load on a single path and balances by enrooting to another path, which reduces packet loss. Path diversity in fat-tree topologies helps mitigate packet loss even under varying traffic conditions. Table 6 shows a packet delivery ratio of 10 intervals with 148 packets lost over 9085 packets, resulting in a 1.62 % packet loss ratio.

6.3.2 Custom Topology

Packet loss in custom topology depends on design with adequate redundancy, and a balanced path can achieve less packet loss. However, its limitations with redundancy suggest that multipath routing could be more effective in foreseeing packet loss. The configuration of the POX controller effectively utilizes multipath routing, although performance depends on traffic patterns and the network design. Still applying multipath routing algorithm on custom topology acquired a loss ratio of 3.65%, where 328 packets were lost during the transmission of 8980 packets over ten intervals from source to destination.

6.3.3 Tree Topology

Tree topology mostly has a problem of enormous packet loss due to a hierarchical structure where traffic is congested at the root, which increases packet loss when the network hits heavy traffic. Multipath routing helps distribute the traffic over alternative paths, significantly reducing packet loss, though lack of path diversity and redundancy limits its effectiveness. Multiple paths may exist, but they differ in factors like Capacity and not equal size.

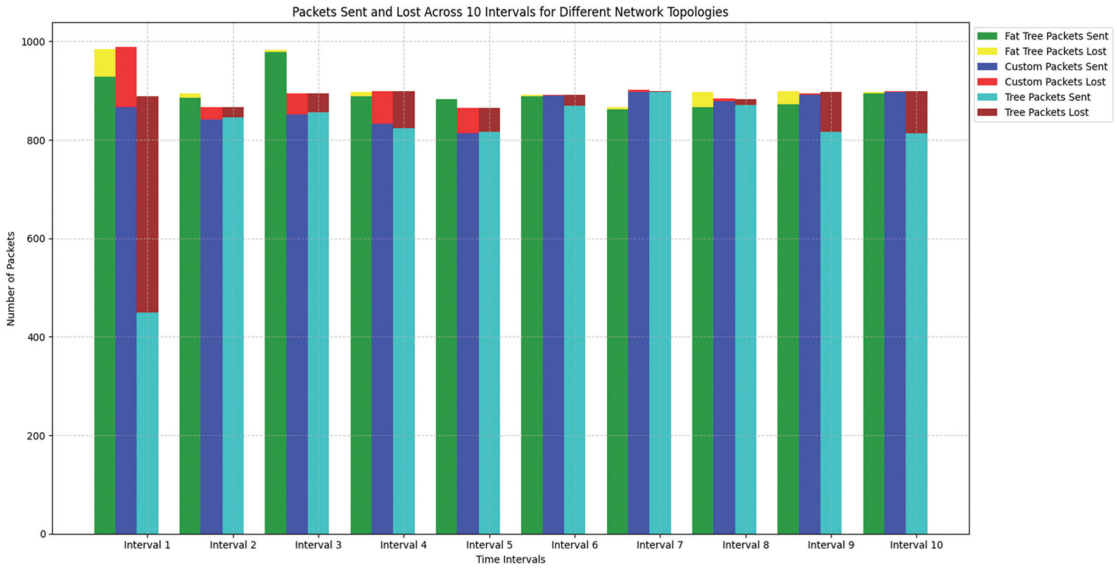

Fig 9. Packet delivery over Fat-Tree, Custom, Tree

The POX controller may need to help distribute traffic efficiently enough to avoid these losses. So, it witnessed more packet loss than the other topologies. During ten intervals, the transmission lost 651 out of 8886 packets, producing a 7.32% packet loss ratio. Figure 9 shows comparisons of fat-tree, custom, and tree topologies regarding packet loss and packets sent.Fat-tree topology showed better performance, as shown in Figure 9. The bar chart over interval 1 to interval ten shows packet loss, where each bar represents the packets delivered at the bottom and the top.at interval one, the tree topology had a packet loss of 440 packets and sent 449 packets. Later, they reduced the packet loss percentage in the tree topology. All three topologies sent packets without a considerable loss in a few intervals. In the end, fat-tree performed well when multipath routing was applied with the POX controller over topology with quality-of-service parameters like throughput, jitter, and packet loss ratio.

7. CONCLUSION

Software Defined Network (SDN) enable developers to build SDN applications due to architecture compatibility, which separates the control plane and data plane and allows centralized network management with programmability. POX controller is an open-source and Python-based controller that works with MiniNet as it supports Python-based coding to create a network. Congestion is the most significant problem; We need immediate solutions to avoid data loss. Selecting the optimal path repeatedly between dedicated nodes may lead to overload and congestion. Multipath routing has been introduced in SDN to address this problem. It identifies multiple paths between all nodes with Depth First Search and selects one path among available paths. If the selected path becomes congested, the system will reroute packets to the following path. This action can reduce congestion, and tested this algorithm on Fat-Tree, custom, and Tree topologies using the POX controller.

Throughput, jitter, and packet loss ratio are the parameters used to measure the performance of the multipath routing algorithm. Fat-Tree showed improved performance over remaining topologies due to redundancy and path diversity, which are limited in remaining topologies. If a custom topology is designed well, then this can also produce better throughput. In the case of a tree topology, the root itself is getting congested, so packet loss is more with tree topology. In the

future, the Multipath routing algorithm is going to be tested with the Ryu controller.

This work helps the researcher who wants to work with optimal path identification over various topologies and load balancing over the POX controller, as well as with POX and various topologies. Fat-tree topology can be selected when the choice of performance metric is throughput and less packet loss. They can even test custom topology with proper design. Tree topology clearly shows the occurrence of congestion at the root itself.

8. REFERENCES

[1] A. Ram, S. K. Chakraborty, "Analysis of Software-Defined Networking (SDN) Performance in Wired and Wireless Networks Across Various Topologies, Including Single, Linear, and Tree Structures", Indian Journal of Information Sources and Services, Vol. 14, No. 1, 2024, pp. 39-50.

[2] Y. Zhang, M. Chen, "Performance evaluation of Software-Defined Network (SDN) controllers using Dijkstra's algorithm", Wireless Networks, Vol. 28, 2022, pp. 3787-3800.

[3] J. Ma, R. Jin, L. Dong, G. Zhu, X. Jiang, "Implementation of SDN traffic monitoring based on Ryu controller", Proceedings of the International Symposium on Computer Applications and Information Systems, 19 May 2022.

[4] N. Gupta, M. S. Maashi, S. Tanwar, S. Badotra, M. Aljebreen, S. Bharany, "A Comparative Study of Software Defined Networking Controllers Using Mininet", Electronics, Vol. 11, No. 17, 2022, p. 2715.

[5] T. H. Obaida, H. A. Salman, "A novel method to find the best path in SDN using firefly algorithm", Journal of Intelligent Systems, Vol. 31, No. 1, 2022, pp. 902-914.

[6] M. C. Saxena, M. Sabharwal, P. Bajaj, "Exploring path computation techniques in Software-Defined Networking: A review and performance evaluation of centralized, distributed, and hybrid approaches", International Journal on Recent and Innovation Trends in Computing and Communication, Vol. 11, No. 9s, 2023, pp. 553-567.

[7] M. Caria, A. Jukan, M. Hoffmann, "SDN partitioning: A centralized control plane for distributed routing protocols", IEEE Transactions on Network and Service Management, Vol. 13, No. 3, 2016, pp. 381-393.

[8] D. R. Akbi, W. Suharso, "A comparison of Ryu and Pox controllers: A parallel implementation", Journal of Intelligent Systems, Vol. 9, No. 1, 2024, pp. 1-9.

[9] E. Adedokun, A. O. Adesina, O. O. Olabiyisi, "Improved extended Dijkstra's algorithm for software defined networks", Proceedings of the International Conference on Computing, Networking and Informatics, Lagos, Nigeria, 2017, pp. 1-6.

[10] "POX controller manual current documentation", https://noxrepo.github.io/pox-doc/html/ (accessed: 2022)

[11] B. Lantz, N. Handigol, B. Heller, V. Jeyakumar, "Introduction to Mininet", Mininet Project, https://github.com/mininet/mininet/wiki/Introduction-to-Mininet (accessed: 2022)

[12] R. Mohammadi, A. Nazari, M. Nassiri, M. Conti, "An SDN-based framework for QoS routing in Internet of Underwater Things", Telecommunication Systems, Vol. 78, No. 2, 2021, pp. 253-266.

[13] M. I. Salman, "A hybrid SDN-multipath transmission for a reliable video surveillance system", Association of Arab Universities Journal of Engineering Sciences, Vol. 29, No. 2, 2022, pp. 46-54.

[14] R. Patel, N. Gupta, "Comparative evaluation of SDN controllers: POX, Ryu, and OpenDaylight", IEEE Access, Vol. 11, 2023, pp. 29015-29028.

[15] J. Smith, L. Zhang, "Enhancements in POX SDN topologies for improved network management", Journal of Network and Computer Applications, Vol. 75, No. 1, 2023, pp. 45-56.

[16] I. Koulouras, S. V. Margariti, I. Bobotsaris, E. Stergiou, C. Stylios, "Assessment of SDN controllers in wireless environments using a multi-criteria technique", Information, Vol. 14, No. 9, 2023.

[17] A. Wilson, C. Lee, "Advancements in Ryu SDN controller for scalable network topologies", IEEE Transactions on Network and Service Management, Vol. 19, No. 4, 2022, pp. 789-800.

[18] L. Zhu, M. M. Karim, K. Sharif, C. Xu, F. Li, X. Du, M. Guizani, "SDN controllers: A comprehensive analysis and performance evaluation study", ACM Computing Surveys, Vol. 53, No. 6, 2020.

[19] S. Lee, K. Park, "Ryu SDN framework: Design and performance", IEEE Transactions on Network and Service Management, Vol.17, No. 2, 2020, pp. 123-135.

[20] P. E. Numan, K. M. Yusof, M. N. B. Marsono, S. K. S. Yusof, M. H. B. M. Fauzi, S. Nathaniel, M. A. B. Baharudin, "On the latency and jitter evaluation of software defined networks", Bulletin of Electrical Engineering and Informatics, Vol. 8, No. 4, 2019, pp. 1507-1516.

[21] A. S. Sajid, S. F. N. Niloy, K. Hossain, T. Rahman, "Comprehensive evaluation of shortest path algorithms and highest bottleneck bandwidth algorithm in software-defined networks", Report, Department of Computer Science and Engineering, BRAC University, Bangladesh, 2018.

[22] J. P. Duque, D. D. Beltrán, 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, 2018, pp. 348-357.

[23] N. Farrugia, V. Buttigieg, J. Briffa, "A globally optimized multipath routing algorithm using SDN", in Proceedings of the IEEE International Conference on Innovation Networking and Services, Paris, France, 19-22 February 2018, pp. 1-8.

[24] A. Abdul-hafiz, E. A. Adedokun, S. Man-Yahya, "Improved extended Dijkstra's algorithm for software defined networks", International Journal of Applied Information Systems, Vol. 12, No. 8, 2017, pp. 22-26.

[25] I. Z. Bholebawa, U. D. Dalal, "Design and performance analysis of OpenFlow-enabled network topologies using Mininet", International Journal of Computer Communication Engineering, Vol. 5, No. 6, 2016, pp. 419-429.

[26] W. Yahya, A. Basuki, J. R. Jiang, "The extended Dijkstra's-based load balancing for OpenFlow network", International Journal of Electrical and Computer Engineering, Vol. 5, No. 2, 2015, pp. 289-296.

[27] X. Zhang, Y. Lu, Q. Wu, "Tree-based topology design in software-defined networks", Proceedings of the IEEE Global Communications Conference, 2015, pp.1-6.

[28] A. L. Stancu, S. Halunga, A. Vulpe, G. Suciu, O. Fratu, E. C. Popovici, "A comparison between several software defined networking controllers", Proceedings of the 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services, Nis, Serbia, 14-17 October 2015, pp. 223-226.

[29] N. Naimullah, M. Imad, M. Hassan, M. Afzal, S. Khan, A. Khan, "POX and RYU controller performance analysis on Software Defined Network", EAI Endorsed Transactions on Internet of Things, Vol. 9, 2023.

[30] M. N. A. Sheikh, I.-S. Hwang, M. S. Raza, M. S. Ab-Rahman, "A qualitative and comparative performance assessment of logically centralized SDN controllers via Mininet emulator", Computers, Vol. 13, No. 4, 2024, p. 85.

[31] I. Koulouras, I. Bobotsaris, S. V. Margariti, E. Stergiou, C. Stylios, "Assessment of SDN Controllers in Wireless Environment Using a Multi-Criteria Technique", Information, Vol. 14, 2023, p. 476.

[32] Y. Wang, L. Li, Y. Zhao, "Optimizing tree-based topologies in SDN: Techniques and applications", Computer Communications, Vol. 217, 2023, pp. 10-22.

[33] Y. Chen, J. Lee, "Integrating Dijkstra's algorithm with the POX SDN controller for network optimization and path computation", Computer Networks, Vol. 211, 2023, p. 108160.

[34] D. Cabarkapa, D. Rancic, "Performance analysis of Ryu-POX controller in different tree-based SDN topologies", Advances in Electrical and Computer Engineering, Vol. 21, No. 3, 2021, pp. 47-56.

[35] S. Sryheni, "Introduction to depth first search algorithm (DFS)", Baeldung on Computer Science, www.baeldung.com (accessed: 2023)

[36] "Mininet commands", available online: http://mininet.org/ (accessed: 2022)

[37] K. K. Sharma, M. Sood, "Mininet as a container-based emulator for software defined networks", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 4, 2014, pp. 681-685.

[38] K. K. Sharma, M. Sood, "Mininet as a container-based emulator for software defined networks", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 4, 2014, pp. 681-685.