SSRG International Journal of Electrical and Electronics Engineering
ISSN: 2348-8379/ https://doi.org/10.14445/23488379/IJEEE-V11I5P127

Volume 11 Issue 5, 299-305, May 2024

Original Article

Shortest Path Forwarding in Software-Defined Networks Using RYU Controller

Kishan P. Patel[1], Jıtendra P. Chaudhari[2], Hiren K. Mewada[3,] Hardik S. Jayswal[4], Rajeshkumar V. Patel[5], Dnyaneshwar K. Kirange[6]

[1]*Department of Electrical Engineering, CSPIT, Charotar University of Science and Technology, Gujarat, India.*
[2]*Charusat Space Research and Technology Centre, E&C Engineering Department, CSPIT, Charotar University of Science and Technology, Gujarat, India.*
[3]*Department of Electrical Engineering, Prince Mohammad Bin Fahd University, Al Khobar, Kingdom of Saudi Arabia.*
[4,5]*Department of Information Technology, Devang Patel Institute of Advance Technology and Research, Charotar University of Science and Technology, Gujarat, India.*
[6]*Department of Computer Engineering, SSBT's College of Engineering and Technology, Maharashtra, India.*

[2]*Corresponding Author : jitendrachaudhari.ec@charusat.ac.in*

Received: 16 March 2024 Revised: 16 April 2024 Accepted: 14 May 2024 Published: 29 May 2024

Abstract *- The shortest path forwarding is not provided by OpenFlow. The benefit of using OpenFlow is that programmers can control the network devices by writing different applications. This research paper deals with the design of the shortest path algorithm using the RYU controller and OpenFlow. The datacenter topology with different network sizes is used for evaluating various shortest-path algorithms. In this work, the RYU controller's basic switch application is used. The network application is divided into three parts, namely topology discovery, network view construction and forwarding. Mininet is used as an emulator with an RYU controller for SDN. The performance depicts that Dijkstra's algorithm gives better throughput as compared to other shortest-path algorithms under consideration during this study.*

Keywords *- Dijkstra's shortest path algorithm, Mininet, OpenFlow, RYU controller, SDN.*

1. Introduction

The data and control plane are combined in a typical network. The control plane of the devices is responsible for providing information regarding the forwarding table. Decisions regarding sending or receiving frames are taken by the network device based on the forwarding table [1]. Figure 1 shows the control plane and data plane for a traditional network.

SDN offers the following benefits over traditional networks:

- Reduction of time for managing the network and deployment of new resources or applications [3].
- Programmable Network: Applications can be linked to the network using open APIs. Traditional networks lack a standard set of APIs [4]. As a result, programming applications directly to network resources is quite challenging.
- Flexible network: A fault-tolerant network can be built using SDN [5]. It can be configured from the central location. In case if primary link fails, the network connectivity can be restored using a backup link [6].

The separate control plane and data plane of SDN allow easy creation and introduction of new abstractions in networking. It simplifies the network management and facilitates network evolution [7, 8]. SDN enables the use of software to automate tasks and enhances the ability to modify network policies dynamically [9]. SDN controller manages the flow in the data plane. SDN controllers are based on protocols like OpenFlow. This protocol is responsible for forwarding packets through switches and routers [10].

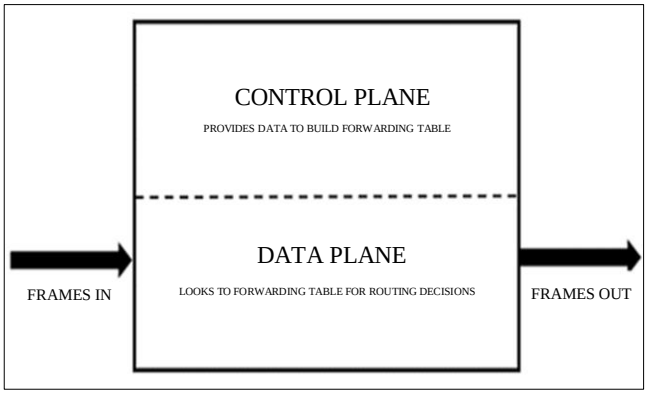

Fig. 1 Traditional network

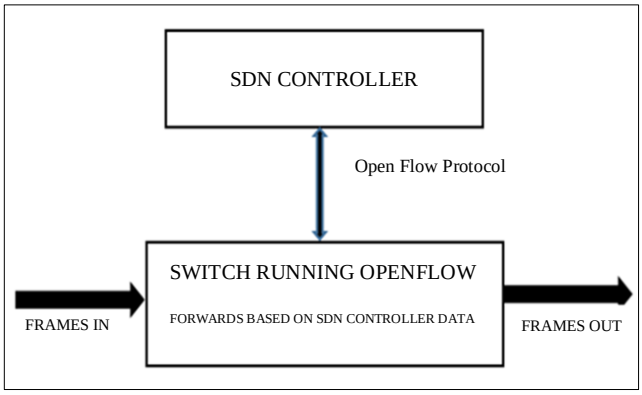

Fig. 2 SDN framework

One of the open-source SDN controllers written in Python is RYU. Software components of RYU with well-defined API allow the creation of various network management and control applications. SDN is an innovative networking paradigm that separates the network's control plane from the data plane, enabling centralized control and programmability.

RYU provides a flexible and scalable platform for developing SDN applications, allowing network administrators to define and enforce network policies, orchestrate network resources, and implement various network functions. RYU is responsible for the creation and sending an OpenFlow messages, listening to asynchronous events and handling incoming packets [11].

Shortest path forwarding is an important requirement in Software-Defined Networking (SDN) because it allows for efficient and optimal routing of network traffic. By determining the shortest path between source and destination nodes, SDN can minimize the number of network hops and reduce latency.

This leads to improved network efficiency and faster data transmission shortest path forwarding is not performed by simple_switch.py of RYU. Therefore, we modified this file, integrating the shiftest path forwarding to this file. For the implementation of shortest path forwarding, the performed steps are as follows.

- Topology discovery
- Network view construction
- Forwarding

This study aims to implement the datacenter topology using the Mininet emulator. Using the RYU controller, track the effectiveness of the datacenter topology for various network sizes. Here, the modification is done in the simple_switch.py of RYU for shortest path forwarding. If any link fails, the packets will be forwarded along the next shortest path of SDN. The overall contribution of the paper is as follows:

- The study contributes by implementing a datacenter topology using the Mininet emulator. This allows researchers and practitioners to simulate and evaluate network scenarios resembling real-world datacenter environments.
- The study utilizes the RYU controller to evaluate the effectiveness of the implemented datacenter topology across various network sizes. This evaluation helps assess the performance and scalability of the topology under different conditions.
- The researchers modify the "simple_switch.py" module of the RYU controller to incorporate shortest path forwarding. By implementing this modification, the study enhances the controller's capabilities to compute and enforce optimal routing paths based on the shortest path algorithm.

Section 2 presents basic requirements for the shortest path forwarding algorithm implementation. Later, the experiment setup and its results are presented in Sections 3 and 4. Finally, a conclusion is presented in Section 5.

2. Preliminaries

2.1. Dijkstra's Shortest Path Algorithm

The Dijkstra algorithm [12] is used to determine the most efficient routes between nodes in a graph, such as a network of roads. The concept was conceived by scientist Edsger W. Dijkstra in 1956 and subsequently published three years later.

Algorithm: The work started with the Initial node as the node to begin with. Any node's distance will be measured from the starting node. There will be some initial distance values assigned to all nodes, and these values will be improved at each step.

1. All nodes are initially designated as unvisited. An unvisited set refers to a collection of all unvisited nodes set.
2. Each node is assigned a tentative distance value. The initial value of the first node is zero, while all subsequent nodes are assigned a value of infinity. The current node is designated as the initial node.
3. Now considering the current node, get all its unvisited nodes and get the distance from the current node. Update the distance values of the nodes by comparing the newly computed distance and assigning the smaller one. For example, consider node X with an initial distance of 6, and if the neighboring node Y has a distance of 2, then the length of the path to Y through X will be 6+2 =8. Now, compare this value with the previously assigned value. If the newly computed path cost is less, then update it to 8. Otherwise, the current value of the path is kept as it is.
4. The present node is marked as visited and deleted from the unvisited set after all of its unvisited neighbours have been visited. A node will never be checked again after being visited.

5. The algorithm will stop after marking the destination node as visited or after all nodes in the unvisited list are visited.
6. Otherwise, get the new current node from the unvisited node having the smallest tentative distance and go back to step 3.

2.2. Bellman-Ford Algorithm

In this algorithm, each node can have only partial knowledge of the network. The node must know its number only, and each node should be able to compute the number of its neighbours. To calculate the shortest path from the current node to the destination, hope is done neighbor by neighbor from each source to the destination [13].

2.3. Floyd-Warshall Algorithm

This algorithm can locate the shortest route on a graph with edge weights that are either positive or negative [14]. The solution matrix is initiated the same as the input graph matrix. Then, all vertices are taken into account as intermediate vertices, updating the solution matrix. All vertices are updated gradually, and the shortest path is one by one.

Here, the current vertex is assigned to the shortest path as an intermediate vertex. After picking vertex k as an intermediate vertex, it is assumed that it is already considered that all vertices from 0 to k-1 are intermediate vertices. There are two possible situations for each pair (i,j). The route from i to j does not include K. K is not an intermediate vertex in the scenario. Dist[i][j] remains unchanged.

The distance value of dist[i][j] is updated and computed as dist[k][j] + dist[i][k] if k is on the shortest route from I to j. If dist[i][j] > dist[k][j] + dist[i][k].

2.4. A star Algorithm

A* is a best-first search or an informed search algorithm. A* algorithm works in terms of weighted graphs [15]. A specific node from the graph is marked as a start node. Finding the shortest path between the starting node and the goal node is the algorithm's main objective [16]. This is achieved by maintaining a tree of paths from the initial node to the destination.

Algorithm:

1. Open list is initiated.
2. Closed list is initiated. The initial node is put on the open list by leaving its f at 0.
3. Keep going until the open list is not empty
   a. The node with the lowest f is found on the open list. Call it "q".
   b. Take q out of the open list.
   c. Locate the parents of 8 successors of q and set them to q.
   d. Repeat for each successor
4. If a successor is a target, cease looking.

   successor.g = distance (successor, q) + q.g
   successor.h = distance (goal, successor)
   successor.f = successor.h + successor.g
5. If the node has a lower f than the successor is in the open list and occupies the same place as the successor, avoid the successor.
6. If the closed list node in the same place as the successor has a low f value, do not include it. The node is moved to the bottom of the open list.
7. Send q to the closed list.

2.5. Mininet

Mininet emulator [17] provides a simulation for a complete network of hosts, links, and switches. It is very easy to prototype OpenFlow-based network controllers in Mininet. It is possible to run any code on the Mininet host, which any Linux server can run. It is feasible to build a Mininet host that only sees its interfaces and has a private network interface. MiniNet allows rapid prototype creation with constrained resources.

For network namespaces and virtualization features, it is light. It also enables real-time debugging. OpenFlow switches, sometimes referred to as open vSwitches, serve as the forwarding devices [18]. Open vSwitch in the Mininet, as well as OpenFlow reference switch in the Mininet, are software based switches. It is possible to use any SDN-based controller in Mininet [19].

2.6. RYU Controller

One of the most flexible open-source SDN controllers is the RYU controller. The controller increases the agility of the network. Using the RYU controller makes the network easy to manage and easy to adapt to traffic flow. SDN controllers are called as brains of the network because they are responsible for communication between routers and switches and southbound APIs and using northbound APIs, also up to the applications and business logic [20, 21]. Many SDN-based interference mitigation strategies, including ML-based interference mitigation and NS in 5G, are implemented using RYU [22].

Software components having well-defined APIs are provided by the RYU controller [23]. Due to these API developers can easily create a variety of network management and control applications. Due to these modular programming organizations are able to fulfill their specific needs by deployment. Developers can quickly upgrade old modules and construct new applications using these components, guaranteeing that the underlying network can support the varying requirements of their applications.

3. Experimental Setup

To conduct this experiment, we built a data centre topology with various numbers of racks. The RYU

controller's simple_switch.py application has been altered. The application is divided into three steps: (1) Discovery of network topology, (2) Construction of the network view, and (3) shortest path forwarding.

For the discovery of network topology, the work uses a RYU module called topology. Networkx is the Python graph library providing different shortest-path algorithms. This library is used for the construction of the network view and shortest path computation. OpenFlow is used for forwarding.

3.1. Networkx for Network View and Shortest Path Computation

NetworkX is a Python module for building, modifying, and studying the dynamics, structure, and complex network functions [24].

Command to create an edge-free, empty graph.

Import networkx as nx

G = nx.Graph()

G.add_node() is used for adding the nodes in the network, and G.add_edge() for adding the edges in the network.

Networkx provides with various functions for shortest path computations

shortest_path(G, source=None, target=None, weight=None, method='Dijkstra')

The function determines the shortest route between two points. If the weight is None, then every edge has assigned the weight as 1. The default method for shortest path computation is Dijkstra's. Bellman-Ford method is also supported.

floyd_warshall(G, weight='weight')

This function extracts the graph's shortest pathways for all pairs.

astar_path(G, source, target, heuristic=None, weight='weight')

Using the a* algorithm, this function determines the shortest route between the target and the source. Heuristic in this case, is a function to assess the estimated distance between a node and the goal.

3.2. Mininet and RYU Controller

In this work, the RYU application is developed to find the shortest path in the SDN network. The application is named as shortestpath.py. The command to run the RYU application is:

PYTHONPATH=../bin/ryu-managerryu/app/shortestpath.py –observe-links

The datacenter topology script is written in Python using the Mininet emulator. The command to run custom Mininet topology is:

Sudo mn – custom datacenter.py –topo mytopo – controller remote

3.3. Iperf for Evaluating the Performance of the Network

Iperf is used for basic performance evaluation over mininet.

Open windows for h1 and h2 using xterm.

Initiate the Transmission Control Protocol (TCP) server on host h2, utilizing port (-p) 5566. Additionally, continuously observe the outcomes at a frequency of one second (-i).

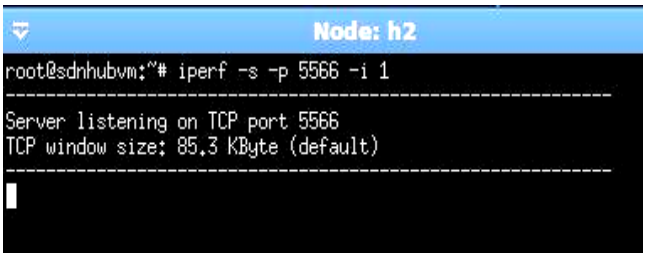

Fig. 3 TCP server

Initiate the TCP client on host h1 using the -c option. Additionally, please adjust the transmission duration (-t) to a precise duration of 15 seconds.

Fig.4 TCP client

Observe results for host h1 as shown in Figure 5. It shows that between 0 to 15 seconds, 14.2 Gbps average throughput is received.

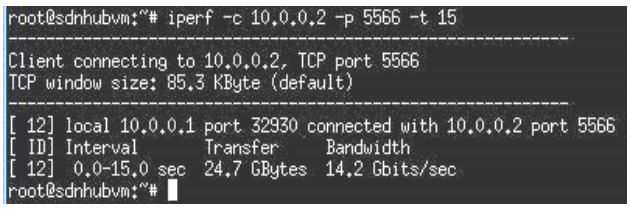

Fig. 5 Throughput

4. Performance Measure

Software tool Mininet is used as an emulator and the RYU controller of SDN is used as a remote controller in our experimentation. Our main goal is to use the shortest path technique to improve network throughput.

This work uses an implementation of datacenter network topology in Mininet and Python. Also, the network library is used for writing shortest-path algorithms in the RYU application. The iperf is used to measure the TCP throughput. OpenFlow messages are sent between the nodes using RYU handlers and decorators by the SDN controller. The h1r1 is configured in client mode to carry out the TCP packet transfer between two hosts, namely h1r1 and h10r10. TCP packets are sent to the client node.

H10r10 is set to server mode. The server calculates the bandwidth for the quantity of data transferred after receiving the TCP packets. During this experimentation, the packets are transmitted from client to server for 15 seconds, and bandwidth is observed. In the network, the bandwidth of the data decides the performance of the packet transmission from source to destination.

The throughput tests are carried out on different shortest path algorithms, including Bellman-ford Algorithm, Floyd-Warshall Algorithm, Dijikstraw's Shortest Path Algorithm, and A* shortest path algorithm. Also, the results are compared with the simple_switch.py application of RYU, which does not include shortest path computation. The data centre topology evaluates the performance for different network scales. The throughput measured for all shortest paths under consideration is depicted in Table 1.

As depicted in Table 1, Dijikstraw's algorithm gives an average better throughput of 7.293 as compared to other shortest path algorithms. Figure 6 depicts the comparative performance evaluation of different shortest path algorithms for the datacenter topology. From the results, it has been observed that the shortest path forwarding ensures that network traffic follows the most efficient routes between source and destination nodes. By computing the shortest paths using Dijkstra's algorithm, the implementation optimizes the routing decisions made by the network devices. This reduces the number of network hops, minimizes latency, and improves overall network performance.

Table 1. Throughput comparison for different algorithm

Data Center Scales	Shortest Path Algorithms				
	Simple Switch	Dijikstraw's Algorithm	Bellman-Ford Algorithm	Floyd-Warshall Algorithm	A Star Algorithm
4	5.94	8.23	6.38	3.56	3.7
5	7.67	5.72	7.72	3.72	4.11
6	6.84	7.74	5.75	3.69	3.81
7	7.01	8.26	8.21	3.82	3.84
10	7.32	7.68	7.82	7.72	3.86
12	7.09	6.31	5.36	6.8	3.61
14	7.15	7.56	5.54	6.58	3.36
16	6.4	6.98	3.24	7.43	3.63
18	6.18	7.11	3.46	8.51	3.67
20	5.75	7.34	7.07	6.51	3.75
Average	6.735	7.293	6.055	5.834	3.734

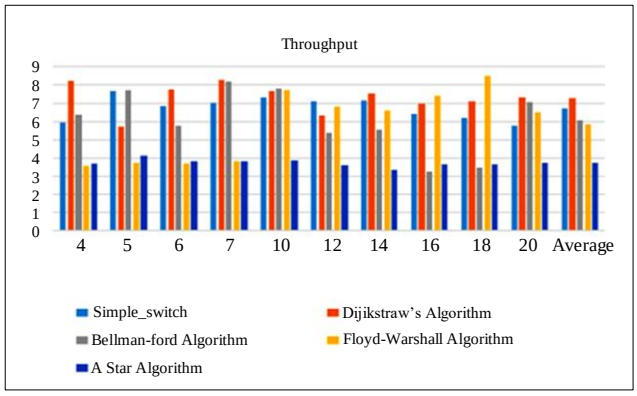

Fig. 6 Performance evaluation for different datacenter topology scales

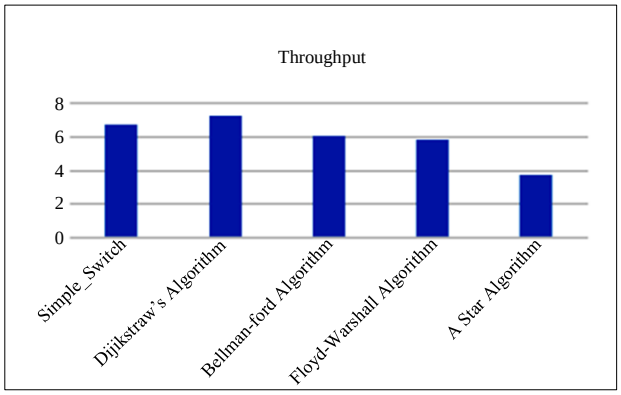

Fig. 7 Average network throughput

5. Conclusion

In this paper, different shortest-path algorithms have been implemented in a Software Networking environment using Mininet and RYU. Software-defined networking is a relatively new field, yet it is expanding quickly. Simple_switch.py application of RYU does not support shortest path computation.

Hence, in this paper, the RYU application is modified by incorporating Dijkstra's SP Algorithm, Bellman-ford Algorithm, Floyd-Warshall Algorithm and shortest path algorithm A* from the network library. A mininet-based Python script is written for generating datacenter topology. For different scales of data centre topology, the calculation of the performance of the SDN network with regard to throughput is carried out in this work.

The Dijkstra's' Shortest path algorithm gives a better average throughput as compared to other shortest path algorithms under consideration. In future, the performance evaluation can be done by using more performance measures, including packet transfer rate, bandwidth utilization, packet loss ratio etc. Also, an attempt can be made to investigate dynamic routing using the shortest path in case of any link failure. The comparative performance of average throughput is depicted in Figure 7.

References

[1] Nachikethas A. Jagadeesan, and Bhaskar Krishnamachari, "Software-Defined Networking Paradigms in Wireless Networks: A Survey," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, pp. 1-11, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[2] Murat Karakus, and Arjan Durresi, "Quality of Service (QoS) in Software-Defined Networking (SDN): A Survey," *Journal of Network and Computer Applications*, vol. 80, pp. 200-218, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[3] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui, "Software-Defined Networking (SDN): A Survey," *Security and Communication Networks*, vol. 9, no.18, pp. 5803-5833, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[4] Kannan Govindarajan, Kong Chee Meng, and Hong Ong, "A Literature Review on Software-Defined Networking (SDN) Research Topics, Challenges and Solutions," *Fifth International Conference on Advanced Computing (ICoAC)*, Chennai, pp. 293-299, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[5] A.U. Rehman, Rui. L. Aguiar, and Joao Paulo Barraca, "Fault-Tolerance in the Scope of Software-Defined Networking (SDN)," *IEEE Access*, vol. 7, pp. 124474-124490, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[6] Katayoun Bakhshi Kiadehi, Amir Masoud Rahmani, and Amir Sabbagh Molahosseini, "A Fault-Tolerant Architecture for Internet-of-Things Based on Software-Defined Networks," *Telecommunication Systems*, vol. 77, pp. 155-169, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[7] Junjie Xie et al., "Control Plane of Software-Defined Networks: A Survey," *Computer Communications*, vol. 67, pp. 1-10, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[8] Diego Kreutz et al., "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14-76, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[9] Rajat Chaudhary et al., "A Comprehensive Survey on Software-Defined Networking for Smart Communities," *International Journal of Communication Systems*, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[10] Josep Batalle et al., "On the Implementation of NFV Over an OpenFlow Infrastructure: Routing Function Virtualization," *IEEE SDN for Future Networks and Services (SDN4FNS)*, Italy, pp. 1-6, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[11] Danijel Cabarkapa, and Dejan Rancic, "Performance Analysis of RYU-POX Controller in Different Tree-Based SDN Topologies," *Advances in Electrical & Computer Engineering*, vol. 21, no. 3, pp. 31-38, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[12] David Walden, The Bellman-Ford Algorithm and Distributed Bellman-Ford, pp. 1-12, 2005. [Online]. Available: https://www.walden-family.com/public/bf-history.pdf

[13] Dorit S. Hochbaum, Lecture Notes for IEOR 266: Graph Algorithms and Network Flows. [Online]. Available: https://hochbaum.ieor.berkeley.edu/files/266Notes-F2020.pdf

[14] Vassilis Kaffes et al., "Finding Shortest Keyword Covering Routes in Road Networks," *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, pp. 1-12, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[15] Ouardi Amine, and Mestari Mohammed, "Generating A-Star Algorithm Admissible Heuristics Using a Dynamic Dataloader on Neural Networks, Enhanced with Genetic Algorithms, on a Distributed Architecture," *IEEE Access*, vol. 11, pp. 18356-18373, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[16] Dian Rachmawati, and Lysander Gustin, "Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem," *Journal of Physics: Conference Series*, vol. 1566, no. 1, pp. 1-8, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[17] Bob Lantz, and Brian O'Connor, "A Mininet-Based Virtual Testbed for Distributed SDN Development," *ACM SIGCOMM Computer Communication Review*, vol. 45, no.4, pp. 365-366, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[18] Syed Hussain Ali Kazmi et al., "Routing-Based Interference Mitigation in SDN Enabled Beyond 5G Communication Networks: A Comprehensive Survey," *IEEE Access*, vol. 11, pp. 4023-4041, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[19] Daniela Sousa, Susana Sargento, and Miguel Luis, "A Simulation Environment for Software Defined Wireless Networks with Legacy Devices," *Proceedings of the 18th ACM International Symposium on QoS and Security for Wireless and Mobile Networks*, pp. 1-10, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[20] Saleh Asadollahi, Bhargavi Goswami, and Mohammed Sameer, "RYU Controller's Scalability Experiment on Software-Defined Networks," *2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC)*, India, pp. 1-5, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[21] Md. Tariqul Islam, Nazrul Islam, and Md. Al Refat, "Node to Node Performance Evaluation through RYU SDN Controller," *Wireless Personal Communications*, vol. 112, pp. 555-570, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[22] Mohammad Nowsin Amin Sheikh et al., "A Qualitative and Comparative Performance Assessment of Logically Centralized SDN Controllers by Mininet Emulator," *Computers*, vol. 13, no. 4, pp. 1-27, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[23] Himanshi Babbar, and Shalli Rani, "Performance Evaluation of QoS Metrics in Software-Defined Networking Using RYU Controller," *IOP Conference Series: Materials Science and Engineering*, vol. 1022, no. 1, pp. 1-12, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[24] Aric Hagberg, Pieter J. Swart, and Daniel A. Schult, "Exploring Network Structure, Dynamics, and Function Using NetworkX," *Conference: SCIPY 08*, 2008. [Google Scholar] [Publisher Link]