An Efficient Routing Algorithm to Improve the Performance of Floodlight SDN Controllers

Yohanis Wolde Fikremariam

A Thesis Submitted to

The department of Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

March, 2023

Adama, Ethiopia

An Efficient Routing Algorithm to Improve the Performance of Floodlight SDN Controllers

Yohanis Wolde Fikremariam
Advisor: Dr.-Ing Frezewd Lemma

A Thesis Submitted to
The department of Computer Science and Engineering
School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies
Adama Science and Technology University

March, 2023
Adama, Ethiopia

Declaration

I hereby declare that this Master Thesis entitled "**An Efficient Routing Algorithm to Improve the Performance of Floodlight SDN Controllers**" is my original work. That is, it has not been submitted for the award of any academic degree, diploma, or certificate in any other university. All sources of materials that are used for this thesis have been duly acknowledged through citation.

Yohanis Wolde	_____	_____
Name of the student	Signature	Date

Recommendation

I/we, the advisor(s) of this thesis, hereby certify that I/we have read the revised version of the thesis entitled "**An Efficient Routing Algorithm to Improve the Performance of Floodlight SDN Controllers**" prepared under my/our guidance by **Yohanis Wolde** submitted in partial fulfilment of the requirements for the degree of Mater's of Science in Computer Science and Engineering (CSE). Therefore, I/we recommend the submission of revised version of the thesis to the department following the applicable procedures.

Dr.-Ing. Frezewd Lemma	_____	_____
Major Advisor	Signature	Date
_____	_____	_____
Co-advisor	Signature	Date

Approval of Board of Examiners

I/we, the advisors of the thesis entitled "An Efficient Routing Algorithm to Improve the Performance of Floodlight SDN Controllers" and developed by Yohanis Wolde, hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

Dr.-Ing. Frezewd Lemma

Major Advisor	Signature	Date
Co-advisor	Signature	Date

We, the undersigned, members of the Board of Examiners of the thesis by Yohanis Wolde have read and evaluated the thesis entitled "An Efficient Routing Algorithm to Improve the Performance of Floodlight SDN Controllers" and examined the candidate during open defense. This is, therefore, to certify that the thesis is accepted for partial fulfilment of the requirement of the degree of Master of Science in Computer Science and Engineering.

Chairperson	Signature	Date
Internal Examiner	Signature	Date
External Examiner	Signature	Date

Finally, approval and acceptance of the thesis is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

Department Head	Signature	Date
School Dean	Signature	Date
Office of Postgraduate Studies, Dean	Signature	Date

ACKNOWLEDGEMENT

First and foremost, praise and thank God, the Almighty, for his protection throughout this challenging time.

I express genuine gratitude to my passionate advisor, Dr. Eng. Frezewd Lemma, for his constructive comments and advice. He is always kind enough to correct my work consistently. My sincere thanks also go to the Department of Computer Science and Engineering School of Electrical Engineering and Computing for their countless support.

Finally, I thank my family for their appreciation, motivation, and support.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

ACO Ant Colony Optimization

AFR Adaptive Flow Routing

API Application Programming Interface

ARP Address Resolution Protocol

BDDPs Broadcast Packets

BFA Bellman-Ford Algorithm

BP Blocking Probability

CBench Controller Benchmark Tool

CLI Command Line Instruction

DCN Data Center Network

DRAF Distance-Based Adaptive Flow Routing

GUI Graphics User Interface

IDE Integrated Development Environment

ILP Integer Linear Program

IP Internet Protocol

IPv4 Internet Protocol Version 4

IT Information Technology

LAN Local Area Network

LFA Link Flooding Attacks

LLDPs Link Layer Discovery Protocols

L2 Layer 2

MAC Media Access Control

MHSPA Multi-Heuristic Shortest Path Algorithm

MPLS Multiprotocol Label Switching

JVM Java Virtual Machine

NaaS Network as a Service

OF OpenFlow

ORWFA Overlapping Routing with Weighted Flow Aggregation

QoE Quality of Experience

OS Operating System

OVS Open Virtual switch

RAF Reliability Aware Flow

REST Representational State Transfer

SBI South Bound Interface

SBIAPI South Bound Interface APIs

SDN Software Define Network

TCAM Ternary Content Addressable Memory

TCP Transmission Control Protocol

UDP User Datagram Protocol

UI User Interface

VM Virtual Machine

VLAN Virtual Local Area Network

ABSTRACT

With the advancement of high-tech devices and the growing demand for large data processing, networking systems have become unresponsive to user demands. Due to the high traffic on the network system, the capacity of network technologies such as wired, wireless, and cellular networks have been dramatically increased. The traffic nature of today's network system is highly complex, making conventional networks incapable of handling it. These traditional network characteristics could not be adapted to changing demands. As a result, managing the various networking devices is extremely difficult because of the inflexibility of increasing network size and the reliance on the specific vendor's software. Software-defined networking was created by decoupling network control from data-forwarding devices. SDN aimed to create a networking paradigm that responds quickly to changing network requirements. The SDN controller uses the OpenFlow protocol, which handles rules for traffic at the switch. The Floodlight controller uses Dijkstra's as a shortest path finding algorithm. Dijkstra's algorithm searches for the shortest path without prior information. This makes Dijkstra's algorithm waste valuable resources on useless nodes. In this thesis, the multi-heuristic shortest path algorithm, which removes unnecessary node checking, was proposed to optimize the performance of the Floodlight SDN controller. This study was implemented and tested on the network that the Mininet network emulator created. The network consists of virtual switches and hosts that the Floodlight controller manages. After network setup, the performance evaluation was performed using the CBench tool, which tests for throughput and latency metrics. Also, iperf is used to test packet loss. The proposed multi-heuristic shortest path algorithm can increase the throughput by an average of 22.27%, reduce an average latency by 5.63%, reduce response time by an average of 3.34%, and also decrease packet loss of 17.09% when compared with Floodlight controller with Dijkstra's algorithm.

Keywords: Software Defined Networking (SDN), OpenFlow, Floodlight controller, Dijkstra's, Multi-heuristic shortest path algorithm (MHSPA)

CHAPTER 1 INTRODUCTION

1.1 Background of the study

Software-defined networking (SDN) is an evolving networking paradigm that hopes to change the limitation of current network infrastructures. Also, it decouples the vertical combination by separating the network's control logic from the underlying forward devices. Because of the separation of the control planes and data planes, network switches become simple forwarding devices. We can implement the control logic in a logically centralized or distributed controller. SDN makes network reconfiguration simple and introduces a more flexible architecture. It is a method for developing computer networks that distinguish and abstracts the underlying infrastructure of these systems from applications and network services [1].

The SDN controller finds the control plane. Also, it controls the forwarding path through which the packet transfer takes place in the network [2]. Several SDN controllers, such as OpenDaylight, pox, and Floodlight [3], are available. Floodlight is a java based open source OpenFlow controller contributed by Big Switch Networks. Floodlight has a topology manager module to compute the shortest path using Dijkstra's algorithm [4].

Routing is transferring information from source to destination through the optimal path. The controller calculates the optimal path by the shortest path algorithm. So, improving the shortest pathfinding technique will improve network performance. Performance metrics can measure network performance. The parameters employed to measure network performance are throughput, delay, and latency [5].

This study aims to improve optimal pathfinding by implementing multi-heuristic shortest pathfinding algorithms (MHSPA) into the Floodlight SDN controller. Floodlight uses Dijkstra's shortest pathfinding algorithm by default. Dijkstra's algorithm blindly searches a path, which wastes lots of resources. A multi-heuristic shortest pathfinding algorithm is implemented on the controller as an optimal pathfinding algorithm. The multi-heuristic algorithm uses extra information to guide its search by estimating the distance from the next node to be evaluated to the goal position. This value enables the algorithm to guess the next best node. Getting the next best node provides an optimal path with better performance.

1.2 Motivation of the study

With the increasing demand for the fastest connectivity, operators need their networks to be capable of programmability, centralized intelligence, and abstraction. The programmability nature encourages innovation and accelerates service introduction, and their network's centralized intelligence behavior helps simplify control and optimize performance and intelligent management. The abstraction of the network infrastructure decouples the control with the data plane of the traditional network.

The need for high-performance SDN controllers is crucial to meet the demand. The floodlight controller lack performance. Therefore, it is essential to enhance the Floodlight controller's performance, an open-source controller implemented by different enterprises. Unlike closed-source controllers, represented as distributed controller instances, open-source controllers are represented as single controller instances. So, this demand motivates me to enhance the Floodlight controller's performance using the Multi-Heuristic shortest path algorithm, which has a high impact on SDN-based networks.

1.3 Statement of the Problem

Software-Defined Networks offer adaptable and intelligent network processes by splitting a traditional network into a centralized control plane and a data plane. The control plane adds flow paths to switches and improves network performance. The controller is a critical component for all data plane activities in the control plane. Hence, the performance and capabilities of the controller itself are critical [6].

The shortest path problem is the problem that finds the minimum distance or pathway between nodes or vertices in a graph. Many algorithms solve the shortest path problem. Dijkstra's algorithm is one form of the greedy algorithm. Dijkstra uses blind search to calculate the shortest route. Because of this, the algorithm wastes valuable resources on unnecessary nodes to the destination. The Floodlight controller uses Dijkstra's algorithm to search and calculate the optimal path from source to target nodes. The limitation of Dijkstra's algorithm makes the Floodlight Controller waste unnecessary resources to find the optimal path. This study implements multi-heuristic shortest-path algorithms to solve the above problem

.

1.4 Research Question

Based on the problems stated in the statement of the problem section, this research tries to answer the following questions.

1. How to design a multi-heuristic shortest path algorithm for the Floodlight controller?
2. How to implement a multi-heuristic shortest path algorithm to the Floodlight controller to improve the performance of the controller?
3. How can a multi-heuristic shortest path algorithm improve the performance of the Floodlight SDN controller?
4. How to the performance of a Floodlight SDN controller with a multi-heuristic shortest path algorithm be evaluated using throughput, latency, packet loss, and response time?

1.5 Objective of the Study

General Objective

The study's primary objective is to improve the performance of the Floodlight SDN controller using a multi-heuristic shortest path algorithm.

Specific Objective

To reach the general objective, the researcher defines the following specific objectives:

- To review the literature and related work in an improvement of Floodlight controller routing and shortest path finding algorithm,
- To study the Floodlight controller's shortest pathfinding and routing algorithm,
- To propose and design a multi-heuristic shortest path algorithm,
- To Implement a multi-heuristic shortest path algorithm in the Floodlight SDN controller,
- To simulate MHSPA using Mininet simulation tool,
- To analyze throughput, latency, packet loss, and response time of Floodlight SDN controller from simulation result

1.6 Significant of the study

This study's primary significance is allowing OpenFlow-enabled SDN device developers to develop a fast SDN controller. In addition, it is helpful for Network administrators, ICT experts, and SDN controller developers. Furthermore, it contributes to the research community as input for further research.

1.7 Expected outcome of the study

This study's expected outcome is the performance improvement in Floodlight controllers. Also, the modified version of the Floodlight controller routing algorithm will be produced.

1.8 Delimitation scope

Scope

The scope of the research within the given resource and time is:

- There are a lot of SDN controllers in the market. This study assumes the proposed MHSPA works over different SDN controllers. But this study focuses only Floodlight SDN controller,
- To develop a multi-heuristic shortest path algorithm for Floodlight SDN controller,
- To implement a multi-heuristic shortest path algorithm to Floodlight SDN controller,
- To simulate MHSPA and Dijkstra's algorithm using linear topology and Mininet tool,
- To analyze the Floodlight controller's throughput, latency, packet loss, and response time from the simulation result.

Limitations

Due to some constraints, this study has the following limitations:

- Because of the unavailability of real OpenFlow devices, this study uses virtual devices, i.e., switches and controllers.
- This work is only implemented and analyzes the performance of a Floodlight controller.
- This work evaluates throughput, latency, packet loss, and response time performance matrices.

CHAPTER 2 LITERATURE REVIEW

2.1 Overview of Traditional Network

The computer network is the basis for today's telecommunications technologies, as networks become part of the critical infrastructure of our businesses, homes, and schools. Network technology reaches today's high technologies, and network computing technology has become a part of our day-to-day life due to the massive amount of information and data stored in the cloud.

2.1.1 Traditional Network Architecture

It is essential to examine the conventional network architecture to understand the new SDN approach better. It has transmission equipment, software, wireless or wired connectivity between components, and communication protocols. Different mediums are used to connect hardware and build a network. These connections lead to different transmission modes. Ethernet, wireless, or optical are these modes of transmission. Hardware like servers, switches, and routers make up the network's core. They are connected via one or more transmission methods. When we think of end-to-end communication, the data moves through the network hardware one hop at a time until it reaches its destination. Sometimes, the data can be broken up into multiple pieces and sent on different routes to reach its destination.

2.1.2 Limitations of Traditional Networking

Traditional network designs need to be more capable of meeting today's market demands. The limitations of traditional networks, which include the following, prevent existing network architectures from meeting the needs of today's customers, organizations, and providers.[7]

- **Inability to scale.** The network must expand at the same rate as the demand for the data centers' rapid expansion. However, the addition of thousands of networking devices like routers, firewalls, and switches required for configuration and management made networks extremely complex. Network administrators have always relied heavily on bandwidth oversubscription to scale enterprise networks. However, with the virtualization of data centers, traffic patterns have become highly

dynamic, making it difficult to predict them. Scalability issues are even more challenging for large companies like Google, Amazon, eBay, and Facebook. These operators work with massive data processing systems and complex datasets that cannot be manually configured.

- **Vendor dependence.** To meet changing industry standards or customer needs, internet service providers and data centers are always looking ahead to implementing new features and services. However, the equipment vendor's life cycles for the generated service and equipment, which might be roughly three years or even more in certain situations, limit the capacity to respond. Additionally, network operators' ability to adapt the system to their contexts has been restricted by the need for more open, standardized interfaces. This mismatch between network capabilities and client demand has severely impacted the industry. Therefore, the industry created an SDN paradigm to address these difficulties.

- **Inconsistent policies**. It may need to configure thousands of devices and procedures to apply a network-wide policy. For instance, IT may need hours or even days to update ACLs throughout the whole network each time a new virtual machine is installed. IT personnel finds it very challenging to apply a consistent set of access, security, quality of service, and other policies to users becoming more mobile due to the complexity of today's networks. Because of this, the organization is prone to security breaches, non-compliance with laws, and other adverse effects because

- **Management Complexity.** Most computer network technologies have always been built on routing protocols designed to connect hosts reliably over long distances at high speeds and in various network topologies. Over the past few decades, protocols have been designed in various ways that lead to separation. Each protocol solves a particular problem without implementing concept abstractions to meet current industry requirements like high availability, security, and extended connectivity. Network management complexity is the main issue that network administrators face today because of this approach. One illustration of this is that network administrators now find it difficult to add or remove devices from a network, such as switches, routers, firewalls, and Web authentication portals. For example, access lists, VLAN quality of service policies, routing protocols, and network topologies need to be changed.

Until the introduction of the virtualization service, only a single server connected a few clients. Thanks to virtualization services, applications can now be distributed across multiple virtual machines. In addition, VMs frequently need to migrate to achieve workload balance; Traditional networking, which was not intended for such dynamic flow changes, was confronted with several difficulties due to this functionality of virtualized platforms.

In addition to the above, equipment vendor dependence and software version compatibility must be considered before making any changes to the network. As a result, network administrators maintain a relatively static network to avoid or minimize service disruptions brought by any change. The dynamic landscape of server virtualization is limited by the static nature of static network design, increasing the number of hosts that require connectivity.

A networking paradigm called Software-Defined Networking (SDN) is envisioned as the next generation of computer networking to address the above problems in traditional networking. SDN has two characteristics that distinguish it from traditional networks. These characteristics are the separation of the management plane and the data plane and the logically centralized control plane to control multiple data plane elements under a single software control Program.[8]

2.2 Software-Defined Network

The Open Networking Foundation defines SDN as “an emerging network architecture in which network control is decoupled from forwarding and is directly programmable.” The control plane, formerly firmly tied to specific network devices, is now being migrated into available computing devices, allowing the fundamental infrastructure to be hidden from applications and network services, which can treat the network as a logical or virtual entity.[9]

Technological advancement has increased the number of network applications, users, and devices that existed before. The networking system could not handle the new technologies and innovations in a way that would maximize its performance. Applying newly developed technologies can enhance the quality of experience (QoE) and service (quality of service) to address networking challenges. Software Defined Networking (SDN) is one of these

technologies. The network bottleneck, primarily data centers and cloud computing services, could be resolved by selecting high-speed network parameters.[10]

Nowadays, the networking system comprises sophisticated equipment and new technologies that are added regularly. The demand for Big Data systems that process substantial data volumes, such as cloud computing and mobile traffic, and traffic from mobile devices is also overgrowing. Therefore, the capacity of wired and wireless network technologies and cellular networks (i.e., 4G and 5G) grows. Because of the user demand and network capacity, the networking system requires a high-performance network management system. Furthermore, the present network traffic pattern is highly complicated, and traditional network design needs to meet the demands of today's consumers.[11]

Software Defined network is a new and emerging technology that breaks the vertical integration of traditional networks. The data and control planes integrate into a similar network unit in a traditional network. SDN is a new approach to designing, implementing, and managing networks that decouple the network control (control plane) and the forwarding

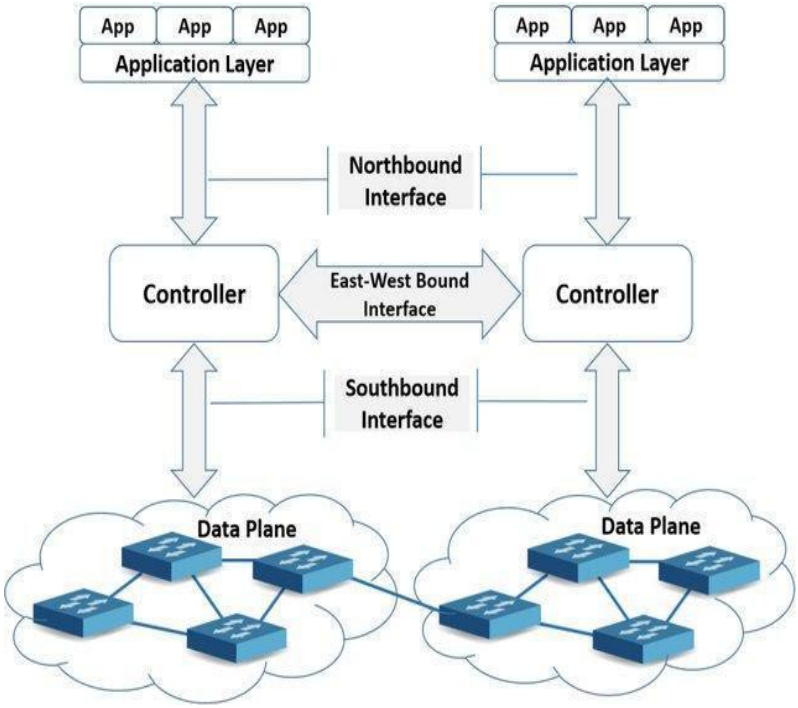

Figure 1: SDN architecture [14]

process (data plane). The fundamental distinction between conventional networks and SDN is that SDN removes the decision-making part from the routers and offers a centralized

Control-Plane that creates a network view for the control and management applications. Through the launch of SDN, several new network capabilities and services have been enabled, such as Traffic Engineering, Software Engineering, Network Virtualization, and Automation. [12]. The application, control, and data layers are the three layers of the SDN network architecture, as depicted in figure 1 above.

- **Application layer/plane.** This layer, often known as the management layer, comprises network programs that look after an SDN's control logic. Routing, load balancing, policy enforcement, or a custom application from a service provider are just a few examples of the new applications and services that may be easily and quickly implemented in networks with SDN capabilities. Additionally, it enables network orchestration and automation through already-existing APIs.[9]
- **Northern Bound Interface.** Northbound APIs are used by applications or higher-layer control programs running on top of the controller. The application layer is an integral part of the SDN technology since the value of SDN is pinned to the innovative applications. Northbound APIs are also used to connect the controller to cloud management, such as OpenStack and CloudStack, which are used to manage clouds. Nowadays, REST API has been the utmost used API, and most controllers (including RYU, Floodlight, ONOS, and OpenDaylight) support it [21]
- **Control layer/plane. -** This layer is often known as the brain or the intelligent part of the network. Because it controls and makes decisions for the entire network and allows the SDN to abstract its physical network state as a comprehensive network view to a controller instance running in the application layer. Once individual network devices are configured individually, an overall network can be configured or organized as one single entity. In other words, the SDN controller (control plane) acts as a logically centralized brainpower with an overall view of the network and the capability to reconfigure it dynamically. Network management has everything to do with the control plane. Northbound application programming interfaces (APIs) is the API that facilitates the communication between the application plane and control planes (Controller). Furthermore, in the control plane, several controllers can connect with each other using an East-West bound interface. Southbound API of the OpenFlow (OF) protocol is used by the control plane to communicate with the data plane. [14]

- **Southbound interface.** Southbound APIs create a connection between the SDN controller and infrastructure layer (data plane devices). This interface creates infrastructure layer abstraction in the network. OpenFlow is the most popular southbound protocol. Nevertheless, the marker has several non-OpenFlow South Bound Interface APIs (SBIAPI).
- **Data layer/plane.** The infrastructure layer consists of the forwarding equipment on the network, such as load balancers, switches, and routers. It uses the southbound APIs to connect with the control plane by receiving the forwarding rules and policies to apply to the corresponding devices.[22]

2.2.1 Benefits of SDN

Software-defined networking will change how engineers and designers build and run their networks to meet business needs. With the advent of SDN, networks have become a non-proprietary open standard and are easy to program and manage. SDN will give businesses and operators more control over their networks, allowing them to adapt and optimize their networks to reduce overall network maintenance costs. Some of the main benefits of SDN can be summarized below [15].

- **The programmability of the network.** By implementing a new layer of orchestration, SDN can address the inflexibility and complexity of traditional networks. SDN allows companies to programmatically control and scale their networks without sacrificing performance, reliability, or user experience. The data and control plane abstraction are what make SDN so valuable. By removing complexity from the infrastructure layer and adding visibility to applications and services, SDN simplifies network management and brings virtualization to the network. It abstracts flow control from individual devices to the network layer. Network-wide flow control allows administrators to define network flows that meet connectivity needs and the specific needs of individual user communities. The SDN approach relieves network administrators of the need to implement personalized procedures and policies on each network device individually. In SDN architecture, control plane functions are detached from the physical device and performed by an external controller. SDN offers programmability in the control plane itself. Programmability allows changes to be implemented and propagated through secure channels across specific devices or network hardware. This approach makes it easier

to integrate new devices into existing architectures. SDN controllers enhance the traffic engineering capabilities of network operators working with video traffic. It allows network operators to control congestion hotspots and reduces the complexity of traffic engineering.

- **The Rise of Virtualization.** SDN holds great promise for large-scale data center (DC) management. Data centers have significant scalability issues, particularly with the growth and migration of virtual machines (VMs). Migrating the virtual machine and updating the Media Access Control (MAC) address table using a traditional network architecture can disrupt the user experience and applications. Thus, network virtualization, which can be thought of as an SDN application, presents a significant opportunity for large-scale data centers. It provides tunnels that can extract MAC addresses from the infrastructure layer, allows Layer 2 traffic to run on Layer 3 overlays, and simplifies the deployment and migration of virtual machines in the network.
- Additionally, SDN enables multi-tenant hosting providers to link physical and virtual servers, local and remote facilities, and public and private clouds into a logical network. Therefore, each customer will have a separate opinion of the network provider. SDN adds a layer of virtualization to cloud providers' architecture and allows their tenants to get different views of the Data Center Network (DCN) based on their needs. SDN is a promising approach to delivering network as a Service (NaaS).
- **Device Configuration and Troubleshooting.** With SDN, device configuration and troubleshooting can be done from a single point on the network, bringing networking closer to achieving the goal of a configurable dynamic network. Shape and make adaptations according to needs. SDN also offers an opportunity to drive network innovation by providing a programmable platform for experimentation with new protocols and policies using production traffic.
- **Cost reduction.** Installing SDN does not need a large budget and labor. Some SDN products are even free and open source for everyone. However, there are licensing fees for some SDN solutions. Overall, this reduces costs from a deployment, operations, and management standpoint.

To achieve the above benefits, the SDN must make the desired changes in the network by implementing the appropriate forwarding rules through the south interface connecting the

control layer and the infrastructure layer, as shown in Figure 1. There are different southbound interface protocols, such as OpenFlow [16], Opflex [17], NETCONF [18], ForCES[19], and POF [20]. There are other options for southbound interface implementation, but OpenFlow is the first choice of researchers and enterprises [16]. The north interface is the one that responds to business policy implementation requests on the controller's application layer. The widely used northern interfaces are to implement service policy to define traffic behavior.

2.3 OpenFlow(OF) Protocol

The OF protocol is the most widely used and applied southbound API for SDN-enabled networks. OpenFlow was introduced and maintained by the Open Networking Foundation (ONF) [23], contributed by the leaders of IT industries. Flow can be represented in an OpenFlow-enabled network as a transmission control protocol (TCP) connection, packets with a matching MAC address, an IP address, a VLAN tag, or a switch port. There are one or more flow tables on the OpenFlow switch. A flow table consists of a collection of flow entries to match, and process packets a flow entry is used. It consists of many matching fields for matching packets, encounters for tracking packets, and application instructions [9]. The OpenFlow switch employs an OpenFlow channel to interact with the OpenFlow controller. The Open Networking Foundation (ONF) has released several versions of OpenFlow. Different functionality and features are added in each release to comply with

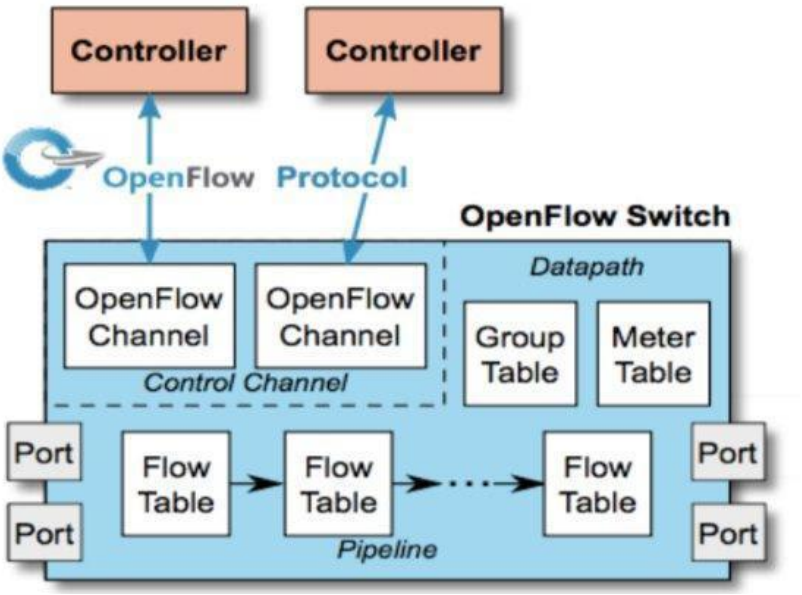

Figure 2: OpenFlow architecture [24]

commercial deployment. The OpenFlow protocol's overall design, feature, and summarization of the features of versions 1.0–1.5 specifications are listed in table 1. In detail, it gives an overview of current OpenFlow protocols organized by related fields.

Table 1: OpenFlow feature comparison

Version	Major Feature	Reason	Use Cases
1.0-1.1	Multiple tables	Avoid flow entry explosion	
	Group table	Enable applying action sets to a group of flows	Load balancing, failover, link aggregation
	Full VLAN and MPLS support		
1.1 - 1.2	OXM Match	Extending matching flexibility	
	Multiple controllers	HA/load balancing/scalability	Controller failover, controller load balancing
1.2-1.3	Meter table	Add the quality of service and DiffServ capability.	
	Table miss entry	offer flexibility	
1.3-1.4	Synchronized tables	improve table scalability	MAC learning
1.4-1.5	Egress table	Enable processing to be done in the output port	
	Scheduled bundled	Further, improve switch synchronization.	

2.4 Routing

Routing is how the networking system decides where to send a packet. It also discovers the path using a combination of Link discovery and management, topology management, and a routing engine. Instead of using destination-based routing, which is popular in traditional networks, SDN uses flow-based routing. For each incoming traffic flow, the relevant flow rule is set up in the switch's Flowable Memory. It is a logical data structure that governs data flow in OpenFlow switches. This data structure utilized Ternary Content Memory Addressable (TCAM) on switches, a particular high-speed memory, to guarantee matching flexibility and good lookup performance in constant time[51].

2.5 Comparison of SDN controller

There are a lot of open sources and proprietary SDN controllers available. These controllers have different features and functionalities. For comparison reasons most famous of these controller features and functionalities are summarized in table 2.

Table 2: OpenFlow controllers' comparison

Name	Lang.	SBI	Platform	UI	License	Modularity	Documentation
Floodlight [25]	Java	OF 1.0-1.5	Linux, MacOS, Windows	CLI, WUI	Apache 2.0	Fair	Good
ODL [27]	Java	OF 1.0,1.3	Linux, MacOS, Windows	CLI, WUI	EPL 1.0	High	Good
POX [28]	Python	OF 1.0	Linux, MacOS, Windows	CLI, GUI	Apache 2.0	Low	Limited
Ryu [29]	Python	OF 1.0-1.5	Linux, MacOS	CLI	Apache 2.0	Fair	Good

The Floodlight controller is achieving this approach in practical delivering open SDN features than available alternatives listed in Table 2. Furthermore, the Floodlight controller has the capability of combining both SDN and Traditional networks by using islands. This feature is significant as it allows companies to transition smoothly from traditional networks to SDN.

2.5.1 Floodlight controller and Its Architecture

The Floodlight [25] controller is an open-source SDN/OpenFlow controller written in Java. It is Apache-licensed and is supported by a large community of SDN controller developers, including several network researchers and engineers from BigSwitch Networks [31]. It also supports OpenFlow protocol from version 1.0 through 1.5, the latest version for communication between controller and forwarding devices.

The Floodlight controller design includes features useful for data centers and enterprise-class networks. Because OpenFlow defines protocols via an OpenFlow-enabled switch, a controller activated remotely can change the behavior of core networking devices via a well-defined data forwarding instruction set. The Floodlight controller is intended for use with various switches and routers that support OpenFlow standards. The Floodlight controller could also support hybrid networks where OpenFlow-enabled switches are connected through non-OpenFlow switches. Some of the Floodlight controller's essential features are listed below [32]:

- It is compatible with many real and virtual OpenFlow-enabled devices.
- Its modularity makes it simple to improve and extend.
- It is straightforward to set up and configure with minimal dependencies.
- It can also manage a network of hybrid environments, managing multiple sections of OpenFlow and non-OpenFlow clusters with minimum restrictions.
- It provides a high-performance network, a commercial product from BigSwitch Networks.

Furthermore, the Floodlight controller has a built-in collection of modular applications. These modular applications are used for different solutions. The internal architecture, the available Java module applications, and some of the Floodlight controller's core features are described below:

A) Device Manager. This core module tracks devices moving around a network and defines the destination device for a new flow. The device manager knows and learns about devices through PacketIn requests. It uses data from the PacketIn to classify the device based on how the entity classifier is configured. The entity classifier recognizes a device by default using its VLAN and MAC address. These two characteristics will define the device's uniqueness. Other characteristics, such as IP addresses, are also covered in this module. The device connection points are an essential piece of information. An attachment point for that device is created When a packet arrives on a switch.

B) Topology management. Oversees maintaining the controller's topology information and finding routing in the network. Also, in the computation of the shortest path by using Dijsktra's Algorithm,

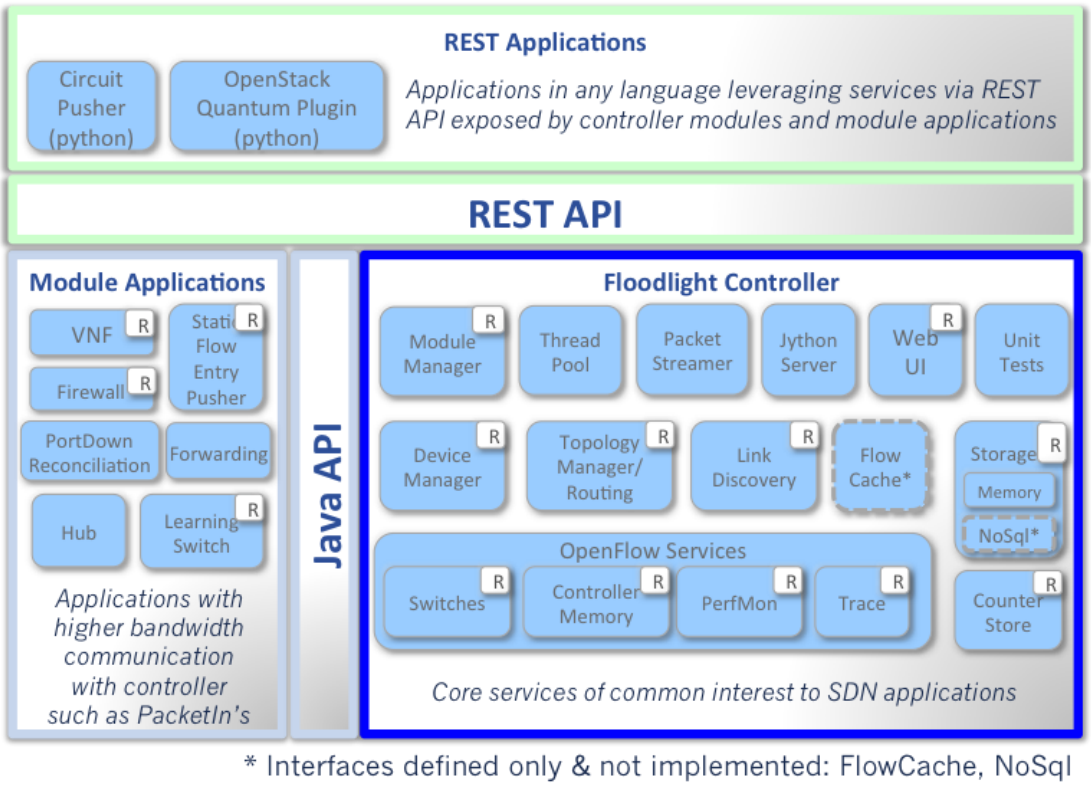

Figure 3: Floodlight SDN controller architecture[25]

C) Link Discovery. It is responsible for maintaining and discovering the status of connection in the OpenFlow network. The link discovery services use LLDPs and broadcast packets (BDDPs) to detect links.

D) **Forwarding module.** The Floodlight controller is designed to operate in networks that contain both OpenFlow and non-OpenFlow switches. The forwarding mechanism must take this situation into account. The algorithm will find all OpenFlow clusters with device attachment points for both the source and destination devices. Following that FlowMods will be stored along the flow's shortest path. If the cluster receives a PacketIn and there is no connection point on that island for the device, the packet will be flooded through the network.

E) Packet Streamer. A packet streamer or filter service can use the static flow principle to forward the OpenFlow packet switch and the controller to any other connected monitoring device. This service has two functionalities; filtering characteristics of OpenFlow messages based on a REST interface and streaming the filtered packets using a thrift-based interface.

F) Static Flow Pusher. This Floodlight module allows the insertion of flows and groups that include action and match columns into the OpenFlow network via REST API.

2.5.2 Floodlight Design Features

Floodlight SDN controllers have different features that are preferable by the scientific community. Some of these features are listed and described as follows.

- **OpenFlow and non-OpenFlow support:** The Controller supports both OpenFlow and OpenFlow networks. Supporting OpenFlow and non-OpenFlow devices enables the controller to manage the combination of devices through OpenFlow and other protocols using clustering. From the OpenFlow cluster to the none OpenFlow cluster, there should be only one link between, and not loop back and assume that the non-OpenFlow cluster is a single L2 broadcast domain. There are six versions of OpenFlow protocols starting from V1.0 to V1.6. This version has different features and improvements, as described in table 1. The latest Floodlight controller supports V1.5, which has an egress port table; it helps the process can be done in the output port and enhanced switch synchronizations and multiple switch configuration.

- **Network Virtualization:** Network virtualization is one of the essential benefits of SDN. The network resource can be allocated to different flows of traffic, and it has been a means of isolating different types of traffic that share the same switched LAN infrastructure Using this technology. The advantage of virtualization is enhanced to

enhance network reliability, speed, flexibility, scalability, and security. It is said to be especially useful in networks that experience sudden, large, and unforeseen surges in usage.

- **OpenStack support:** OpenStack is an open-source cloud-based framework service that can provide virtual servers and services for customers.

2.6 Related Work

Researchers did several works to improve SDN performance and best pathfinding. The SDN assigns the user to the nodes with higher latency, affecting the system performance. In [33], the researchers propose a Multicast routing model to minimize the number of flow entries in SND. Because multicast trees are determined individually, further reductions in the size of flow entries are possible when multicast trees of multiple multicast requests are determined using the conventional scheme. The proposed model determines all multicast trees at the same time and encourages the sharing of unicast entries among multiple multicast trees. They expressed the model as an ILP problem and created a heuristic algorithm that can be used if the ILP problem is intractable. They assessed the proposed model by solving the ILP problem with the heuristic algorithm. The proposed model is designed for the static scenario, in which the sender and set of receivers for each multicast request are known ahead of time. The proposed model can be applied when multiple multicast request routes must be optimized, such as at the start of network operation or during scheduled network maintenance.

In [34], the researchers propose energy-aware routing considering load balancing for SDN. They proposed a two-phase SDN-based routing mechanism that minimizes energy consumption while maintaining a certain quality of service for user flows and considers link load balancing. The first stage is a scheduled offline method using the ACO algorithm to find a graph with the fewest switches accountable for routing existing flows. This graph is modified and serves as the input for the following phase. The second phase is an online event-driven algorithm that routes incoming flows to meet the quality-of-service requirements while also balancing link loads.

Another study [35] proposes Reliability Aware Multiple Path Installations in Software-Defined Networking. This method calculates the primary path based on higher reliability.

Then, they propose an RAF variant, Distance-based RAF (DRAF), that calculates the optimal path based on the combined value of shorter distance and higher reliability. The RAF process is divided into the Bootstrapping Process, the Graph Composition Process, and the Path Computation/Installation Phase. During the bootstrap phase, the controller starts feature requests and several other asynchronous control requests for global awareness of the infrastructure layer. Their mechanism employs switch state inspection, modification, flow rule and interface statistics, aliveness monitoring, and capabilities. Applications that are publisher-subscribers and act as event-driven sources for network control are how the process is typically presented. In the second stage, the controller regularly informs graph connectivity in response to end-users and device detection events. The controller's reliability inquiry represents the link's stability between forwarding devices and end-user connectivity with devices. In the final phase, the controller installs rules reactively. After the accurate calculation method is completed, the controller will determine the most reliable paths and backup to install.

Flow aggregation and routing are jointly designed in this study [36]. Their study goal is to minimize the total cost by trading off those two factors by properly defining costs for each. An Integer Linear Program (ILP) is created to find the optimal solution by leveraging flow table costs and average transmission delay. The study also proposes a heuristic algorithm ORWFA (Overlapping Routing with Weighted Flow Aggregation) for fast design to make their solution more scalable for large network scenarios. First, ORWFA constructs multiple spanning trees with different flow targets as the roots. Then it aggregates based on the corresponding spanning trees to share overlapping routes. The routing tree is built using the spanning tree's non-closed circle characteristics, which allows for more overlapping routes and better flow aggregation conditions. It considers the trade-off relationship between the transmission delay cost and the hardware cost of the storage chip, ensuring that the flow table is aggregated while selecting the shortest route to achieve a low transmission delay. It also considers the flow's priority so that the flow with the higher priority takes the shortest route possible. In comparison, the flow with the lower priority takes the longest route possible. The aggregation can save flow table space while decreasing average transmission delay.

An adaptive greedy flow routing algorithm was proposed by Shirmarz et al. [38]. The greedy algorithm for flow routing to meet performance requirements. In addition to the greedy

algorithm, they implement three other routing algorithms in Java to be embedded in the Floodlight controller to evaluate the adaptive flow routing (AFR). In the proposed greedy scheme, the controller selects the optimal path every second for each flow based on the flow's performance requests. Each flow has a source, destination, and four performance metrics limitations to guarantee flow performance. The proposed approach tries to order flows and routes and selects the optimized paths. Additionally, the proposed scheme improves average delay, blocking probability (BP), utilization, and running time.

In this research [39], the routing method based on BFA is applied to the SDN to solve the scalability problem. The SDN is represented in the graph with its nodes and edges. With less computation time, the bellman ford finds the best path for the user based on the graph nodes. A fast and efficient routing protocol is required to optimize the extensive SDN network for traffic demands. The proposed method analyzes network routing in SDNs, uses BFA to divide the network into many subnets, and finds the shortest path in each subnet and between subnets. The BFA for implementing the proposed algorithm in the SDN controller. The test environment design for the proposed routing algorithm underpins the proposal of a routing algorithm for large-scale SDNs.

Another study proposes self-Routing Techniques in Software-defined Networking [40]. In the proposed self-routing technique, a stack of labels is allocated to the first packet of each traffic flow. Forwarding instructions are produced simply through the content of these labels. The main advantage is to reduce the number of SDN entries distributed by the controller. Therefore, this helps minimize the load on the controller and increases its scalability. Instead of a controller distributing entries to each switch involved in a communication path, routing instructions will be added as extension labels to the arriving traffic flows. For each new traffic flow, an OpenFlow switch receives a stack of routing labels from the SDN controller. Also, it affixes the routing labels to the first packet of the arriving new traffic flow and forwards it to the next-hop switch involved in this route. Other switches on a similar routing path will use these routing labels to store the route instruction in their flow tables; while processing is always done on the first label, the remaining stack is passed to the next involved switch.

Table 3: Summary of related works

Ref. No	Method used	Achievements	Limitation
[33]	Routing model for determining multicast trees to minimize the total amount of flow entries for several multicast requests. Determines all multicast trees simultaneously and promotes the sharing of unicast entries among different multicast trees.	Reduces the required number of flow entries	Sharing a single entry for many flows may induce congestion
[34]	Two-phase SDN-based routing method to minimize energy consumption and improves the quality of service. Uses a minimum graph-based Ant Colony Optimization (ACO) approach and an innovative weighted routing approach.	Improve the performance of a network in small scale network	It may require a longer time to converge in a dynamic large-scale network
[35]	Use the Reliability Aware Multiple Path Flow Rule (RAF) that computes links consistency and installs the smallest flow rules for multiple paths based on the accurate value of the optimal path.	Improves flow installation traffic Overhead and end-to-end delay	Reliability competition value imposes extra processing load on the controller
[36]	Rule-merging-based and Dijkstra-based solutions to Reduce the number of rule changes required to reroute some traffic from a congested link.	Save flow table space and reduce transmission delay.	The controller may not regularly have flow visibility

[37]	Flow aggregation and routing are jointly designed. An Integer Linear Program (ILP) is created to find the ideal solution by leveraging flow table costs and average transmission delay. They also propose a heuristic algorithm ORWFA (Overlapping Routing with Weighted Flow Aggregation) for fast design.	Improve average transmission delay and minimize flow table size	link congestion may occur when the flow table compression is large
[38]	In the proposed greedy scheme, the controller selects the optimal path every second for each flow based on the flow's performance requests to meet performance requirements.	Improves average delay, utilization, and running time	effect of scalability on the performance of the routing algorithm not considered
[39]	Routing technique based on the Bellman-Ford Algorithm (BFA). The BFA determines the optimal path for the nodes to reach the user with low latency.	Improves latency and computation time	This study did not consider other performance parameters.
[40]	Self-routing technique to reduce controller interaction by reducing the number of entries distributed by the controller. The primary method is applying a stack of routing labels to the first packet of each new traffic flow. The routing labels serve as a forwarding index, indicating the next hop.	reduced the number of interactions between the controller and the network devices,	adding stack labelling increased the controller load

CHAPTER 3 MATERIAL AND METHOD

3.1 Introduction

This section will discuss the method for organization, simulation, and analyzing and comparing the available tools I reviewed. I analyzed the past overview and searched for the best-fit technology to distinguish which method to choose for my preferred methodology. This gives me the best option and flexibility for development and design strategy. I used the scientific research review technique during the initial research phase to obtain relevant information by reviewing the paper and defining the gap. I collect resources from various sources to gain in-depth knowledge. In this research, I followed the inductive approach and quantitative research mechanism. To work with this previously done research, I typically emphasize the existing use case thoroughly. There are several use cases in our research too. In this step, a critical scientific paper review is the primary source of our use cases. Additionally, on top of the paper review, a different data source is used.

3.2 Research Method

This section shows how advanced system solutions are designed and implemented and how to analyze the research based on previously conducted research and currently available technology. At this stage of the study, all mechanics of every step and procedure will clearly be defined based on the identified problem domain and characteristics. So, in this step, I have demonstrated a method of approach, a flowchart of the design system, the proposed system test and evaluation mechanism, comparison default and the proposed algorithm in action, and defined the tools we used in every step. The tasks mentioned above in the research show some quick steps that I get around to the strategy of this research.

- I collected the relevant resource accurately by analyzing published articles, research papers, a dedicated forum about the problem I identified, and visiting official sites of the technology to look around specifications, methodology, technology stack, conventions, and archetype. After all of these, some available experts and community members the community discussion to discuss the best possible ideas around.
- The crucial primary step was to define the identified problem to be solved clearly.
- After that, Then I identified the parameters for designing the proposed solution.

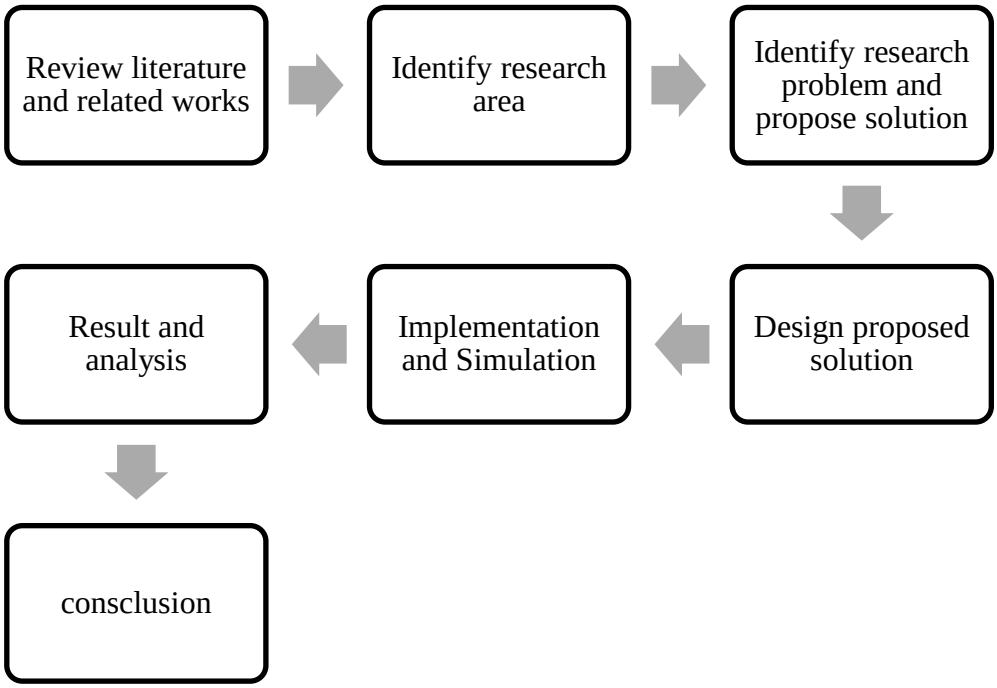

Figure 4: Research methodology

- Then I Implement the proposed solution.
- Finally, I prepared for the conduct of the experiment using different mechanisms that suited the problem I identified and documented the result I obtained from the analysis. Analyze the result to check what is wrong or whether the intended task works or not and look in detail at my conclusion for my algorithm. My algorithm may trigger another issue that needs a solution for the future. We clearly show this without any hesitation to fellow researchers and reviewers in an ethical manner.

3.3 Proposed solution

The suitable solution for the identified problem is proposed and discussed in detail in this section. The solution for the defined problem is to implement a multi-heuristic shortest path finding algorithm to identify the best route from the start node to the goal. Dijkstra's algorithm is one of the many search algorithms that take input, evaluate some paths, and return the solution. Dijkstra's is widely used in pathfinding and graph transversal, plotting an efficiently traversable path between nodes. Dijkstra's is a greedy search algorithm.[41]

The proposed system works by modifying the shortest path algorithm of the controller. For my implementation, I have modified one file in Floodlight Controller,

TopologyInstance.java, which is found under the Topology and path management services section, as depicted in figure 3. This module calculates the path from start to destination and routing selection.

My main modification to this module is adding a multi-heuristic algorithm to calculate the remaining cost to the destination. Based on this cost result, the algorithm selects the next node to the destination. The proposed multi-heuristic routing algorithm solves Dijkstra's limitation by removing unnecessary node checking.

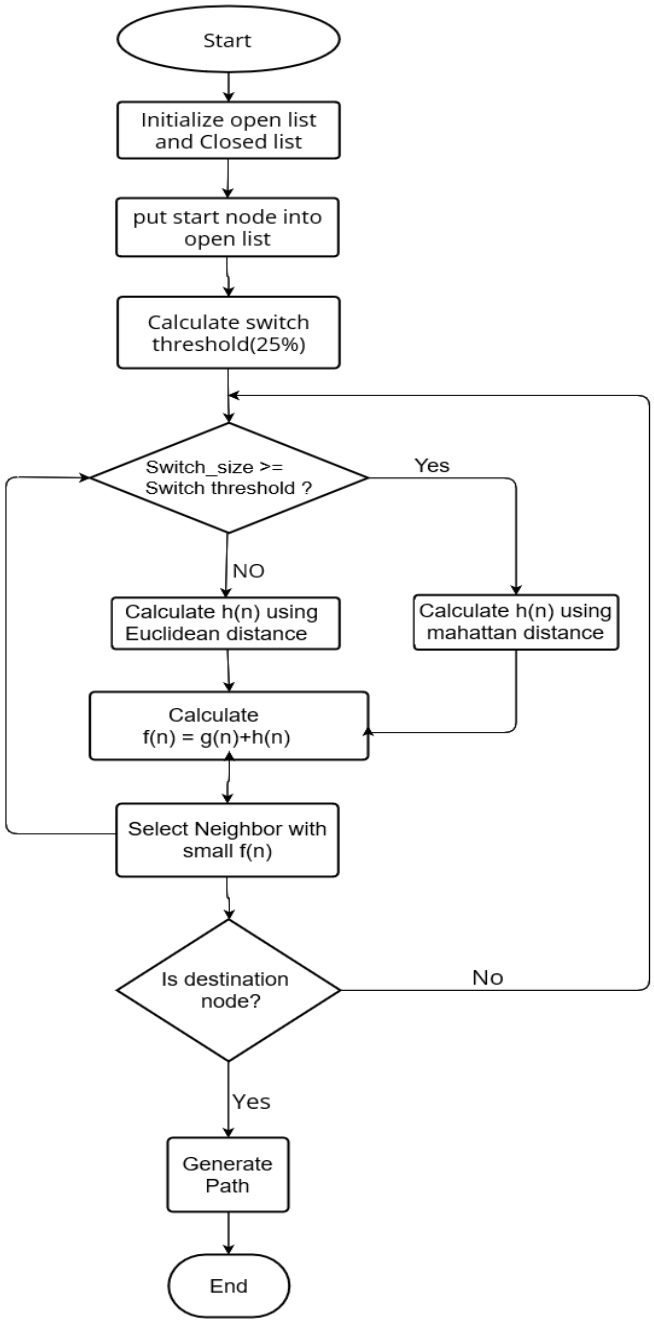

Figure 5: MHSPA flow chart

3.3.1 Multi-Heuristic Shortest Path Algorithm

The Multiple-heuristic shortest path algorithm (MHSPA) is an improved version of Dijkstra's algorithm. This algorithm adds a multiple-heuristic method to optimize the pathfinding mechanism. Multiple-heuristic methods give information to the algorithm about where to go next. Hence, the algorithm only checks some paths. MHSPA uses the f(n)=g(n)+h(n) function to calculate the total cost of the path. Where:

f(n)= total cost from source to destination

g(n) = summation of costs from the start node to the current node

h(n) = estimation of cost from the current node to the destination node

Additionally, the h(n) value guides the algorithm on which node to check next. a switch size threshold to change between heuristic calculation methods. When the first packet arrives on the OpenFlow switch, the switch requests the controller which path to send the packet. First, MHSPA receives topology information from the topology manager module; it calculates network size using the total number of connected switches. After getting the network size, it calculates 25% of them as the threshold limit and stores them to size_threshold.

size_threshold = number of connected switches * 0.25

If the network size in the path exceeds the threshold level, the MHSPA changes the heuristic method to calculate the estimation cost to the destination.

If switch_size > size_threshold

h(n) = Use the Manhattan distance

Else

h(n) = Use Euclidean distance

After calculating the h(n) value, add it to the g(n). finally, based on the total value, the algorithm selects the next node. The algorithm automatically adjusts the estimated cost function, as depicted in figure 6. This makes the algorithm run efficiently.

Algorithm 1: MHSP Algorithm

Input: graph (Node, Start, Target)

Output: path

```
1:  algorithm MHSPA (Graph, source):
2:      for each vertex v in Graph:
3:          cost[v] ← infinity
4:          previous[v] ← undefined
5:      dist[source] ← 0
6:      size_threshold ←Total_num_switch*0.25
7:      Q ← the set of all nodes in Graph
8:      while Q is not empty:
9:          u ← node in Q with smallest cost []
10:         remove u from Q
11:         for each neighbor v of u:
12:             if (link_size > size _threshold)
13:                 remain_cost ← manhattan(u,v)
14:             else
15:                 remain_cost ← euclidean(u,v)
16:             alt ← cost [u] + remain_cost
17:             if alt < cost[v]
18:                 cost[v] ← alt
19:                 previous[v] ← u
20:     return path
```

Figure 6: MHSPA algorithm

3.4 Development Tool and Technique

This section is focused on tools and techniques for the simulation and development of the proposed solution. These tools and techniques are listed below. In this thesis, we will use different software tools; some of them are Floodlight controllers from Big Switch, Mininet, and CBench with these, we use Eclipse ant for developing and building the Floodlight.

3.4.1 Simulation and Designing Tools

The software used to simulate the proposed system is listed and summarized below.

Table 4: Simulation and Designing tools.

Ref. No.	Tools and Technologies	Name	Description
[25]	SDN Controller	Floodlight	Java-based SDN controller
[42]	OpenFlow switch	OVS	A multi-layer virtual switch
[43]	OS	Ubuntu	Open-source Linux-based operating system
[44]	IDE	IntelliJ	For building Floodlight
[45]	Simulator	Mininet	An emulator for SDN and OpenFlow
[46][50][53]	Traffic Generator	CBench, iperf, SIEGE	Open-source. Provides multiple parameters for achieving throughput, latency, and packet loss tests.
[47]	Development Tool	Java	Java is a class-based, high-level, object-oriented programming language

3.4.1.1 Installation and running of Floodlight Controller

As explained in section **2.2.1**, the Floodlight controller is an open-source and enterprise OpenFlow controller. Some of the main features it discussed below [25]:

- Apache-licensed, multi-purpose, Open-source software,
- The community developed, including BigSwitch,
- Because of its ease to use, Floodlight is broadly used in the network community,
- It is designed in a way to scale to the widely increasing number of OpenFlow and non-OpenFlow-enabled devices,
- It supports OpenFlow protocols 1.0 through 1.5

To install and run the Floodlight controller, Linux Ubuntu 16.04 LTS, python, and JVM must be installed. The following steps and prerequisites must be fulfilled. Since Floodlight is Java-based, it needs a JVM which can be installed using the following command.

sudo apt-get install oracle-java8-set-default

To build and run the Floodlight controller, ant, maven, and python dependencies must be installed first. The following code must execute to install these dependencies.

sudo apt-get install build-essential ant maven python-dev

After installing all the required dependencies, it is time to download and run the Floodlight controller using the following command

git clone https://github.com/floodlight/floodlight.git; cd floodlight

git submodule init

git submodule update

mvn install -DskipTests.

sudo mkdir /var/lib/Floodlight

sudo chmod 777 /var/lib/floodlight

3.4.1.2 Installation and running of Mininet.

Mininet is an open-source network emulation tool that creates different types of networks. It creates a realistic virtual network that can be run on an actual kernel, switch, and application code, on a single VM, cloud, or native machine, with a single command in seconds. Using Mininet, anyone can develop, share, and research with software-defined networks using OpenFlow support and Mininet CLI and python API, and we can easily connect with the network. Moreover, it can be customized, shared, and deployed on real hardware.

Mininet supports OpenFlow protocol used in software-defined networks. This feature provides the interface between the control plane and the data plane. Furthermore, it can support a diversity of topologies and have the capability of creating custom topologies.[45]

Installing Mininet is straightforward. The installation steps and commands are on the Mininet official page [26]. To install and test Mininet, use the following commands.

git clone https://github.com/mininet/mininet

cd mininet

git tag # list available versions

git checkout -b mininet-2.3.0 2.3.0 # or whatever version you wish to install

cd ..

mininet/util/install.sh [options]

Mininet available installation options are:

- **-a:** install everything included in the Mininet VM, including dependencies like Open vSwitch and additions like the OpenFlow Wireshark dissector and POX.
- **-nfv**: install Mininet, the OpenFlow reference switch, and Open vSwitch
- **-s mydir:** to place source/build trees somewhere other than the home directory, use this option before any other options.

From the above installation option for this thesis purpose, the -nvf option is enough to create the network. After the successful installation of Mininet, test if the installation works as expected using the below command.

sudo mn --test pingall

3.4.1.3 Installation and running of CBench.

OpenFlow benchmarking tool is a tool for testing OpenFlow controllers by generating packet-in events for new flows. The Floodlight documentation recommended CBench (controller benchmark). CBench emulates a bunch of switches that connect to a controller, send packet-in messages, and watch for flow mods to get pushed down [46]. CBench tool has two primary benchmarking tests; throughput and latency. A detailed description of how CBench works to test throughput and latency; and the CBench algorithm is described in Section 4.4.1

To install CBench, first install all dependencies using the following commands.

sudo apt-get install autoconf automake libtool libsnmp-dev libpcap-dev

After all the dependencies are fulfilled, use the following command to install CBench [46].

git clone https://github.com/mininet/oflops

cd oflops

sh boot.sh || true # possible error in autoreconf, so run twice

sh boot.sh

./configure --with-openflow-src-dir=<absolute path to OpenFlow branch>

make liboflops_test.la

make

sudo make install || true # make install fails; force past this.

Table 5: CBench parameters

Option	Description	Default value
-c/--Controller	<str> hostname of the controller to connect to	(“localhost”)
-d/--debug	enable debugging	(off)
-h/--help	print this message	
-l/--loops	<int> loops per test	(16)
-M/--mac-addresses	<int> unique source MAC addresses per switch	(100000)
-p/--port	<int> controller port	(6633)
-r/--ranged-test	test range of 1...$n switches	(off)
-s/--switches	<int> fake $n switches	(16)
-t/--throughput	test throughput instead of latency	
-C/--cooldown	<int> loops to be disregarded at the test end. (Cooldown)	(0)
-D/--delay	received (in ms) <int> delay starting. testing after features_reply	(0)
-L/--learn-dst-macs	macs before testing, send gratuitous ARP replies to learn the destination	(on)

CHAPTER 4 RESULTS AND DISCUSSIONS

4.1 Introduction

This chapter describes the implementation details and experimental results of the proposed design for an efficient routing algorithm for the Floodlight controller. Different experiments were performed to verify the efficiency of the proposed algorithm. Sections 4.2 and 4.3 present the proposed system's implementation and experimental setup. Section 4.4 describes the Floodlight controller evaluation benchmarking methodology. Section 4.5 presents the test results of the systems. Finally, discussions are made in the last sections of this chapter.

4.2 Implementation

The following command is used to run the Floodlight controller. During start-up processes, the controller loads the different modules and properties defined in Floodlight default properties and core modules.

sudo java -jar target/Floodlight.jar

The Floodlight controller, as Figure 4.1 shows, consists of a GUI that can be opened in any browser with the controller address after the controller runs with the command written above. It shows the network topology that Mininet, switches, and hosts of the specified network create.

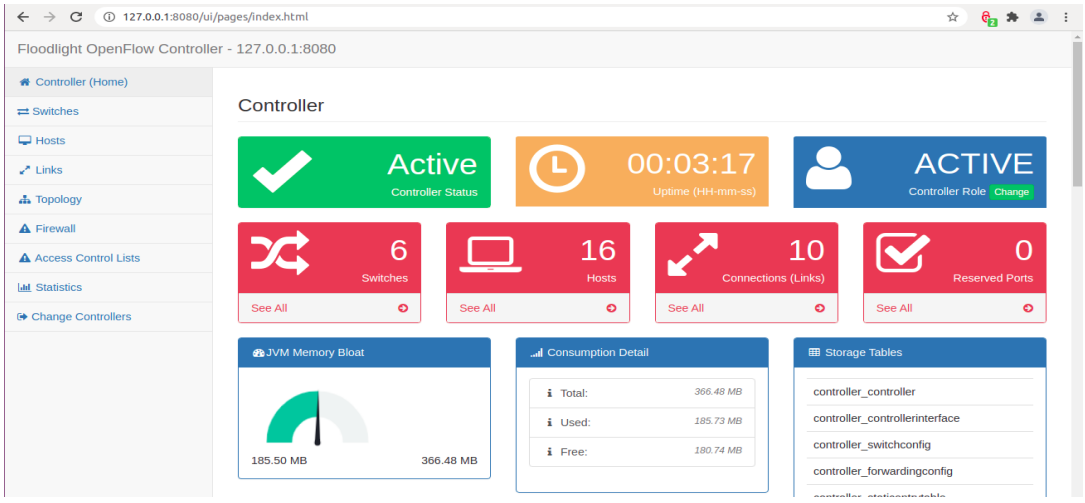

Figure 7: Floodlight GUI

4.3 Experimental Setup

The experimental setup to test the proposed system includes the following:

- Processor: Intel® Core™ i3-3110M CPU @ 2.40GHz ×
- RAM: 8 Gb
- Operating system: Linux Ubuntu 16.04LTS, x64 based
- The OpenFlow emulation software-Mininet will be used to create the test network, covered in Section 3.4.1.2. The switch that is used in this testbed is open vSwitch version 2.15.0.

The Floodlight controller is software that controls the devices of the network. Therefore, the controller does not have performance parameters like hardware because it is a type of network operating system and depends upon the underlying hardware. The controller's sole purpose is to manage the network and devices using the OpenFlow protocol. The test bed used to measure the system's performance comprises a controller and n number of switches. Each switch has n number of host devices connected to the switch with linear topology.

These switches and host devices are emulated with Mininet. A detailed explanation of the Mininet tool is included in Section 3.4.1.2. The network was created by Mininet using the following command, as shown in Fig 9

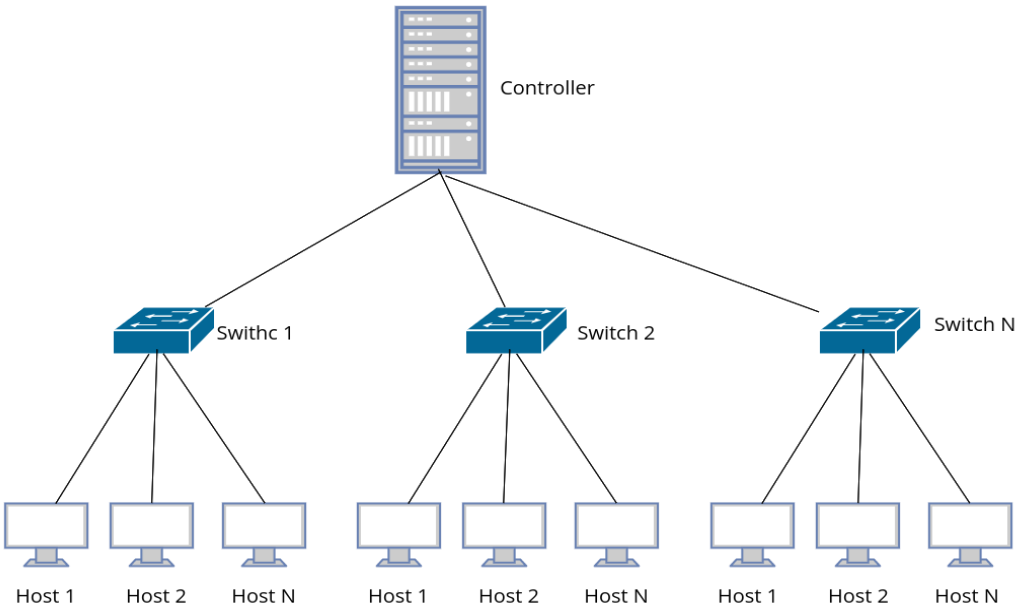

Figure 8: Topology setup

sudo mn --controller remote,ip=127.0.0.1,port=6653 --switch=ovsk --topo linear,16

The above commands are used to create the network at some remote controller. Let us discuss each of the terms one by one. Sudo mn command gives the administrator privilege for the users on Mininet and instructs Mininet to create the specified network on the controller's address that is specified, in this case, 127.0.0.1. Topo linear command specifies the network's topology, in this case linear, and creates a network of 16 switches. Moreover, the Controller's IP and port specify the type of controller to be connected to the network. The IP address and the port at which the controller has located The controller, as specified as 'remote,' indicates that the default controller inside Mininet is not used. However, the control located remotely outside the Mininet at the specified address will be used for network management.

The traffic used for testing the network is created by the controller, which is default traffic [48]. The following is the traffic generated by the controller to test the network.

1. ARP (Address Resolution Protocol). It is a protocol used by IPv4 to map IP network address to hardware address. This address resolution mechanism communicates ARP messages between the sender and receiver.

2. UDP (User Datagram Protocol). It is a connectionless protocol used primarily for establishing loss-tolerating and low-latency connections between applications on the internet.

3. TCP (Transmission Control Protocol). is a connection-oriented protocol for exchanging messages and error-free data transmission. It supports the retransmission of dropped packets and the acknowledgment of all received packets.

Three different types of traffic are used for testing the network. ARP is used for mapping the address, UDP is for sending any amount of traffic without waiting for the response from the receiver, and TCP is used for adding reliability to the sent data. This variety of traffic helps in testing the network under different workloads.

I have used a ping command to send a packet in a host-host connection in the network setup. As illustrated in Figure 8, the first host sends a request for a MAC address, and the switch transmits the PacketIn messages to the controller. Then the controller responds with the

packetOut message instructing the switch to forward packetOut messages to all ports. When the controller obtains the reply, it sends a message for the switch to install new flows.

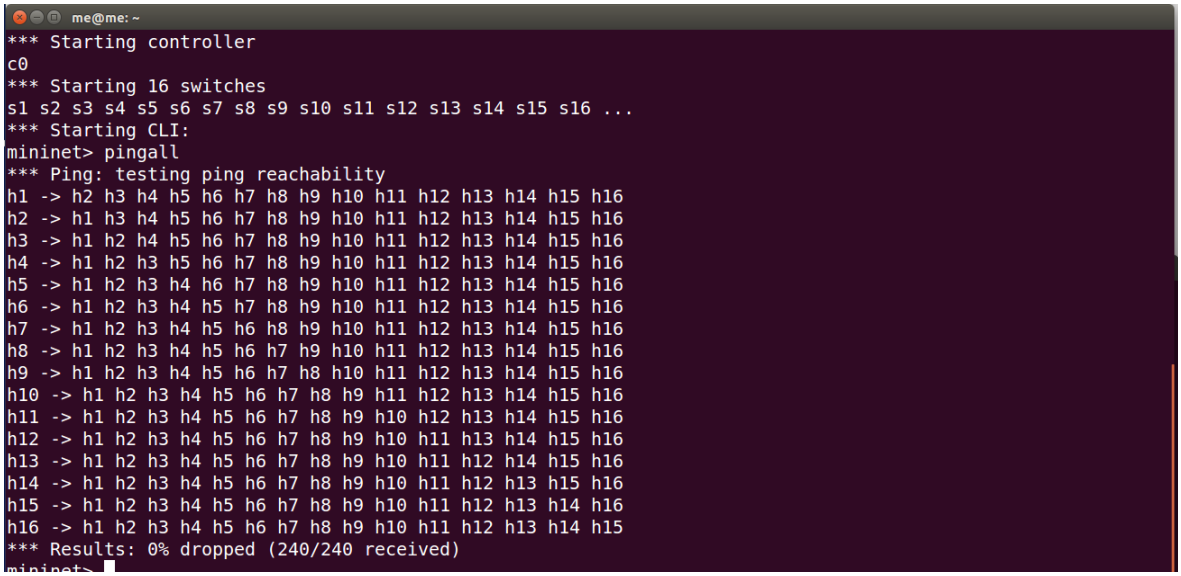

Figure 9: Connectivity test

The test runs on the same device with the learning-switch application and the CBench evaluation tool, as described in section 4.4, to evaluate the throughput and latency. The CBench tool emulates 16 switches connected directly to the controller. Each switch has connected devices, which are connected directly to the controller. A test has been performed on increasing the number of switches available (16 Switches) to send the maximum number of Messages PacketIn to evaluate the controller's performance.

4.4 Floodlight Controller Evaluation Benchmarking Methodology

Throughput, packet loss, and latency benchmarking evaluate the controller's performance.

4.4.1 CBench Algorithm

To perform throughput and latency tests on the Controller CBench uses the following algorithm [46].

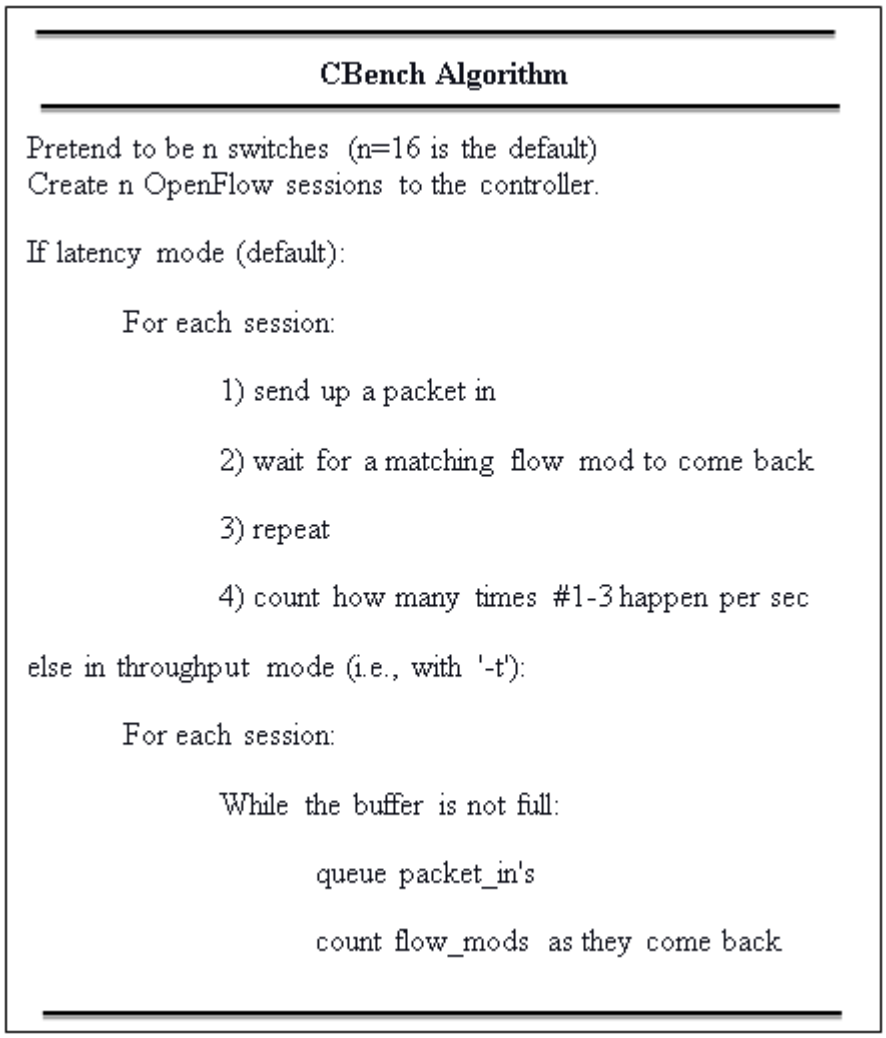

Figure 10: CBench algorithm

4.4.2 Throughput

In the throughput mode, the controller will be made to manage a large amount of traffic in which the switch continuously sends the PacketIn messages to the controller within a period and watches for the total number of flows installed per second on that switch.[49]

4.4.3 Latency

Latency measures the ability of OpenFlow controllers to measure the speed of incoming packet In's intention to reply to switches quickly. In latency mode, the switches generate a packet In message and wait for the response from the controller before sending the following message. This procedure will be done as quickly as possible. Then the total number of responses per millisecond was counted and calculated to measure the average per-packet latency the controller took to process.[49]

4.4.4 Response Time

Response time is the time that algorithms require to address a specific client request. It covers the whole amount of waiting, transmission, and service time the system needs. Consequently, the objective of performance algorithm optimization will be to reduce response time.

4.4.5 Packet Loss

Packets are small units of data that are sent across a network from one source to destination. Packet loss occurs when a network packet does not arrive at its intended destination, resulting in data loss. Packet loss can be expressed as a percentage based on received packets compared to send packets.

To calculate packet loss iperf [50] tool with different command line arguments was used to generate the traffic. After generating the traffic and capturing it in the files, the network's throughput, delay, and packet loss were calculated. For doing so, the most distant two hosts were used as iperf client and server. Host 1, which is connected to switch 1, is the server, and Host 16, which is connected to switch 16, is the client. To generate traffic with iperf, the following commands and parameters are

used.

To make h1 a server machine

Iperf3 -s -p 6653

To make h16 a client machine

Iperf3 –c <server Ip address> -p 6653 -u -t 60 -i 5 -b 10M

The arguments used above are described below:

-s: is used to run a host as a server machine

-c: is used to run a host as a client machine

-p: port

-u: is used for generating UDP

-t: is the duration for which the client will run

-i: is the interval on which the server will measure the traffic

-b: bandwidth send in bits/sec

4.4.5 Floodlight Controller Benchmarking Parameters

For evaluating the performance of the controller, the following parameters were used for testing both latency and throughput:

- The number of OpenFlow-enabled switches:- the number of OpenFlow switches will establish a connection with the controller.
- Number of nodes:- number of hosts connecting to the switch.
- OpenFlow Version: OpenFlow protocol version that the controller will use for connection setup, OpenFlow protocols such as 1.1, 1.2, 1.3,1.4, and 1.5
- Test loops: the number of frequencies the test needs to be repeated.
- Test Duration: the duration of test iteration, articulated in seconds.

In this test, 14 loops of tests were performed, of which the first 4 were warm-up tests that were omitted when calculating the overall test result. Then the average of these ten tests appears in the overall test results. Also, a delay was added before starting the test to wait for some handshake process between the switch and controller. Therefore, in this evaluation, a delay of 60 milliseconds is used, as shown in Figures 9 and 10. For benchmarking purposes, I substitute LearningSwitch in place of Hub and run Floodlight with -cf src/main/resources/floodlightbenchmarking.properties

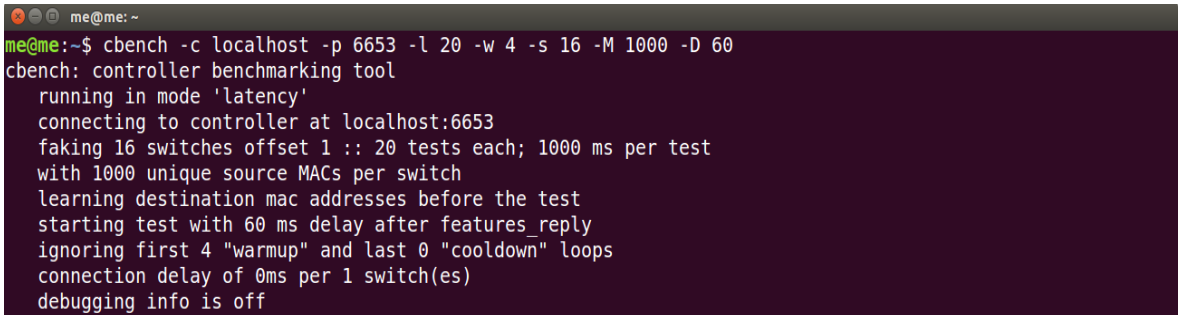

Figure 11: CBench running in latency mode.

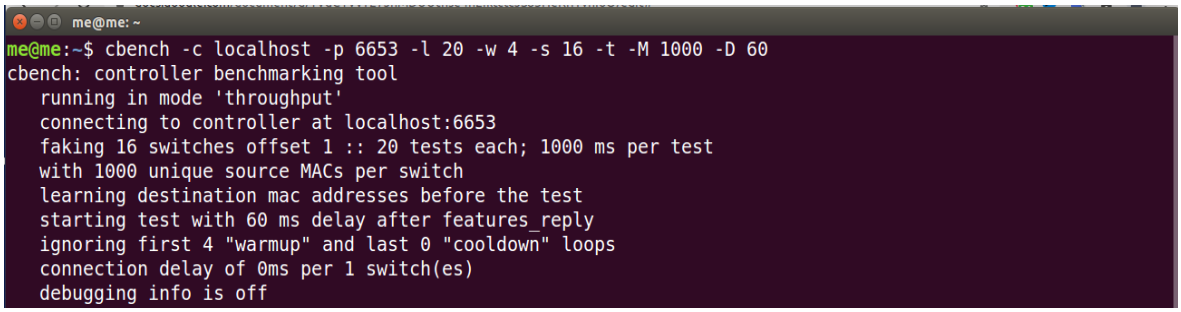

Figure 12: CBench running in throughput mode.

4.5 Test Results

Throughput and latency are two main parameters that have been used to measure the performance of a system. We created the network using Mininet based on the learning switch app. Evaluation is performed separately for the enhanced version and the regular Floodlight controller. At each evaluation, the parameters are tested on increasing switches by keeping the number of hosts constant.

4.5.1 Throughput

The first parameter to evaluate performance is throughput. Throughput measures the average number of flows that are installed in each switch.

Throughput is the number of transactions per second that an application can manage. There is another way to measure this parameter, one that will use bits per second (bit/s or bps), and another way to measure throughput performance is through flows per second. So, in this evaluation, flows per second are used to evaluate the throughput of the controller. The maximum flow measurement results for the Floodlight controller, shown in Figure 13, clearly demonstrate that the controller can process a maximum of 191,900 flows/s (flow rates) with the improved and 162,528 flows/s with the regular Floodlight controller.

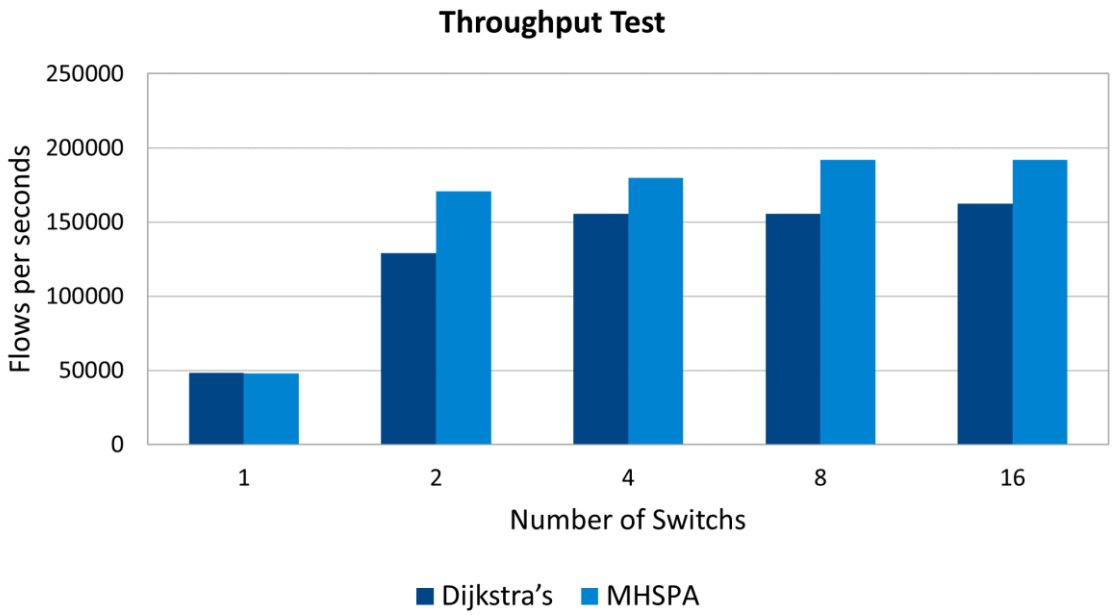

Figure 13: Throughput test

4.5.2 Latency

The latency results represent the average number of milliseconds a flow consumes to be installed on each switch.

The latency results represent the average number of milliseconds a thread takes to install per switch. Latency in a packet-switched network is the time it takes for a packet to reach its destination through the network. The physical environment or devices that make up the network (switches, routers) can cause delays. There are two approaches to measuring latency: the first is measuring the time it takes for a packet to travel from source to destination (one-way), and the second is measuring the time a packet takes to go to and from (round-trip). This last form of latency measurement, Round-Trip, is the most used because it allows a single device to measure network latency used in this paper.

The maximum latency measurement of the Floodlight controller, as shown in figure 14, clearly shows the maximum latency of 50 flows per millisecond for improved and 51 flows per millisecond for regular Floodlight controllers.

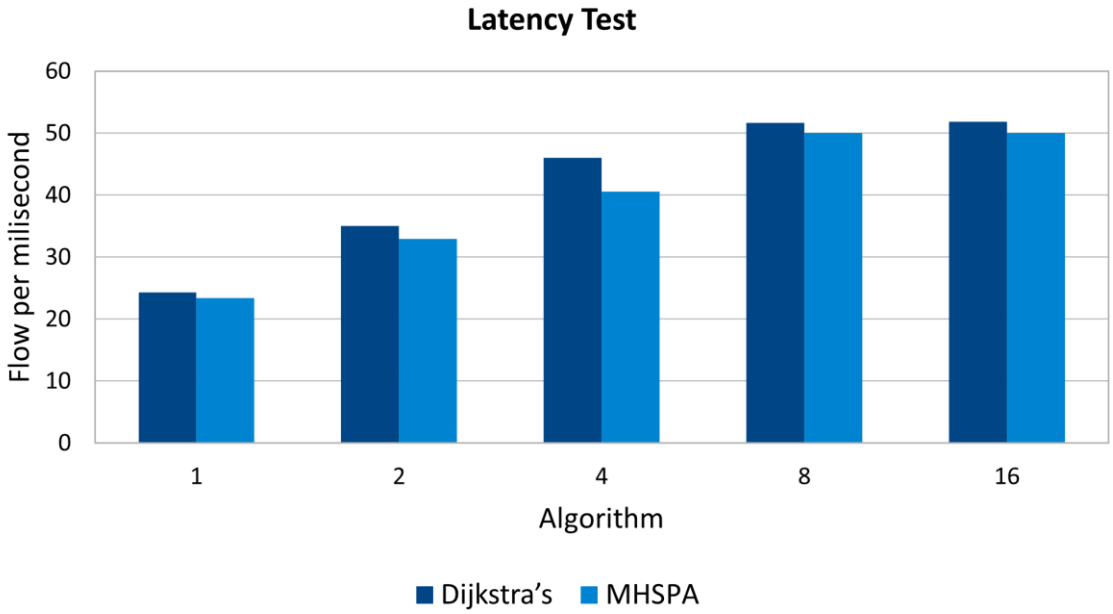

Figure 14: Latency test

4.5.3 Response Time

The time it takes algorithms to respond to a specific client request is referred to as response time. It accounts for the entire amount of waiting, transmission, and service time required by the system. As a result, the goal of performance algorithm optimization will be to reduce response time. The response time of the Floodlight SDN controller with MHSPA and Dijkstra's algorithms is presented in table 6 below.

Table 6: Response time (sec)

Concurrent users	**Algorithms**	
	Dijkstra's	**MHSPA**
100	0.12	0.12
200	0.17	0.16
300	0.33	0.30
400	0.62	0.57
500	0.64	0.65
600	0.72	0.70
700	0.75	0.80

800	0.90	0.87
900	1.12	1.00
1000	1.14	1.12

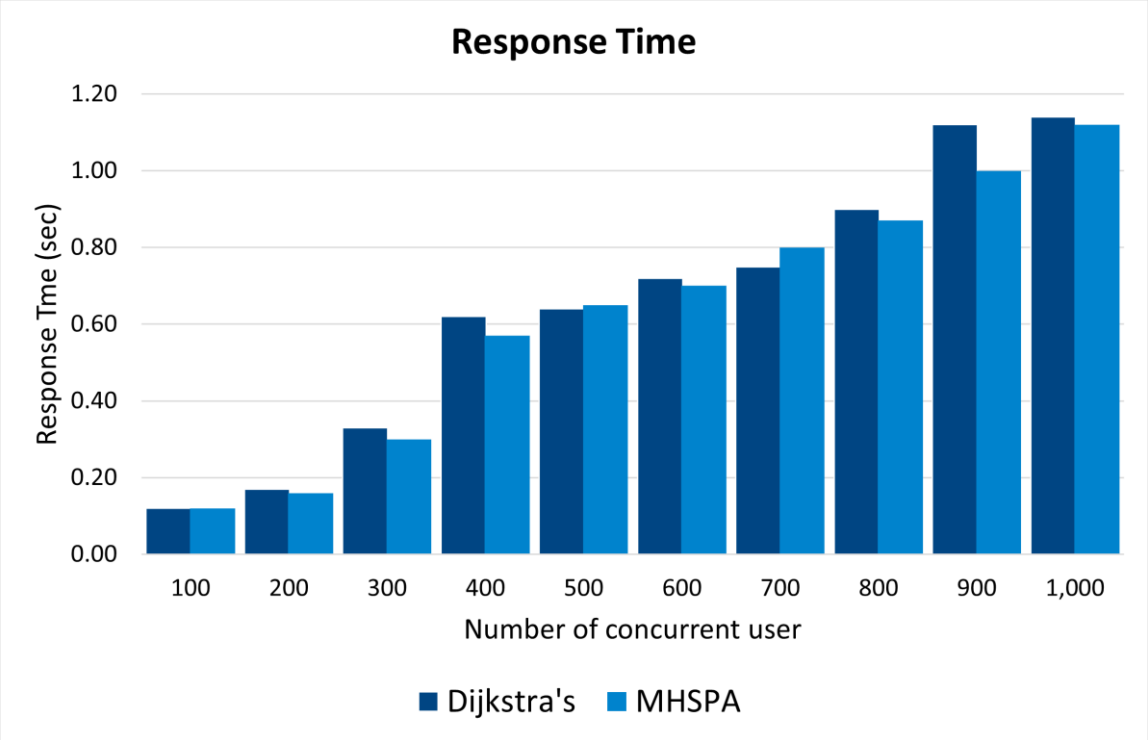

Figure 15: Response time

As we see in the response time graph in figure 15, the response time of MHSPA and Dijkstra's algorithms increases as the number of concurrent users increases. However, the response time of the MHSPA algorithm is less than when compared to Dijkstra's algorithm. The MHSPA algorithm has low latency and fast round trip time. However, the response time of Dijkstra's algorithm has a lesser response time of 0.64 and 0.75 (sec) at 500 and 700 concurrent users, respectively.

Moreover, when we see the overall response time of the controller, the MHSPA has a lesser response time than Dijkstra's algorithm. MHSPA has an average response time improvement of 3.34% over the Dijkstra's. This improvement was achieved because MHSPA reduced the number of nodes to check and faster optimal path calculation than Dijkstra's.

4.5.4 Packet Loss

Packets are small data sent across a network from one source to the destination [52]. Packet loss occurs when a network packet does not arrive at its intended destination, resulting in data loss. Packet loss can be expressed as a percentage based on received packets compared to send packets.

After sending UDP traffic using iperf3 commands for 60 seconds from H1 to the longest node, H16, the results are captured below in table 7.

Table 7: Packet loss

Algorithm	Sent	Received	Lost
Dijkstra's	9141	8983	158
MHSPA	9141	9010	131

After sending 1941 datagrams for 60 seconds, the regular controller lost 158, and MHSPA lost 131.

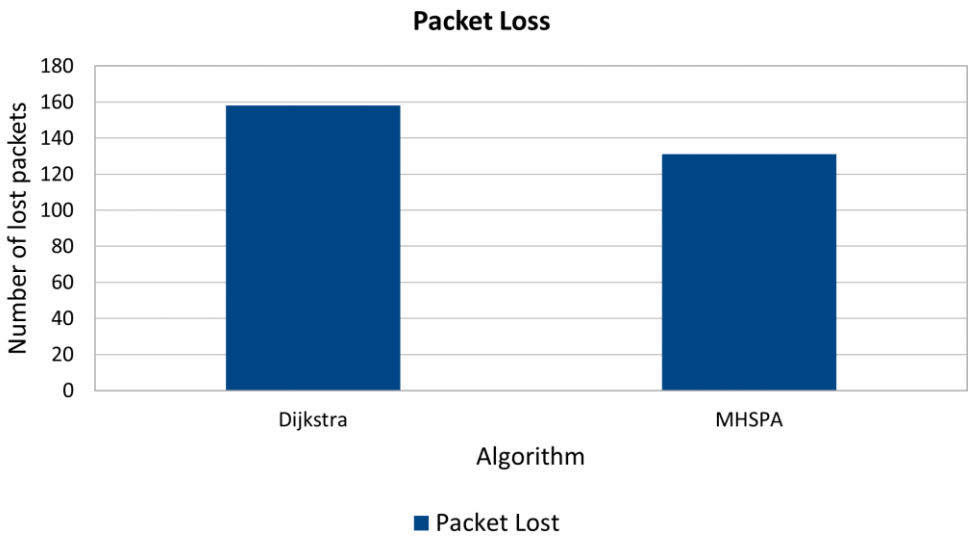

Figure 16: Packet loss

4.6 Discussions

The test result of implementing a multi-heuristic shortest path algorithm to the Floodlight SDN controller and the regular controller is described in the previous section with throughput, packet loss, response time, and latency. This section described test results by comparing the modified version with a default Floodlight controller.

As shown in Figures 13 and 14, the improved version of the Floodlight controller has an average throughput increment of 22.27% over a regular Floodlight controller and an average latency decrement of 5.63%. This improvement is obtained because implementing the multi-heuristic routing algorithm produces an optimal path and reduces unnecessary path checking and calculation. This improvement helps the controller install many flows to the switch within a second and respond quickly.

Also, implementing MHSPA into the Floodlight SDN controller has a 17.09% improvement in packet delivery, as depicted in figure 16. The regular Floodlight SDN controller has more packet loss than the improved version. This means MHSA has more stability in real-time applications than the regular Floodlight controller.

Implementing the multi-heuristic algorithm to the controller has a significant advantage for SDN-based network companies and cloud providers. It helps allow them to optimize their productivity by improving system performance. Increasing throughput and reducing latency helps them reduce various costs. In addition, it makes it easy for them to work with tedious network management tasks involving configuring various devices to the network with the OpenFlow protocol. Therefore, implementing the multi-heuristic algorithm with Floodlight's SDN controller makes network management more straightforward. Moreover, deploying new services with this new Floodlight controller is effortless. In addition, servers can respond quickly to user needs by deploying a multi-heuristic algorithm implemented in the SDN controller.

As the network size increases, more data plane requests are generated and must be satisfied. In a vast network, the number of requests processed at once increases, leading to a decrement in throughput and an increment in latency. Implementing the multi-heuristic algorithm helps to boost performance and improve throughput and latency.

How can implementing the multi-heuristic shortest path algorithm improve the performance of the Floodlight SDN controller? This is the research question presented in the first chapter. Implementing the multi-heuristic algorithm enhances the performance of the Floodlight SDN controller. Figures 13 and 16 illustrate that both throughput and packet delivery ratio significantly improve for the same test environment.

The latency of the multi-heuristic algorithm-based Floodlight SDN controller was decreased because it expanded only to the destination direction, not in every direction evenly. This minimizes unnecessary node checking. The delay was reduced due to a few nodes checking until the destination node was found.

The throughput of the multi-heuristic algorithm-based Floodlight SDN controller improved due to the algorithm generating an optimal path and minimizing unnecessary path checking and calculation. The throughput maximizes as the controller can install many flows to the switch within a short amount of time.

CONCLUSIONS AND FUTURE WORKS

In this thesis, I have proposed and implemented the multi-heuristic shortest path algorithm for the Floodlight SDN controller, studying different SDN technologies, specifically, Floodlight controllers design features and OpenFlow protocol. The performance of the controller was evaluated using the CBench tool. Literature reveals that the current Floodlight controller was built with Dijkstra's shortest path algorithm, which could waste resources by checking unnecessary nodes to the destination. The Multi-heuristic shortest path algorithm was used to optimize the shortest path finding mechanism as well as the performance of the Floodlight controller.

The proposed multi-heuristic technique was implemented on the Floodlight SDN controller, the central part on which different network control applications are deployed. The controller lies between the infrastructure layer, which consists of the different network devices, and the application layer. Therefore, the controller is responsible for managing the resource of the whole network using the OpenFlow protocol.

The controller administers the underlying networking device, such as OpenVswitch, by updating the flow entry table containing different information about the packets. When a new packet arrives, the switch transmits the packet information to the controller and requests the best path to the destination. The controller calculates the path using heuristic information.

The designed multi-heuristic shortest path algorithm for the Floodlight controller was implemented by building a modified version as described above. After that, the performance of both regular and MHSPA-based Floodlight controllers was evaluated on the same device and with the same metrics. The SDN network was created on the learning-switch application to evaluate their performance. Then, separately running the different versions of the Floodlight controller and testing was done using the CBench and iperf tool. In CBench, throughput and latency metrics are tested. And in the iperf tool, the packet loss metric is tested. Throughput measures the total number of flows the switch installs within a second interval. Simultaneously, latency measures the time it takes the flow to be installed on the switch.

As shown in Figure 13, I conclude from the Floodlight OpenFlow controller's evaluation that the multi-heuristic shortest path algorithm Floodlight shows the highest throughput,

achieving 191,900 flows per second. Moreover, the throughput result for the regular Floodlight controller yields a throughput of 162,528 flows/sec. The multi-heuristic shortest path algorithm Floodlight controller has an average of 22.27% throughput enhancement.

Latency tests for the controller were run with various numbers of connected switches to see how the response time of these controllers would change as more switches were added, see Figure 14. The tests showed that MHSPA Floodlight has the highest latency of 50 flows per millisecond and 51 flows per millisecond for Normal Floodlight controllers. Generally, the MHSPA Floodlight has an average of 5.63% latency improvement than the regular Floodlight controller.

As shown in Figure 15, the response time of MHSPA and Dijkstra's algorithms increases as the number of concurrent users increases. However, the response time of the MHSPA algorithm is less than when compared to Dijkstra's algorithm. Dijkstra's and MHSPA have a maximum response time of 1.12 and 1.14 seconds, which is a little more than the acceptable range. MHSPA has an average improvement of 3.34% in response time.

Additionally, as illustrated in figure 16, adding MHSPA to the Floodlight SDN controller improves packet delivery by 17.09%. The standard Floodlight SDN controller has higher packet loss than the regular controller. This indicates that MHSPA is more stable than a standard Floodlight controller in real-time applications.

Implementing a multi-heuristic shortest path algorithm to the Floodlight controller has resulted in an average enhancement of 22.27% throughput, 17.09% packet loss, and 5.63% latency. The throughput was improved, and the latency and packet loss were reduced with this proposed mechanism. In my test, communications between the switches and the controller are done via the loopback interface, which has no latency when packets traverse virtual ports versus hardware ports. Therefore, the latency results are measured higher in actual hardware implementation due to virtual ports compared to it.

I have designed and implemented the MHSPA to enhance the performance of the Floodlight SDN controller. Moreover, with this methodology, I have optimized the controller's performance. However, incorporating the following ideas as future work would have a better result.

- I compared the performance of the Floodlight controller to that of the optimized one using linear topology. Furthermore, testing with other topologies will determine the performance of the Floodlight controller under different topologies.
- The MHSPA was implemented in the Floodlight Controller. Additionally, implementing and evaluating the algorithm under different SDN controllers will determine the algorithm's performance.

REFERENCES

[1] A. Shivendu, D. Dhakal, and D. Sharma, “Emulation of Shortest Path Algorithm in Software Defined Networking Environment,” International Journal of Computer Applications, vol. 116, no. 1, pp. 47–49, Apr. 2015, doi: https://doi.org/10.5120/20304-2344.

[2] P. Tejaswini, T. Divyavani, and R. Manikandan, “Efficient Routing Approach In Software Defined Networking,” International Journal of Pure and Applied Mathematics, vol. 119, no. 12, pp. 14155–14163, Jan. 2018.

[3] M. Jawad, M. Salam, and A. Tariq, “Floodlight Controller onto Load Balancing of SDN Management Floodlight Controller onto Load Balancing of SDN Management,” vol. 04, no. 08, p. PP. 124-131, Aug. 2017.

[4] S. Asadollahi and B. Goswami, “Experimenting with scalability of floodlight controller in software defined networks,” 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Dec. 2017, doi: https://doi.org/10.1109/iceeccot.2017.8284684.

[5] Q. Martiña and D. Jose, “Research on path establishment methods performance in SDN-based networks,” Master’s Thesis, Universitat Politècnica de Catalunya, 2015.

[6] 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.

[7] Z. of the J. W. K. and T. 1st I. C. on A. \& S. I. Ramadhan, Siahaan, and M. Mesran, “Prim and floyd-warshall comparative algorithms in shortest path problem,” in Proceedings of the Joint Workshop KO2PI and The 1st International Conference on Advance \& Scientific Innovation, 2018, pp. 47–58.

[8] L. Yang, B. Ng, W. K. Seah, L. Groves, and D. Singh, “A survey on network forwarding in Software-Defined Networking,” Journal of Network and Computer Applications, vol. 176, p. 102947, 2021.

[9] “Software-Defined Networking (SDN) Definition,” Open Networking Foundation. https://opennetworking.org/sdn-definition/ (accessed Oct. 31, 2022).

[10] M. Oppitz and P. Tomsu, “Software Defined Virtual Networks,” Inventing the Cloud Century, pp. 149–200, Aug. 2017, doi: https://doi.org/10.1007/978-3-319-61161-7_8.

[11] “Open Networking Foundation Software-Defined Networking: The New Norm for Networks,” Open Networking Foundation. https://www.opennetworking.org (accessed Oct. 31, 2022).

[12] K. Nisar et al., “A survey on the architecture, application, and security of software defined networking: Challenges and open issues,” Internet of Things, vol. 12, p. 100289, 2020.

[13] O. Blial, B. Mamoun, and R. Benaini, “An overview on SDN architectures with multiple controllers,” Journal of Computer Networks and Communications, vol. 2016, 2016.

[14] J. H. Cox et al., “Advancing software-defined networks: A survey,” Ieee Access, vol. 5, pp. 25487–25526, 2017.

[15] S. H. Haji et al., “Comparison of software defined networking with traditional networking,” Asian Journal of Research in Computer Science, vol. 9, Art. no. 2, 2021.

[16] A. Alghamdi, D. Paul, and E. Sadgrove, “Designing a RESTful northbound interface for incompatible software defined network controllers,” SN Computer Science, vol. 3, Art. no. 6, 2022.

[17] “OpFlex control protocol,” IETF Datatracker. http://tools.ietf.org/html/draft-smith-opflex-00 (accessed Oct. 31, 2022).

[18] “Network configuration protocol (NETCONF),” Ietf.org. http://www.ietf.org/rfc/rfc6241.txt (accessed Oct. 31, 2022).

[19] N. Gupta, Maashi, Mashael S, S. Tanwar, S. Badotra, M. Aljebreen, and S. Bharany, “A comparative study of software defined networking controllers using mininet,” Electronics, vol. 11, Art. no. 17, 2022.

[20] X. Tang, X. Zeng, and L. Song, “Accelerating protocol oblivious forwarding programmable data plane with flow cache,” IEEE, 2022.

[21] L. Mamushiane, A. Lysko, and S. Dlamini, A comparative evaluation of the performance of popular SDN controllers. 2018, pp. 54–59. doi: https://doi.org/10.1109/WD.2018.8361694.

[22] O. Blial, B. Mamoun, and B. Redouane, “An Overview on SDN Architectures with Multiple Controllers,” Journal of Computer Networks and Communications, vol. 2016, pp. 1–8, Jan. 2016, doi: https://doi.org/10.1155/2016/9396525.

[23] W. Li, W. Meng, and L. F. Kwok, “A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures,” Journal of Network and Computer Applications, vol. 68, pp. 126–139, Jun. 2016, doi: https://doi.org/10.1016/j.jnca.2016.04.011.

[24] A. AlbuSalih, “Performance Evaluation of Ryu Controller in Software Defined Networks,” vol. 14, p. 1, Feb. 2022, doi: https://doi.org/10.29304/jqcm.2022.14.1.879.

[25] “Floodlight Controller - Project Floodlight,” floodlight.atlassian.net. https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview (accessed Oct. 31, 2022).

[26] “Open Network Operating System (ONOS) SDN Controller for SDN/NFV Solutions,” Open Networking Foundation. https://opennetworking.org/onos/ (accessed Oct. 31, 2022).

[27] “Home - OpenDaylight,” OpenDaylight, 2017. https://www.opendaylight.org/

[28] noxrepo, “noxrepo/pox: The POX network software platform,” GitHub, Nov. 23, 2017. https://github.com/noxrepo/pox (accessed Oct. 31, 2022).

[29] “Ryu SDN Framework,” ryu-sdn.org. https://ryu-sdn.org/ (accessed Oct. 31, 2022).

[30] “Big Switch Networks, Inc. | The Leader in Open Software Defined Networking.” http://www.bigswitch.com/ (accessed Oct. 31, 2022).

[31] D. Kannan and R. Thiyagarajan, “Entropy based TOPSIS method for controller selection in software defined networking,” Concurrency and Computation: Practice and Experience, vol. 34, Art. no. 1, 2022.

[32] “LinkDiscoveryManager (Dev) - Floodlight Controller - Confluence,” floodlight.atlassian.net. https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343628/LinkDiscov (accessed Oct. 31, 2022).

[33] S. Kotachi, T. Sato, R. Shinkuma, and E. Oki, “Multicast routing model to minimize number of flow entries in software-defined network,” IEICE Transactions on Communications, vol. 104, Art. no. 5, 2021.

[34] S. Torkzadeh, H. Soltanizadeh, and A. A. Orouji, “Energy-aware routing considering load balancing for SDN: a minimum graph-based Ant Colony Optimization,” Cluster Computing, vol. 24, no. 3, pp. 2293–2312, Mar. 2021, doi: https://doi.org/10.1007/s10586-021-03263-x.

[35] S. M. Raza, S. Ahvar, R. Amin, and M. Hussain, “Reliability Aware Multiple Path Installation in Software-Defined Networking,” Electronics, vol. 10, no. 22, p. 2820, Nov. 2021, doi: https://doi.org/10.3390/electronics10222820.

[36] R. Biswas and J. Wu, “Minimizing the Number of Rules to Mitigate Link Congestion in SDN-based Datacenters,” 2021 IEEE International Conference on Networking, Architecture and Storage (NAS), Oct. 2021, doi: https://doi.org/10.1109/nas51552.2021.9605365.

[37] Z. Zhao, W. Yang, and B. Wu, “Flow aggregation through dynamic routing overlaps in software defined networks,” Computer Networks, vol. 176, p. 107293, Jul. 2020, doi: https://doi.org/10.1016/j.comnet.2020.107293.

[38] A. Shirmarz and A. Ghaffari, “An adaptive greedy flow routing algorithm for performance improvement in software-defined network,” International Journal of Numerical Modelling: Electronic Networks, Devices and Fields, vol. 33, no. 1, Jan. 2020, doi: https://doi.org/10.1002/jnm.2676.

[39] I. E. Salem, M. M. Mijwil, A. W. Abdulqader, and M. M. Ismaeel, “Flight-schedule using Dijkstra’s algorithm with comparison of routes findings,” International Journal of Electrical and Computer Engineering (IJECE), vol. 12, no. 2, p. 1675, Apr. 2022, doi: https://doi.org/10.11591/ijece.v12i2.pp1675-1682.

[40] K. P. Kumar Rao and T. Senthil Murugan, “An Efficient Routing Algorithm for Software Defined Networking using Bellman Ford Algorithm,” International Journal of Online and Biomedical Engineering (iJOE), vol. 15, no. 14, p. 87, Oct. 2019, doi: https://doi.org/10.3991/ijoe.v15i14.11546.

[41] A. Almohaimeed and A. Asaduzzaman, “A Self-Routing Technique for Software-Defined Networks,” 2019 SoutheastCon, Apr. 2019, doi: https://doi.org/10.1109/southeastcon42311.2019.9020564.

[42] “Open vSwitch,” www.openvswitch.org. https://www.openvswitch.org/ (accessed Oct. 31, 2022).

[43] Canonical, “The leading operating system for PCs, IoT devices, servers and the cloud | Ubuntu,” Ubuntu, Jun. 19, 2019. https://ubuntu.com/ (accessed Oct. 31, 2022).

[44] “IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains,” JetBrains. https://www.jetbrains.com/idea (accessed Oct. 31, 2022).

[45] Mininet Team, “Download/Get Started with Mininet - Mininet,” Mininet.org, 2018. http://mininet.org/download/ (accessed Oct. 31, 2022).

[46] “CBench-mininet/oflops,” GitHub. https://github.com/mininet/oflops/tree/master/cbench (accessed Oct. 31, 2022).

[47] “Java|oracle,” Java.com, Apr. 14, 2019. https://www.java.com/en/ (accessed Oct. 31, 2022).

[48] R. Izard, “How to Process a Packet-In Message.” https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/9142279/How+to+Process+a+Packet-In+Message (accessed Oct. 31, 2022).

[49] A. Akinola, M. Adigun, and P. Mudali, “Performance Modeling of SoftwareDefined Networking Paradigm in a Public Health Management System,” International Journal of Information and Management Sciences, vol. 2, pp. 123–148, Jul. 2020.

[50] V. GUEANT, “iPerf - The TCP, UDP and SCTP network bandwidth measurement tool,” Iperf.fr, 2013. https://iperf.fr/ (accessed Oct. 31, 2022).

[51] B. Isyaku, K. B. A. Bakar, F. A. Ghaleb, and A. Al-Nahari, “Dynamic Routing and Failure Recovery Approaches for Efficient Resource Utilization in OpenFlow-SDN: A Survey,” IEEE Access, vol. 10, pp. 121791–121815, 2022, doi: https://doi.org/10.1109/access.2022.3222849.

[52] A. Verma, R. Saha, N. Kumar, G. Kumar, and Tai-Hoon-Kim, “A detailed survey of denial of service for IoT and multimedia systems: Past, present and futuristic development,” Multimedia Tools and Applications, vol. 81, no. 14, pp. 19879–19944, May 2022, doi: https://doi.org/10.1007/s11042-021-11859-z.

[53] “Siege Home.” https://www.joedog.org/siege-home/ (accessed Mar. 06, 2023).