مراجعة لموازنة حمل الخوادم في الشبكات المعرفة برمجياً

م.ميساء الضاهر[1] د.ماهر عباس[2] د.محسن عبود[3]

الملخص

تقدم هذه المراجعة تحليلاً شاملاً للعديد من الدراسات البحثية التي تركز على تقنيات موازنة أحمال الخوادم في سياق الشبكات المعرفة بالبرمجيات (SDN). عرضت بعض هذه الدراسات منهجيات وخوارزميات متنوعة لتحسين كفاءة النظام والحفاظ على استقراره في ظل ظروف حمل متغيرة، بينما قارنت دراسات أخرى هذه المنهجيات والأساليب لتحديد الحلول الأكثر فعالية. تناقش هذه المقالة بنية الشبكة المستخدمة والمقاييس الرئيسية التي دُرست في كل مقالة، مثل زمن الاستجابة، والإنتاجية، وزمن الوصول، واستخدام الموارد، وهي مؤشرات أساسية للأداء في بيئات الشبكات المعرفة بالبرمجيات. كما تسلط المراجعة الضوء على الإعدادات التجريبية وأدوات المحاكاة الشائعة الاستخدام في الأدبيات، مما يوفر رؤى حول كيفية تقييم الباحثين لتقنياتهم المقترحة والتحقق من صحتها. بالإضافة إلى ذلك، يتناول البحث التوجهات المستقبلية التي اقترحتها الدراسات لتسليط الضوء على مجالات الابتكار المحتملة في هذا المجال، بما في ذلك دمج الذكاء الاصطناعي، والتعلم الآلي، وآليات اتخاذ القرار الذكية لتعزيز الأداء في ظروف الشبكة الديناميكية. ثم تقدم هذه الورقة ملخصًا واضحًا للدراسات البحثية في جدول منظم يسهل الفهم ويدعم جهود البحث المستقبلية في موازنة أحمال الخوادم ضمن بيئة الشبكات المعرفة بالبرمجيات. تهدف هذه المراجعة إلى أن تكون مرجعًا قيّمًا للأكاديميين والممارسين الذين يسعون إلى تحسين توزيع الأحمال وتحسين أداء الأنظمة القائمة على الشبكات المعرفة بالبرمجيات (SDN).

[1] م.ميساء الضاهر طالبة دكتوراه في كلية الهندسة المعلوماتية، قسم هندسة الشبكات والنظم الحاسوبية، جامعة حمص.
[2] د.ماهر عباس أستاذ مساعد في كلية الهندسة المعلوماتية، قسم هندسة الشبكات والنظم الحاسوبية، جامعة حمص.
[3] د.محسن عبود مدرس في كلية الهندسة المعلوماتية، قسم هندسة الشبكات والنظم الحاسوبية، جامعة حمص.

الكلمات المفتاحية: الشبكات المعرَفة برمجياً ، موازنة حمل الخادم، الخوارزميات، أداء الشبكة.

Review of Server Load Balancing in Software Defined Networks

Mayssaa Aldaher[1], Dr. Maher Abbas[2], Dr. Mohssen Abboud[3]
[1] PhD Student, Department of Network Engineering and Computer Systems, Homs University, Syria (E-mail: maldaher@homs-univ.edu.sy).
[2] Assistant Professor, Department of Network Engineering and Computer Systems, Homs University, Syria (E-mail: mabbaas@homs-univ.edu.sy)
[3]Lecturer, Department of Network Engineering and Computer Systems, Homs University, Syria (E-mail: mabboud@homs-univ.edu.sy).

Abstract—This review provides a comprehensive analysis of many research studies focused on server load balancing techniques in the context of software-defined networks (SDNs). Some of these studies presented diverse methodologies and algorithms used to improve system efficiency and maintain stability under variable load conditions, while others compared these methodologies and approaches to identify the most effective solutions. Our article discusses the topology used and the key metrics studied in each article, such as response time, throughput, latency, and resource utilization, which are essential indicators of performance in SDN environments. The review also highlights the experimental setups and simulation tools commonly used in the literature, offering insights into how researchers evaluate and validate their proposed techniques. In addition, the research addresses future directions suggested by the studies to highlight potential areas of innovation in this field, including the integration of artificial intelligence, machine learning, and intelligent decision-making mechanisms to enhance performance in dynamic network conditions. This paper then provides a clear summary of the research studies in a structured table that facilitates understanding and supports future research efforts in server load balancing within an SDN environment. The review aims to serve as a valuable reference for academics and practitioners seeking to optimize load distribution and improve the performance of SDN-based systems.

Keywords: Software Defined Network; Server Load Balancing; Algorithms; Network Performance

I. INTRODUCTION

Load balancing is now an irreplaceable element of contemporary network architectures, which guarantees that the load is efficiently shared among the servers and eliminates a bottleneck that leads to performance degradation. In software-defined networks (SDNs), load balancing is even more important because the control and data planes are decoupled, and users can centrally control traffic flows using their controllers. This programmability also opens the doors to adaptive and intelligent mechanisms that are stronger than the tools of traditional, static load balancers.

The load balancing of servers in SDN is a thorny issue that is yet to be resolved. Most of the current methods use simplified topologies, are based on test cases or are based on early algorithms to simulate the dynamics and heterogeneity of real traffic. Others are more concentrated on the link-level or routing-level balancing and the optimization of the servers is not fully explored. Moreover, there are no standardized assessment datasets and benchmarking procedures, making the comparison of findings across studies a challenging issue on which the applicability of suggested solutions can be restricted. These flaws highlight the urgency of critical reviews that find trends, limitations, and recommend directions of future innovation.

Recent literature points to a number of new trends such as the use of fuzzy logic, machine learning, and adaptive decision-making models to enhance response time, throughput and the overall quality of service. Nevertheless, these methods also tend to add some complexity, computational load or reliance on artificial data, which lowers their feasibility. Not the least, scalability and energy efficiency are seldom viewed as high-priority values, although they are the focus of SDN-enabled data centres and cloud services. These gaps need to be found and the performance of the current algorithms against them should be assessed in order to advance towards more robust and scalable solutions.

The work is valuable in that it synthesizes and critically evaluates various methods in SDN to load balance servers, with specific focus to the mechanisms employed, the performance measurements taken, and the experimentation employed. It aims to clarify researchers and practitioners

about the existing limitations and innovations in this field of rapid development by expressing the unresolved issues and outlining the emerging opportunities in the area.

II. SOFTWARE-DEFINED NETWORKING

SDN represents a networking paradigm that separates the control plane from the data plane, facilitating network evolution, interoperability, and scalability. This separation is the foundation of SDN and is made possible by decoupling switching components [1]. Software-defined networks were developed to achieve factory-independent control from a single logical point, facilitate innovation, and enable simple programmatic control of the network data path. Separating forwarding devices from control logic allows for easier deployment of new protocols and applications, clear network visualization and management, and the integration of various middle boxes into programmatic control. Instead of enforcing policies and implementing protocols on dispersed devices, the network is reduced to simple forwarding devices and a network controller (or controllers) that performs decision-making [2][37]. Notably, one of the most prominent examples of middlebox functions implemented using SDN is load balancers [3][4].

A. Software-Defined Networking Architecture

The SDN model consists of three layers: the infrastructure layer, the control layer, and the application layer, which stack on top of each other, as shown in Figure 1.

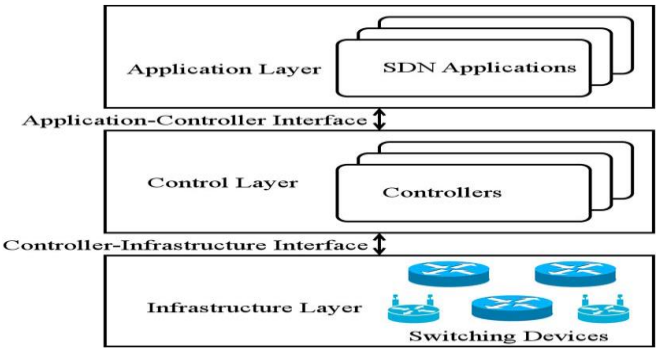

Fig. 1. SDN architecture [5]

- **The infrastructure layer**: Also called the data layer, it consists of switching devices (such as switches, routers, etc.). The functions of switching devices are: First, they are responsible for collecting network status, temporarily storing it on local devices, and sending it to controllers. Network status may include information such as network topology, traffic statistics, and network usage. Second, they are responsible for processing packets based on rules provided by the controller [5].
- **The control layer**: Represents the intelligence of the network. This approach enables the controller to retain a comprehensive awareness of the network' state within the infrastructure layer [1].
- **The Application Layer**: Contains SDN applications designed to meet user requirements. Through the programmability inherent in the control layer facilitates application-level access to, and governance of, the switching infrastructure [5].

The abstraction of the layers described above makes it easier for programmers to work on a single network layer, rather than thousands of different physical devices, through application programming interfaces (APIs) [1]. External management systems or network services may wish to extract information about the infrastructure or control an aspect of network behaviour or policy. This is done through the Northbound Application Interface (NAI) [2].

An important aspect of SDN is the connection between the data plane and the control plane, where forwarding elements are controlled by an open interface. It is important that this connection remains accessible and secure. The OpenFlow protocol can be viewed as one possible implementation of switch-controller interactions, as it defines the connection between switching devices and the network controller and is called the Southbound Interface protocol [2].

B. OpenFlow Protocol

OpenFlow is an open standard protocol specifically designed for SDN networks, allowing communication between the data and control planes,

as shown in Figure 2. OpenFlow switches and controllers can communicate using the OpenFlow protocol over a secure channel. The protocol defines various messages, such as receiving a packet, sending a packet, modifying the routing table, and obtaining statistics, which can be exchanged between the switch and controller [1].

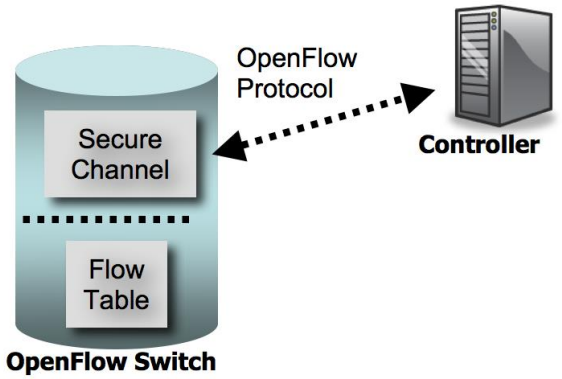

Fig. 2. Communication of an OpenFlow switch with a controller using the OpenFlow protocol [6]

The OpenFlow protocol was initially developed at Stanford University in 2008 to enable researchers to run experimental protocols on campus networks. OpenFlow is now being added as a feature to commercial networking devices, providing a standard, vendor-free interface for accessing Ethernet switches, routers, and wireless access points. Furthermore, these so-called "OpenFlow-enabled" devices allow access without requiring manufacturers to reveal the internal workings of their products. Therefore, the OpenFlow protocol makes it easy to deploy innovative routing and switching protocols [1].

C. Controllers

The controller acts as the main unit responsible for overseeing the key functions of the control plane. It creates updates and removes flow entries in the flow tables on the switch. In addition to its basic functionality, it can be extended to perform additional critical tasks such as routing, network access [6] and load balancing [3][4]. It can serve one or multiple switches,

depending on the network design. Currently, there are many controllers available, open source and based on various programming languages [6][38]. For example, POX and Ryu are based on Python, while Java is used in platforms such as Floodlight and OpenDaylight [7].

D. Load Balancing in Software Defined Networks

Load balancing is one of the fundamental pillars that contribute to improving the performance and efficiency of software-defined networks (SDNs). With the rapid expansion of Internet-based services, such as cloud computing, the Internet of Things, and time-sensitive applications, load balancing has become a critical component for ensuring network stability, optimizing resource utilization, and maintaining low response times while minimizing data loss. This fundamental importance of load balancing has prompted researchers to explore it in depth by developing multiple algorithms and methods aimed at improving the distribution of data traffic within networks**.**

In this context, one study [8] classified load balancing in SDNs based on the data level into two main categories:

[1] Server load balancers: These focus on distributing data traffic among available servers to improve their performance.
[2] Link load balancers: These distribute data traffic across different network links to ensure optimal bandwidth utilization and reduce congestion on network links.

In this article, we will adopt a comprehensive perspective on the studies that have addressed server load balancers as a key area. The reason behind this focus is the pivotal role servers play in processing data traffic and their direct impact on quality of service. Servers are considered the core component of modern networks, and distributing the load on them contributes to improved resource utilization and reduced overall system response time.

We will explain and analyze several methodologies, techniques, and algorithms used in research to achieve server load balancing. What distinguishes this study from others is that it provides a comprehensive view of each method by analyzing it from all possible dimensions. We have addressed each algorithm from all technical aspects, including: the

mechanism of action, how the algorithm was designed, the steps it follows to achieve load balancing, the topology used, the parameters measured, the tools used from network simulators and performance measurement tools, the algorithms compared with them, the future prospects, and the points mentioned by each study to improve performance or expand the use of the algorithm in future. The goal is to provide researchers with a deep understanding that will help them choose or improve the methods used in server load balancing within software-defined networks. This article will identify current challenges and future opportunities in this field, providing a starting point for future research.

Before delving into previous research studies, we will review the basic load balancing algorithms, which are often used as benchmarks in most research. We will also review the most used performance metrics for evaluating the efficiency of algorithms.

Basic Load Balancing Algorithms

- Random Algorithm

In this algorithm, the Load Balancer arbitrarily chooses one server node for forwarding the request from client nodes [9].

- Round Robin Algorithm

With this method, the load balancer distributes incoming requests across multiple servers using a round robin strategy [9].

- Weighted Round Robin Algorithm

The Weighted Round Robin maintains a cyclical request distribution similar to the traditional Round Robin, but allocates a larger portion of the requests to servers with higher specifications, based on predefined weights [9].

The Most Used Performance Metrics

- Response Time

It refers the duration required to handle and reply to incoming requests [10].

- Throughput

It is the amount of data transferred per second [11].

- Transaction Rate

It expresses the number of requests processed per second [11].

III. EXPLORING STRATEGIES FOR SERVER LOAD BALANCING IN SDN

In this article, we will conduct a comprehensive analysis and review of many scientific papers that address the topic of server load balancing in software-defined networks. This information will then be summarized in a clear and organized table that facilitates reviewing the main points of each study.

Hamed et al. [12] proposed a load balancing technique between servers in an SDN architecture that distributes traffic across available servers based on the least number of active TCP server connections (connection-based). The proposed algorithm was compared with round-robin and random in two environments: the first using Mininet with a topology of 3 hosts, one of which works as a client and the rest as web servers, with an OpenFlow switch and a Ryu controller as shown in Figure 3.

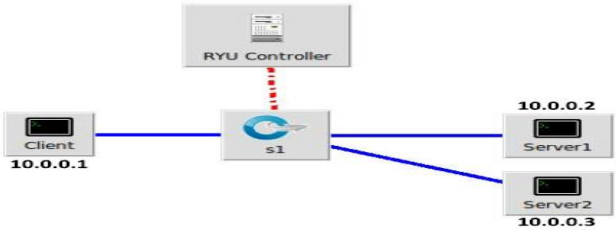

Fig. 3. First topology used in [12,13]

The second using a Raspberry Pi testbed with Open vSwitch installed to be used as an OpenFlow switch, with two laptops, one as a client and the other as a controller, running Ubuntu, respectively, with two Raspberry Pi kits running Linux as web servers, and which are all connected to an OpenFlow-enabled Raspberry Pi switch as shown in Figure 4.

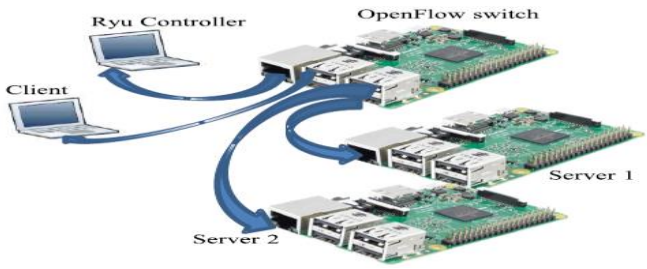

Fig. 4. Second topology used in [12,13]

To measure throughput, number of errors, and average response time, the HTTperf and Openload tools were used. The results and experiments showed that the proposed algorithm is better than the round-robin and random strategies in terms of the measured parameters.

Hamed et al. [13] aimed to develop a new strategy to load balancing using SDN. A bandwidth-based load balancing approach for web servers is proposed, which distributes network requests among a group of servers based on the servers' bandwidth consumption. The proposed BWBLB algorithm selects the server that consumes the least bandwidth during the last 14 seconds to handle requests. The proposed technique was compared with round-robin and connection-based load balancing algorithms in two environments: the first using a Mininet emulation environment with a topology consisting of an OpenFlow switch and a Ryu controller with three hosts, one of which acts as a client and the rest as web servers, as shown in Figure 3.

The second environment used a Raspberry Pi with Open Vswitch installed to turn it into a real-time OpenFlow switch with four hosts, one of which is used as a client generating HTTP requests, and the other is used to run the load balancing application (the Ryu controller), Both run Ubuntu, with two Raspberry Pi kits used as HTTP servers to respond to client requests. The servers, the client, and the controller are all connected to an OpenFlow-enabled Raspberry Pi switch, as shown in Figure 4.

To measure throughput and average response time, the HTTPerf and OpenLoad tools were used, and the results showed that the proposed load

balancing algorithm improves throughput and response time for web servers.

Sroya et al. [14] implemented the least delay dynamic weighted round robin (LDDWRR) load balancing algorithm in an SDN environment, where the link delay was taken into account as there is a link between each server and an OpenFlow switch and a different delay was assigned to each link. The dynamic weight was then determined according to the delay so that the server with the least delay handles more weight than the other servers with the highest delay and thus handles a larger number of requests. To test the LDDWRR algorithm, a Mininet topology was created consisting of 3 web servers, one client, an OpenFlow switch and a POX controller as shown in Figure 5.

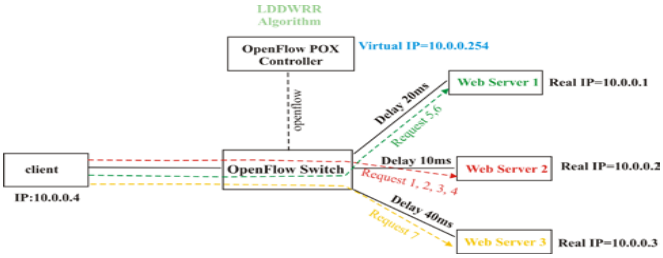

Fig. 5. Mininet topology used in [14]

For load testing, the siege tool was used and the results showed that the LDDWRR algorithm is better than the Round Robin algorithm based on transfer rate, throughput and response time.

Zhong et al. [15] designed a load balancing algorithm in an SDN environment, where the controller was used to obtain the response times of each server via Packet-Out messages, which were then to select the server demonstrating the shortest or most consistent response time. To evaluate the proposed scheme, a topology consisting of an Open Vswitch, a Floodlight controller, and three virtual machines acting as servers was used, while another virtual machine used 30 clients to access the servers. The proposed approach was compared with random, round robin, and least connection algorithms, and the results showed that the proposed algorithm (LBBSRT) yielded a better load balancing effect and processed requests with minimal average server response times. Memory usage and CPU

utilization rates were also extracted for each server, and a slight difference was observed in the memory and CPU utilization of the three servers in the proposed approach, i.e., the resources of all servers were utilized almost equally compared to other algorithms.

Li et al. [16] proposed a fuzzy logic-based load balancing strategy (LBSFL) for SDN networks. Initially, the correlation between several parameters affecting load balancing (processor utilization, memory utilization, and I/O utilization) is analyzed, and the load of multiple virtual servers is obtained using a fuzzy logic algorithm. Then, the load status of the virtual servers is examined in real time, and the lightest virtual server is selected to handle the request. If necessary, the server's sleep/restart policy is adjusted. To validate the proposed load balancing algorithm, a Mininet topology with an OpenFlow switch and an OpenDaylight controller with four web servers and a client is used, as shown in Figure 6.

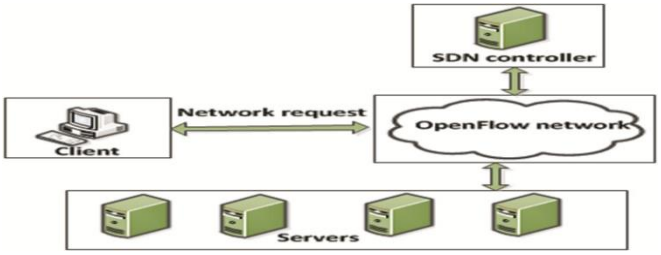

Fig. 6. Test network in [16]

The Iperf tool was used to test the network performance. The efficiency of LBSFL was compared with weighted round robin (WRR) and weighted least connections (WLC) algorithms. Experiments showed that LBSFL had the lowest average server response times, and that LBSFL had a better load balancing effect in terms of processor utilization, memory utilization, and I/O utilization than other algorithms.

Joshi et al. [17] compared different load balancing algorithms among a set of servers in a software-defined networking environment. The compared algorithms were the round-robin algorithm, the server load balancing algorithm, in which the server with the lowest CPU load is selected to handle the next request; and the flow statistics-based load

balancing algorithm, in which the server with the fewest flow connections is selected to handle the next request. These algorithms were also proposed in other studies. These algorithms were compared using a Mininet emulator with a topology consisting of a POX controller (as a load balancer), an OpenFlow switch, and a group of hosts, some acting as clients and others as servers. The results were tested using the following tools: Siege and load balancer sniper. The findings indicated that the server load balancing algorithm outperformed others in both throughput and transaction rate. Regarding response time, the flow statistics-based load balancing algorithm showed better response time compared to other algorithms.

Soleimanzadeh et al. [18] proposed a new load balancing mechanism for web server farms based on a centralized SDN controller that periodically collects information about switch port traffic and server response time and subsequently directs each incoming flow to the most suitable web server. The suggested algorithm is structured around three main modules: one that monitors sever response times and another that tracks traffic on switch ports, both operating at consistent intervals of two seconds. The third component is the best server locator, which selects the most suitable server based on the statistics obtained using the previous two components. To evaluate the performance, a Mininet emulator topology consisting of 30 client nodes, four server nodes, one Floodlight controller, and one switch was used. To generate HTTP requests, a custom application was written in .NET Core. The proposed SD-WLB approach was compared with random and round robin algorithms and the load balancing algorithm based on server response time (LBBSRT). The proposed strategy achieved higher throughput and lower response time compared to other load balancing methods. The processor and memory utilization rates for each server were extracted during the execution of each algorithm. It was found that there was a slight difference in memory and processor utilization for the four servers in the proposed approach. This means that the resources of all servers were used almost equally, and no server used its resources significantly different from the others, meaning that the

available resources were used almost equally. As future work for this paper, the following are proposed:

- Consider server load and status and incorporate these statistics into the workflow to make better decisions.
- Consider the time intervals for aggregated server response time and switch port traffic, as well as the weights α and β used in the equation to calculate the best server dynamically and adaptively.
- Leverage previously recorded metrics-such as server response times, swich port utilization, and various network parameters-as input datasets for machine learning algorithms.

Montazerolghaem et al. [19] presented a new architecture for SIP (Session Initiation Protocol) networks, where an SDN-based framework was introduced to perform load balancing between SIP servers, i.e., controlling SIP overload through an SDN approach instead of using traditional load balancers. The SIP is a signaling protocol for handling a variety of applications, including Voice over Internet Protocol (VoIP) and Instant Messaging (IM), for establishing and terminating calls. SIP architecture primarily involves two essential elements: user agents and servers. Within this protocol, user agents initiate connection requests directed to servers. However, due to the finite processing capacity of SIP servers, a high volume of user agents can potentially lead to server congestion. Therefore, the proposed framework was evaluated using three load balancing algorithms: random, round robin, and least request, with a Mininet topology consisting of a Floodlight controller, a switch, three servers, and a number of user agents as shown in Figure 7.

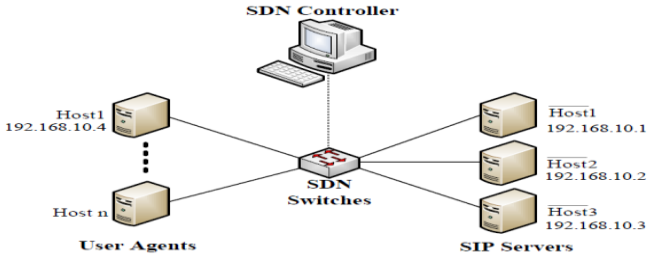

Fig.7. Topology used in [19]

Performance evaluation relied on two primary indicators: average response time and throughput. The results showed that the least request method has a lower average response time and higher throughput than random and round robin.

Aguilar et al. [20] evaluated how effective Software-Defined Networking (SDN) is in enhancing load balancing mechanisms. The proposed load balancer implemented three different load balancing algorithms: random, round robin, and least bandwidth. To evaluate the effectiveness of the load balancer, a Mininet topology consisting of a switch, a POX controller, a host acting as a client, and a varying number of servers was used. Load generation and response time measurement were conducted using Apache ab tool. Performance testing demonstrated that the server farm, when managed by the SDN-based load balancer, maintained response times under 0.1 seconds—well within acceptable user thresholds—highlighting the superior adaptability and effectiveness of SDN-driven load balancing over traditional hardware-based solutions. The experiments also showed that the least bandwidth method had a generally low response time compared to other algorithms.

Linn et al. [21] compared the Round Robin and Weighted Round Robin load balancing strategies in software-defined networks. A topology consisting of a POX controller, an OpenFlow switch, 16 clients, and three servers used as simple web servers was used using the Mininet emulator as shown in Figure 8.

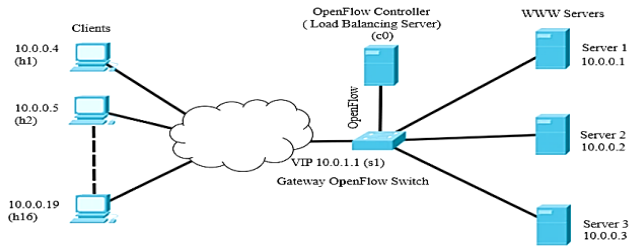

Fig. 8. Topology used in [21]

To measure the average server response time, the OpenLoad tool was used. Experiments showed that the average server response time for the Weighted Round Robin algorithm is smaller than that for the Round Robin algorithm.

Faraj et al. [22] applied the least packet load algorithm, which is used in traditional load balancers using a POX controller to distribute the load among servers in an SDN environment by directing client requests to the servers that are currently receiving the least number of packets. The proposed algorithm was compared with random, round robin, weighted round robin, least connection, and least bandwidth algorithms using a Mininet emulator with a three-server topology and a different number of clients similar to Figure 9.

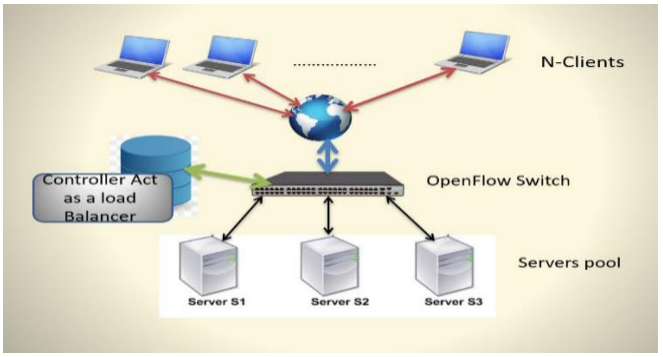

Fig. 9. Network topology used in [22]

The Iperf tool was used to generate TCP and UDP traffic from clients to servers and then calculate the throughput on each server. The results showed that the proposed algorithm outperforms all other algorithms in terms of throughput, especially when the network load is high.

Utomo et al. [23] implemented load balancing on a software-defined network using the Naïve Bayes algorithm. The Naïve Bayes algorithm was designed to obtain the server status based on data received from the Psutil agent and distribute traffic based on the current server status. After the algorithm receives input variables (processor usage, memory usage, disk usage) from the agent, computational operations are performed on them to convert them into a linguistic form based on a pre-defined dataset of 27

states to obtain the server status. After obtaining the server status, traffic is distributed based on the least resource usage of the server by the controller. To test the performance of the Naïve Bayes algorithm, a Mininet topology was used consisting of a POX controller, one switch, three hosts acting as servers, and one host acting as a client, as shown in Figure 10.

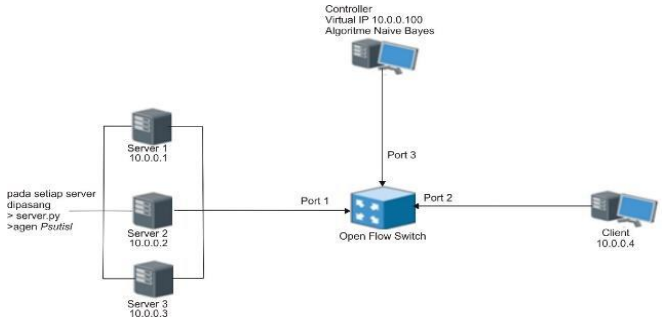

Fig.10. Network topology used in [23]

The test results showed that the Naïve Bayes algorithm performs better than the Fuzzy algorithm in terms of response time, throughput, memory usage, and CPU usage, as the Fuzzy algorithm contains more complex calculations compared to the Naïve Bayes algorithm.

Fancy et al. [24] introduced the TA-ASLB algorithm, a traffic-aware and adaptive approach designed to distribute incoming flows efficiently across servers within a data centre environment. The proposed load balancing operates based on flow size, checking the type of incoming flow and classifying it as a mice flow or an elephant flow. The algorithm is used only for elephant flows. The suggested approach is structured into three phases: monitoring network traffic, making adaptive decisions, and selecting the appropriate server. In the traffic monitoring stage, the controller monitors the entire topology. The proposed method takes into account three important details: the size of each flow, the remaining bandwidth on all links to the servers, and the computational capacity of all available servers. In the second stage, if the flow size is less than a threshold of 10 KB, a decision is made to forego the use of complex load balancing algorithms. Therefore, load balancing is carried out using the

round-robin technique. When the data flow exceeds a predefined threshold, the TA-ASLB mechanism is activated. In the final step, server selection is carried out using the TA-ASLB algorithm. This method evaluates two main factors: the bandwidth of the links leading to the server and the server's own processing capacity. The algorithm aims to identify nodes with the greatest remaining bandwidth capacity to ensure optimal data transmission. To achieve this, the remaining bandwidth of each link is estimated by analyzing its current traffic load. The server that offers both the greatest available bandwidth and the highest capacity is then selected. The algorithm was implemented using the Mininet emulator, creating a Fat-Tree topology containing seven switches and eight hosts, two of which act as clients and the rest as servers, with a Floodlight controller, as illustrated in Figure 11.

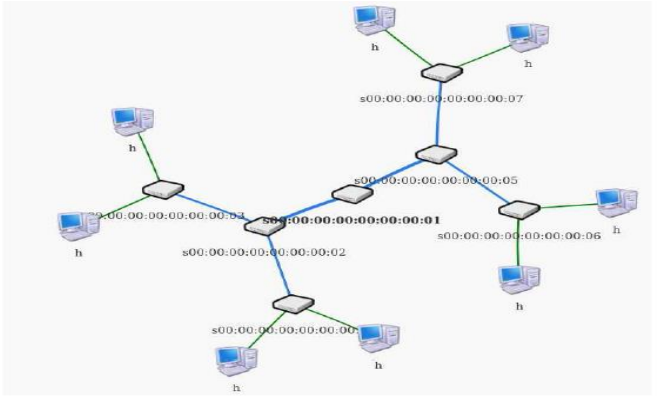

Fig. 11. Network topology used in [24]

To verify the effectiveness of the proposed algorithm, it was compared with round-robin, weighted round-robin, and statistics method based on the remaining bandwidth of the node in terms of throughput measured using the Iperf command and round-trip time measured using the Ping command. Compared to alternative algorithms, the proposed solution achieved higher data transfer rates and lower. The paper suggests future work on:

- Using neural networks in the server selection process, which can suggest optimal solutions, making it a better choice in the future.

- Dynamically determining the threshold based on the current network status, rather than using a static flow classification method.

It explained that this allows for an accurate model to be created that focuses on the current status of machines, such as the number of flows currently waiting to be processed, the number of flows already processed, server capacity, remaining bandwidth on the links connected to the servers, queue length, arrival rate, etc. Focusing on these issues improves the efficiency of current load balancing systems.

Krishna et al. [25] proposed a machine learning-based load balancing system. The architecture includes a Ryu controller, a selector unit, multiple clients, and a set of servers. The controller receives the IP address of the optimal server from the selector module and accordingly forwards the request to the optimal server in the network. The selector is where the trained machine learning model is integrated. It takes trip time, RAM usage, and CPU usage as inputs and outputs the best type of requests that the server can handle with those given statistics. The model was trained on a dataset containing 200 data points, where each data point is a set containing trip time, RAM usage, CPU usage, and a flag that is either 1 or 0 corresponding to the request type (media or text). However, it was not mentioned how the dataset was generated or created. The machine learning model employs logistic regression to train and predict the optimal request type to be handled based on the given features. Clients may request data in the form of media or plain text. The server runs two processes in parallel. One is responsible for processing the request received from the client by the controller, and the second process involves sending server information to the selector module, which uses this information as input to determine the best IP address for any type of request. The experimental network setup includes a controller and five hosts connected to a single switch, where three hosts function as clients and the remaining two serve as servers. As for the results, they only showed that their machine learning model demonstrated an accuracy of 88% in selecting the optimal server for a given type of request. Since the machine learning model, they used in their

paper took trip time, RAM usage, and CPU usage as inputs, the researchers suggest as future work for this paper is to improve the model by adding more features and considering a greater number of request types.

Sobih et al. [26] evaluated the performance of a set of load balancing algorithms in software-defined networks. These algorithms include the random algorithm, the Round Robin algorithm, the weighted Round Robin algorithm, the least-delay weighted dynamic Round Robin algorithm, the least-connection algorithm, and the weighted least-connection algorithm. Evaluation of the six algorithms focused on transaction rate and response time in a network setup where all servers shared identical configurations. Then, measurements of response time and transaction rate were conducted after modifying network conditions through the insertion of variable delays on the links connecting the switch to the server. These parameters were then measured by changing the server specifications, specifically their CPU speed. In the final scenario, the performance of the algorithms was evaluated with an additional load directed to one of the servers only. All of these scenarios were implemented by building a network consisting of a POX controller, an OpenFlow switch, three hosts acting as web servers, and several hosts acting as clients, as shown in the figure 12.

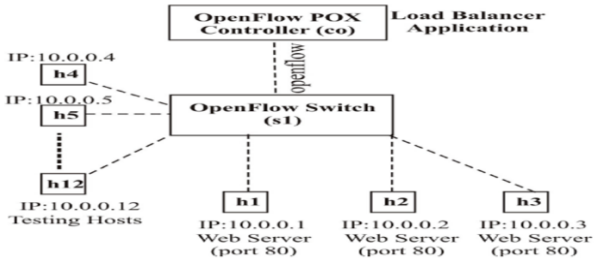

Fig. 12. The network used in [26]

To evaluate web server performance, the Httperf utility was employed to simulate HTTP traffic by generating request loads. Experiments have shown that the response time of static algorithms is less than the response time of dynamic algorithms when the number of requests in the network is small. However, when the network conditions are fixed and predetermined, the static algorithms (Round Robin, Least Delay, Weighted Round Robin,

Weighted Round Robin) that take these conditions into account can outperform the two dynamic algorithms. While the random algorithm was unable to provide good performance in any case when the number of requests per second is large, it provided relatively good performance when the number of requests per second is small (less than 60 requests in the applied scenarios). As for the two dynamic algorithms (least connections, weighted least connections), they can adapt to different network conditions and provide relatively good performance and can outperform the rest of the static algorithms under fluctuating network conditions.

Alssaheli et al. [27] analysed the performance of SDN in balancing the load among a group of servers using the POX controller, which pre-contains the IP_loadbalancer component, which is a simple load balancer that randomly directs new flows to the server addresses in three scenarios with different numbers of servers and hosts, as shown in Figures 13, 14, and 15.

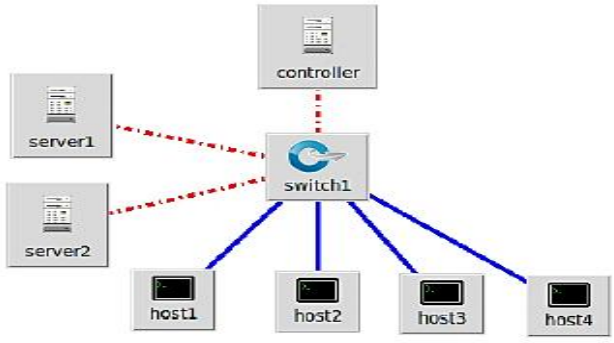

Fig.13. First scenario in [27]

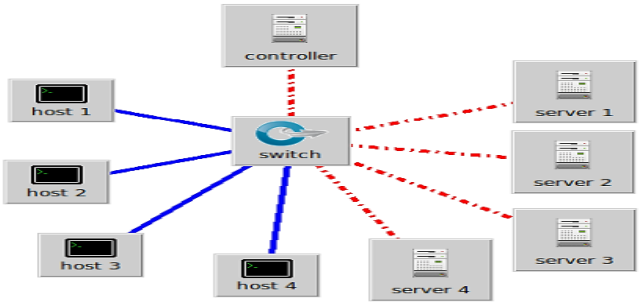

Fig. 14. Second scenario in [27]

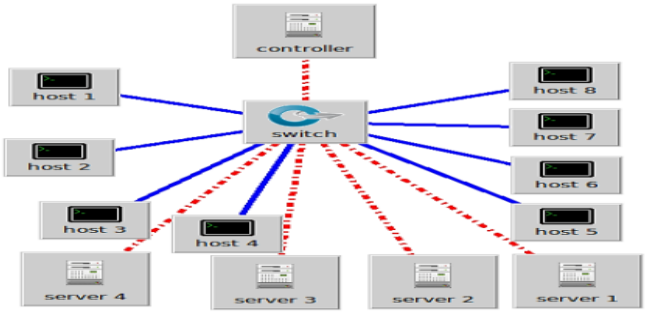

Fig. 15. Third scenario in [27]

These scenarios were implemented using the Mininet emulator, and in each scenario, throughput, delay, and Jitter were calculated. The findings indicated that employing SDN for load balancing leads to a better scheduling mechanism.

Vishwakarma et al. [28] compared different load balancing algorithms in an SDN based data center network such as round robin, random, weighted round robin, least connection and equal load using Mininet emulator with a topology of 4 legacy switches and an OpenFlow switch connected to POX with a group of hosts some of which are working as servers and others as clients as in Figure 16.

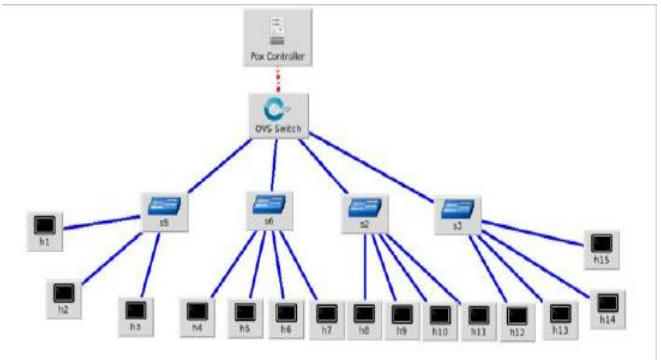

Fig. 16. Test topology used in [28]

The network's performance was evaluated through the deployment of the Siege testing utility. The test focused on the parameters of throughput, average response time and transaction rate. The results showed that the weighted round robin method outperforms all other presented algorithms. Given that average response time, transaction rate and throughput were used as performance metrics, the researchers proposed exploring additional parameters such as bandwidth usage and server utilization as part of future work.

Hamdan et al. [29] proposed to implement load balancing in SDN using linear regression. The dataset for modeling the linear regression function contains thirty samples. The constructed dataset includes key server metrics, specifically memory and CPU usage. The model uses processor and memory usage as input variables to forecast the number of requests for each server. The model is trained to associate resource consumption levels with the corresponding request load per server. The controller selects the server with the fewest requests to process the request. To read resource metrics from the server, this research employs the Psutil util. Server metrics are continuously transmitted to the controller, and the resource information collected via Psutil is stored in the dataset, where the last five samples in the dataset are updated every five seconds. For testing purposes, an SDN topology using a Mininet emulator was used. It consists of a POX controller acting as a load balancer using linear regression

calculations, another POX controller for forwarding, six switches, six clients, and two servers with varying specifications, as shown in Figure 17.

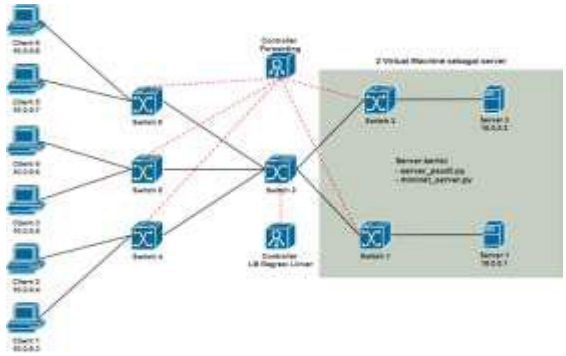

Fig. 17. SDN topology used in [29]

The HTTPerf tool was run on one of the clients. Testing was performed in scenarios with rates of 100, 150, 200, 250, and 300 requests per second. Each scenario was executed three times, with a total of 5,000, 7,500, and 10,000 requests. This research compared linear regression with a round robin algorithm, and the results showed that linear regression and round robin performed relatively similarly in response time testing when the request rate was low. As the request rate increased, linear regression outperformed round robin in response time performance. In throughput testing, linear regression and round robin have a similar pattern as the response time test. When the rate used is small, linear regression and round robin have relatively similar performance. However, linear regression outperforms round robin when the rate used is higher. In the CPU utilization test, linear regression performs better than round robin. During the linear regression analysis, CPU usage peaked at 75 percent. Meanwhile, round robin reached 98 percent. As future work for this paper, the following are proposed:

- Increase the accuracy of linear regression calculations by adding more samples to the dataset.
- Improving the program in terms of the time when linear Regression calculations are performed so that they are computed only if there is a change in the server's information.

Malavika et al. [30] focused on extending SD-WLB and LBBSRT by implementing the closed-loop control theory (CLCT) (this theory means measuring the controlled variable and comparing it to a setpoint that represents the desired performance [35]) with two threshold values L and U to adaptively adjust the timing of server probe. Both SD-WLB and LBBSRT probe servers in the server farm at a uniform and specific time interval to retrieve the factors of server response time and switch port traffic volume. Therefore, the task of the proposed framework LBBNNC was how the controller recovers these factors at variable time intervals by defining the application delay tolerance as threshold values L and U to distinguish the server load status as low, medium, and high based on the maximum response time difference:

MRTD = MAX (response time of all the servers)-MIN (response time of all the servers)

between servers in a server farm that were less than L, between L and U, and higher than U to adaptively adjust the probe time to measure the server response and switch port traffic volume. The values of L and U were adjusted using a neural network then feeds the L and U values into a probe time generation algorithm, which, based on the L, U, and MRTD values, determines the new probe time for the servers in the next iteration. When traffic increases, the probe time is reduced, and when traffic decreases, the probe time is increased. To evaluate the performance of the proposed LBBNNC framework, a Mininet topology with one switch and one controller was used, with four hosts acting as clients and three hosts acting as Apache web servers. For testing purposes, the Httperf tool was used, using 50,000 requests at a request rate of 10 requests per second to 100 requests per second, increasing the request rate by 5 requests per second each time. The proposed scheme proved to be superior to round robin, LBBSRT, and SD-WLB in terms of the achieved request rate, average response rate, average response time, and number of produced errors.

Malbašić et al. [31] proposed a new load balancing scheme for a hybrid SDN environment. The scheme relied on continuous monitoring of server

load indicators and implementing multi-parameter metrics (CPU load, I/O read, I/O write, link upload, link download) to schedule connections. This scheme used the SNMP protocol mechanism to collect information about the resources used by the server to schedule requests. A simulated environment incorporating virtual servers and network components using EVE-NG platform [36] to facilitate testing and experimentation. The following virtual components were implemented: a Linux OVS router as an SDN-aware switch; six virtual machines with Linux Ubuntu servers (two representing web servers, three representing clients, and one representing a POX controller); and the last component represents a Mikrotik ROS router plus an internet connection, as shown in Figure 18.

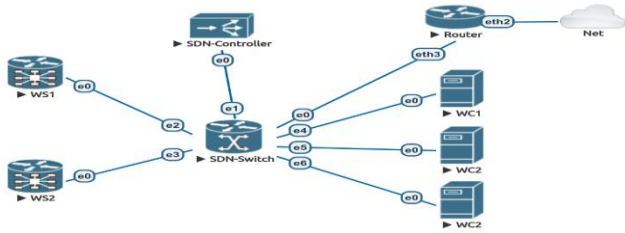

Fig. 18. Testing environment used in [31]

The proposed scheme was compared with Random, Round Robin, and LBBSRT algorithms, and the test results showed that the proposed mechanism yielded the best results in terms of balancing CPU load on servers and made better load balancing decisions than the tested mechanisms.

Singh et al. [32] implemented a Round Robin load balancing algorithm in SDN to overcome the limitations of traditional network load balancers using a Mininet topology consisting of a POX controller, an OpenFlow switch, six hosts acting as web servers, and several hosts acting as clients. Siege and Autobench were used for performance testing. The experiments showed 100% availability of services with an average of 0.028 seconds in the shortest transfer and a response time of less than 1 second.

Vani et al. [33] presented a dynamic load balancing algorithm for software-defined networks based on server response time and content

mapping. When performing load balancing among a group of servers, the controller directs requests to the respective servers in the server pool based on the load balancer result for the requested content type and response time. Server pools are organized based on content type, including dedicated pools for video, audio, images, and textual data. The proposed architecture consists of three main modules: request classification module, server monitoring module, and dynamic load balancing module. The request classification module is designed to determine the nature of incoming requests, categorizing them into formats like video, audio, image, or text. Once this information is extracted, it is sent to the load balancing module. To ensure real-time oversight, the server monitoring module is designed so that each server within the pool transmits its load metrics to the controller at 5-milliseconds intervals. Information such as CPU usage, memory usage, requests per second, time per request, transfer rate, and waiting time are sent to the controller. The dynamic load balancing module is implemented using content-based routing. When a client request is received, the content is analysed and the server with the lowest response time in each server cluster is found. The controller installs the flow based on the requested content and the server with the lowest response time. For example, if the request is related to images, it is redirected to the image server pool, etc. The experiment was conducted in a data center network where a number of different clients and web servers, such as Apach2, Ngnix, and simpleHTTPserver, were connected to the Ryu controller via a Zodiac-FX switch in real time, as shown in Figure 19.

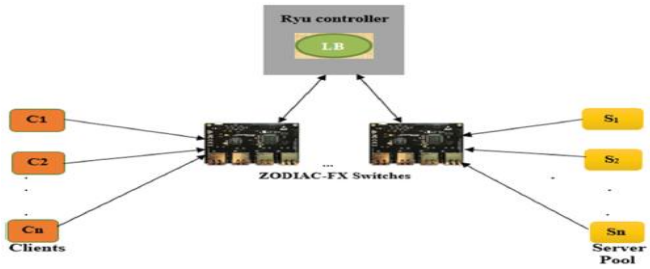

Fig. 19. Data center network used in [33]

The proposed mechanism was compared with different techniques such as round robin, random, LBBSRT (load-balancing scheme based on server response time) and SD-WLB (Web load balancing based on server statistics) based on various metrics such as average response time, transfer rate, time per request, requests per second, and waiting time. The proposed mechanism has shown better performance compared to other techniques.

Fazal et al. [34] proposed an adaptive approach to load balancing between servers for SDN-based data centre networks by identifying an optimal server and choosing the best path to the selected server. To do this, the proposed approach is divided into five modules: the first module is the probe interval generator, which determines the time interval after which servers will be checked for their performance and is inspired by the LBBCLCT (Load Balancing Based on Closed Loop Control Theory) approach. The second module is the server performance monitor, which monitors server response times and stores them in a local database, and is inspired by the LBBSRT approach. The third module is the server selector, which reads the local database of server response times and selects the server with the lowest response time as the optimal server. The fourth module is the switch port aggregator, which checks switches for the traffic volume passing through its ports and stores the data in a local database and is originally inspired by the SD-WLB approach. The fifth module, known as the path selector, will take the specified server and traffic data from the database and apply the ant colony algorithm on the network to determine the most efficient route between the sender and the most suitable server. The proposed approach was tested in a Mininet environment with a fat-tree topology, with 31 switches and 125 hosts, 12 of which were configured as servers, and the rest as hosts with a Floodlight controller, as shown in Figure 20.

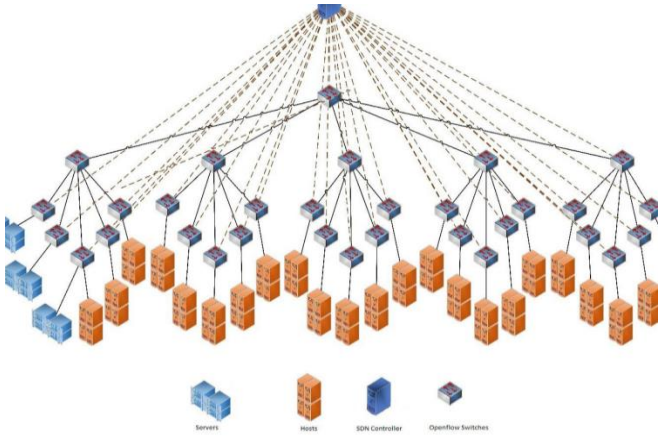

Fig. 20. Fat tree topology used in [34]

To verify the effectiveness of the proposed algorithm (SD-ALB), it was compared with round-robin, SD-WLB, LBBSRT, and LBCLCT algorithms. The results showed that the proposed approach outperforms in terms of throughput and server response time, but it requires a lot of computational resources.

After reviewing the articles and analyzing their methodologies, the Table 1 summarizes their key aspects to facilitate comparison and identify important research trends:

TABLE 1 SUMMARY OF PREVIOUS RESEARCH STUDIES ON SERVER LOAD BALANCING IN SOFTWARE DEFINED NETWORKS (SDN)

Reference	Year	Objective	Controller Used	Network Topology	Algorithms Compared With	Tools Used	Measured Parameters	Future Prospects
[12] Hamed et al.	2017	Proposing a technique for load	Ryu	Switch, controller, 3 hosts (1	Random, Round Robin	Mini net, Rasp berry	Through put, Respons e time,	–

		balancing between servers in an SDN architecture based on the least number of active server TCP connections		client, 2 servers)		Pi, Httperf, Open Load	Number of errors	
[13] Hamed et al.	2017	Bandwidth-based load balancing (BWBLB) approach for web servers	Ryu	Switch, controller, 3 hosts (1 client, 2 servers)	Round Robin, Connection-based	Mininet, Raspberry Pi, Httperf, Open Load	Throughput, Response time	–
[14] Sroya et al.	2017	Implementing Least Delay Dynamic Weighted Round Robin (LDDWRR) algorithm in SDN	POX	3 web servers, 1 client, OpenFlow switch, controller	Round Robin	Mininet, Siege	Response time, Throughput, Transaction rate	Using multiple controllers
[15] Zhong et al.	2018	Load balancing based on server response time (LBBSRT)	Floodlight	Open vSwitch, controller, 3 servers, 30 clients	Random, Round Robin, Least Connection	–	Response time, CPU & Memory utilization	Energy saving
[16] Li et al.	2018	Fuzzy logic-based load	OpenDaylight	OpenFlow switch, controller,	WRR, WLC	Mininet, Iperf	Response time, CPU,	–

		balancing (LBSFL)		4 servers, 1 client			Memory, I/O utilization	
[17] Joshi et al.	2019	Comparison of algorithms (Round Robin, Server Load Balancing, Flow statistics-based)	POX	Switch, controller, hosts as clients/servers	Round Robin, Server load, Flow-based	Mininet, Siege, Sniper	Throughput, Response time, Transaction rate	Suggest Ryu/multi-controller
[18] Soleimanzadeh et al.	2019	SD-WLB: centralized mechanism using server stats and switch port traffic	Floodlight	30 clients, 4 servers, 1 controller, 1 switch	Random, Round Robin, LBBSRT	Mininet, Custom .NET Core app	Throughput, Response time, CPU, Memory utilization	AI integration, adaptive intervals
[19] Montazerolghaem et al.	2019	SDN-based framework for SIP server load balancing	Floodlight	Controller, switch, 3 servers, user agents	Random, Round Robin, Least Request	Mininet	Response time, Throughput	Practical implementation
[20] Aguilar et al.	2019	Effectiveness of SDN in load balancing	POX	Switch, controller, 1 client, servers	Random, Round Robin, Least Bandwidth	Mininet, Apache ab	Response time	–
[21] Linn et al.	2019	Comparison of Round Robin and Weighted	POX	Controller, OpenFlow switch,	Round Robin, WRR	Mininet, OpenLoad	Response time	–

		Round Robin		16 clients, 3 servers				
[22] Faraj et al.	2020	Least packet load algorithm in SDN	POX	Controller, 3 servers, varying clients	Random, RR, WRR, Least Conn., Least BW	Mininet, Iperf	Throughput	Multi-controller, DDoS prevention
[23] Utomo et al.	2020	Naïve Bayes algorithm for server status-based balancing	POX	Controller, 1 switch, 3 servers, 1 client	Fuzzy	Mininet, Psutil	Response time, Throughput, CPU, Memory	–
[24] Fancy et al.	2021	TA-ASLB: traffic-aware adaptive load balancing	Floodlight	Fat-tree, 7 switches, 8 hosts (2 clients, 6 servers)	RR, WRR, Bandwidth-based	Mininet, Ping, Iperf	Throughput, RTT	Neural networks, adaptive thresholds
[25] Krishna et al.	2021	Machine learning-based load balancing	Ryu	Controller, switch, 5 hosts (3 clients, 2 servers)	–	Mininet	Accuracy 88%	More features, request types
[26] Sobih et al.	2021	Evaluation of six load balancing algorithms	POX	Controller, OpenFlow switch, 3 servers, clients	Random, RR, WRR, LDDWRR, LC, WLC	Mininet, Httperf	Transaction rate, Response time	–
[27] Alssaheli et al.	2022	Performance of SDN using POX IP_loadbalancer	POX	Scenarios with varying servers & hosts	–	Mininet	Throughput, RTT, Jitter	AI-based algorithms

[28] Vishwakarma et al.	2022	Comparison of algorithms in SDN data centers	POX	4 legacy switches + OpenFlow switch, clients & servers	RR, Random, WRR, LC, Equal Load	Mininet, Siege	Response time, Transaction rate, Throughput	Include bandwidth & server utilization
[29] Hamdan et al.	2022	Linear regression for load balancing	POX	2 controllers, 6 switches, 6 clients, 2 servers	RR	Mininet, Psutil, Httperf	Response time, Throughput, CPU utilization	Larger dataset, smarter execution timing
[30] Malavika et al.	2022	Closed-loop control theory (LBBNNC) for adaptive probing	Ryu	Switch, controller, 4 clients, 3 servers	RR, SD-WLB, LBBSRT	Mininet, Httperf	Response rate, Response time, Errors, Request rate	Cloud computing applications
[31] Malbašić et al.	2022	Hybrid SDN with multi-parameter metrics	POX	Linux OVS router, 6 VMs (2 servers, 3 clients, 1 controller) + Mikrotik router	Random, RR, LBBSRT	EVE-NG	Transactions/sec, Avg transaction time, CPU balance	New weight distribution models
[32] Singh et al.	2022	Round Robin implementation in SDN	POX	Controller, OpenFlow switch, 6 servers, clients	–	Mininet, Siege, Autobench	Response time, Transfer rate	New LB technologies

[33] Vani et al.	2023	Dynamic LB based on server response & content mapping	Ryu	Zodiac-FX switch, clients, servers, controller	RR, Random, LBBSRT, SD-WLB	–	Response time, Transfer rate, Requests/sec, Waiting time	–
[34] Fazal et al.	2023	Adaptive load balancing (SD-ALB) with optimal path selection	Floodlight	Fat-tree, 31 switches, 125 hosts (12 servers)	RR, SD-WLB, LBBSRT, LBCLCT	Mininet, Iperf	Throughput, Response time	Deep reinforcement learning

IV. CRITICAL ANALYSIS OF PREVIOUS STUDIES: RESEARCH LIMITATIONS IN SERVER LOAD BALANCING IN AN SDN ENVIRONMENT

Many studies have employed artificial intelligence techniques in the server selection process within SDN networks, with the aim of improving load distribution and increasing performance efficiency. However, the lack of a benchmark dataset that simulates realistic environments poses a significant challenge. This lack has prompted many researchers to generate synthetic datasets specifically designed to meet the study's requirements, which may limit the generalizability of results outside of experimental settings.

It is also noted that the proposed algorithms are often compared to the underlying algorithms. This is due to the unavailability of source code for most algorithms, which hinders comprehensive evaluation within the same test environment.

From a design perspective, most studies rely on a simplified topology that includes a single OpenFlow switch connected to servers, clients, and a controller. Despite the simplicity of this model, it is considered functionally appropriate because the switch directly connected to the servers is responsible for receiving flow balancing rules from the controller

and distributing requests based on them. Furthermore, some modern high-performance switches support connecting more than 20 servers, enhancing the realism of the proposed model without the need for more complex topologies.

Although there has been growing interest in load balancing concepts within the SDN environment in recent studies for 2024 and 2025, most of this research focuses on improving quality of service or managing routing without directly addressing the server load balancing problem itself. This highlights a clear research gap and indicates the need for more specialized studies on this vital aspect, especially with the increasing reliance of cloud and virtual infrastructure on SDN servers.

V. CONCLUSION AND FUTURE RECOMMENDATIONS

In conclusion, server load balancing in software-defined networks becomes one of the essential elements to ensure stability, scalability, and efficiency in the present infrastructures. This work makes a contribution by reviewing and analyzing a broad variety of algorithms, showing the advantages of dynamic and intelligent approaches compared to more traditional approaches of static approach, and the weaknesses, including the limited scalability of the approach, use of simplified experimental topologies, and lack of standardized benchmarking practices.

In practical terms, the summary of existing research offers useful information to network engineers, cloud data center operators, and SDN developers, who have to choose or create load balancing solutions depending on the performance requirements, in terms of response time, throughput and resource usage. The review, however, is limited by the fact that the studies reviewed are heterogeneous and they tend to use various testbeds, tools, and metrics, and such direct comparison with the results is not easily possible.

Going forward, the focus of future studies must be put on development of standardized evaluation frameworks, empirical benchmarking data, and adapting algorithms that would be capable of dealing with scalability and energy efficiency in addition to performance. Synthesis of artificial intelligence and machine learning into SDN controllers are also a

promising avenue particularly in an environment with dynamic and unpredictable traffic. To make sure that SDN-based systems are strong and sustainable as the demands on the global networks increase, the multi-controller architecture and energy-conscious designs deserve further attention as well.

REFERENCES

[1]. F. ONGARO, "Enhancing Quality of Service in Software-Defined Networks", UNIVERSITY OF BOLOGNA, 2014.

[2]. B.N. Astuto, M. Mendonça, X.N. Nguyen, K. Obraczka, T. Turletti," A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks", Communications Surveys and Tutorials, IEEE Communications Society, 2014.

[3]. R. Wang, D. Butnariu, J. Rexford, " OpenFlow-Based Server Load Balancing Gone Wild", Princeton University, 2011.

[4]. N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, R. Johari, "Plug-n-serve: Load-balancing web traffic using OpenFlow". ACM SIGCOMM Demo, Barcelona, Spain 2009.

[5]. W. Xia, Y. Wen, C.H. Foh, D. Niyato, H. Xie, "A Survey on Software-Defined Networking", IEEE COMMUNICATION SURVEYS & TUTORIALS, 2015.

[6]. V. Khatri, " Analysis of Openflow Protocol in Local Area Networks", M.S. thesis, Tampere Univ. of Technology, 2013.

[7]. G. Altangerel, T. Chuluuntsetseg, D. Yamkhin, "Performance analysis of SDN controllers: POX, Floodlight and OpenDaylight", the 13th International Forum on Strategic Technology, 2018.

[8]. M.C. Saxena, M. Sabharwal, P. Bajaj, "Review of SDN-based load-balancing methods, issues, challenges, and roadmap", International Journal of Electrical and Computer Engineering Systems, 2023.

[9]. S. K. Vishwakarma, H. Pavithra, " Performance Analysis of different Load Balancing Algorithms in SDN Based Data Center Networks", International Journal of Engineering Research & Technology, 2022.

[10]. Httperf, "httperf", https://github.com/httperf/httperf?tab=readme-ov-file

[11]. Btfak, "sniper", https://github.com/btfak/sniper

[12]. M.I. Hamed, B.M. ElHalawany, M.M. Fouda, A.S. Tag Eldien, " Performance Analysis of Applying Load Balancing Strategies on

Different SDN Environments", Benha Journal of Applied Sciences, Egypt, 2017.

[13]. M.I. Hamed, B.M. ElHalawany, M.M. Fouda, A.S. Tag Eldien, " A new approach for server-based load balancing using software-defined Networking", The 8th IEEE International Conference on Intelligent Computing and Information Systems, 2017.

[14]. M. S. Sroya, V. Singh, " LDDWRR: Least Delay Dynamic Weighted Round-Robin Load Balancing in Software Defined Networking", International Journal of Advanced Research in Computer Science, 2017.

[15]. H. Zhong, Y. Fang, J. Cui, "Reprint of LBBSRT: An efficient SDN load balancing scheme based on server response time", Future Generation Computer Systems, 2018.

[16]. G. Li, T. Gao, Z. Zhang, Y. Chen, " Fuzzy Logic Load-Balancing Strategy Based on Software-Defined Networking", Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering (LNICST) 2018, Springer Nature Switzerland AG, as part of WiCON 2017.

[17]. N. Joshi, D. Gupta, " A Comparative Study on Load Balancing Algorithms in Software Defined Networking", Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering (LNICST), Springer Nature Switzerland AG, 2019.

[18]. K. Soleimanzadeh, M. Ahmadi, M. Nassiri, "SD-WLB: An SDN-Aided Mechanism for Web Load Balancing Based on Server Statistics", ETRI Journal, 2019.

[19]. A. Montazerolghaem, "SIP Server Load Balancing Based on SDN", Quchan University of Technology, Iran, 2019.

[20]. L. C. F. P. Aguilar, D. M. Batista, " Effectiveness of Implementing Load Balancing via SDN ", Brazilian Symposium on Computer Networks and Distributed Systems, 2019.

[21]. A. S. Linn, S. H. Win, S. T. Win, "Server Load Balancing in Software Defined Networking", National Journal of Parallel and Soft Computing, 2019.

[22]. M. K. Faraj, A. Al-Saadi, R. J. Al-Bahadili, " An Investigation of Using Traffic Load in SDN Based Load Balancing", Iraqi Journal of Computer Communication Control and System Engineering,2020.

[23]. A. A. Utomo, P. H. Trisnawan, R. Primananda, "Server Load Balancing Mechanism Using Naive Bayes Method with Psutils Agent on Software Defined Network", Journal of Information Technology and Computer Science Development, 2020.

[24]. C. Fancy, M. Pushpalatha, "Traffic-aware adaptive server load balancing for software defined networks", International Journal of Electrical and Computer Engineering Systems, 2021.

[25]. P. M. Krishna, S. S. Rao, S. S. Bhurat, T. Venkatesh, K. A. Vani, " Machine Learning to Balance Web Applications using SDN", International Journal of Innovative Research in Engineering & Multidisciplinary Physical Sciences, 2021.

[26]. M. Sobih, I. Khalil, "Performance evaluation of controllers in software defined networks", Tartous University Journal for Research and Scientific Studies, 2021.

[27]. O. M. A. Alssaheli, Z. Z. Abidin, N. A. Zakaria, Z. A. Abas, " Software Defined Network based Load Balancing for Network Performance Evaluation", International Journal of Advanced Computer Science and Applications, 2022.

[28]. S. K. Vishwakarma, H. Pavithra, " Performance Analysis of different Load Balancing Algorithms in SDN Based Data Center Networks", International Journal of Engineering Research & Technology, 2022.

[29]. A. A. Hamdan, P. H. Trisnawan, F. A. Bakhtiar, "Implementation of Load Balancing using Linear Regression Method on Software Defined Network", Journal of Information Technology and Computer Science Development, 2022.

[30]. R. Malavika, M. L. Valarmathi, "Adaptive Server Load Balancing in SDN Using PID Neural Network Controller", Computer Systems Science & Engineering, 2022.

[31]. T. Malbašić, P. D. Bojović, ˇZ. Bojović, J. Šuh, D. Vujošević, "Hybrid SDN Networks: A multi-parameter server load balancing scheme", Journal of Network and Systems Management, 2022.

[32]. I. T. Singh, T. R. Singh, "Server Load Balancing with Round Robin Technique in SDN", International Conference on Decision Aid Sciences and Applications (DASA), 2022.

[33]. K. A. Vani, K. N. RamaMohanBabu, " An Intelligent Server load balancing based on Multi-criteria decision-making in SDN", International Journal of Electrical and Computer Engineering Systems, 2023.

[34]. R. Fazal, S. M. M. Gilani, M. J. Khalid, " SD-ALB: Software Defined Adaptive load balancing in Data Center Network", KIET Journal of Computing & Information Sciences, 2023.

[35]. ScienceDirect, "Closed Loop Control", https://www.sciencedirect.com/topics/engineering/closed-loop-control.

[36]. EVE-NG, "Emulated Virtual Environment Next Generation", https://www.eve-ng.net.

[37]. Y. Ghazi, A. Mustafa, "Improving Single-Controller Software-Defined Networks Architecture Using Named Data Networks Architecture", Homs University Journal, 2025.

[38]. M. Abboud, M. Aldaher, " Performance Analysis in software defined networks (SDNs)", Homs University Journal, 2019.