Design and Experimental Evaluation of Adaptive Load Balancing Strategies in Software-Defined Networks Using Mininet

M Shona, Rinki Sharma
Computer Science and Engineering, M S Ramaiah University of Applied Sciences, Bengaluru, India

***Abstract*—Software-Defined Networking (SDN) introduces centralized control and programmability, enabling more flexible and efficient network management compared to traditional architectures. Load balancing is a serious SDN application that improves resource utilization, reduces latency, and enhances service reliability. However, implementing SDN-based load balancers involves several challenges, such as controller overhead, scalability issues, dynamic traffic handling, and protocol integration. This study investigates these challenges and presents practical approaches for implementing SDN load balancers using the Mininet emulation environment. Different load-balancing algorithms are implemented and evaluated, highlighting the trade-offs between static and dynamic techniques. The study also examines traffic generation tools supported by Mininet. Furthermore, the performance of various SDN controllers, including Ryu, POX, OpenDaylight (ODL), ONOS, and Floodlight, is assessed using metrics such as throughput and round-trip time. Key performance evaluation metrics and their computation methods are also discussed. The goal of this research is to examine the challenges of implementing load balancing in Software-Defined Networking. It also aims to explore effective methods for designing and evaluating SDN-based load-balancing solutions using a Mininet test environment.**

***Keywords*—Load balancing; control plane; data plane; OpenFlow; Mininet**

I. Introduction

In recent years, the widespread use of the Internet has surged, leading to the need for greater bandwidth, improved reliability, and reduced latency. These demands have accelerated the development of advanced networking technologies. Software-Defined Networking (SDN) has emerged as a modern network architecture designed to intelligently manage network resources and dynamically adapt to changing requirements. Traditional networks are hardware-based, whereas SDN networks are software-based. Due to this architecture, it has the flexibility to directly program the network and has low operating costs. By using software for core functions such as routing, load balancing, resource allocation, and security, SDN provides centralized control over the network. Unlike traditional network architectures, where the control and data planes are closely integrated, and each switch makes routing decisions independently, SDN introduces a clear separation between these layers. The data plane, with SDN-enabled switches, handles packet transmission, while a centralized controller in the control plane oversees network management and traffic routing. Although the SDN is one of the virtuous solutions for information technology-based applications, it faces many challenging issues like security, scalability, controller placement problem, load balancing, interoperability, and reliability. The primary challenge in network management is achieving an even distribution of workload across multiple servers. When traffic is not evenly spread, it can significantly impact overall network performance. Load balancing addresses this issue by distributing network traffic efficiently among several servers. This study presents a comprehensive review of different load balancing techniques, detailing their implementation and performance evaluation within the Mininet environment, offering useful insights for researchers in related domains.

This study is organized as follows: Section II reviews the existing research on SDN load balancing methods. Section III outlines the types of SDN load balancing algorithms. Section IV highlights the overview of simulators and emulators commonly used in SDN research. Section V presents the features of different SDN controllers. Section VI introduces traffic generators compatible with Mininet. Section VII explains the key performance metrics used for evaluating SDN load balancers. Section VIII analyses the performance of static and dynamic load balancers. Section IX explores potential future improvements. Finally, Section X concludes the study.

II. Related Work

Load balancing typically serves the purpose of distributing network traffics across multiple servers or paths to optimize performance. The basic load balancing techniques such as round-robin and least-connection in a Mininet environment using the POX controller are implemented and evaluated in [1]. In [2], a Round Robin load balancing approach was implemented using a single Floodlight controller. A notable study [3] provides a comparative analysis of various SDN-based load balancing techniques, using Mininet. The authors emphasize the controller's role in state monitoring and highlighted the algorithm complexity and its latency. The impact of load balancers on SDN within cloud environments is explored in [4]. The author analyzed the effect of the application of SDN for load balancing to improve network performance [5]. In [6], the authors integrated Dijkstra's algorithm with Round Robin, achieving enhanced performance in fat-tree topologies. A Mininet-based setup with the POX controller was presented in [7], where load allocation was dynamically regulated based on server availability and health status. The work in [8] enabled SDN controllers to reroute

traffic automatically, ensuring sustained performance without manual intervention. In [15], the Least Connection algorithm was proposed, assigning traffic to the server with the fewest active connections and demonstrating better efficiency compared to Round Robin. The throughput, response time, and availability have improved through prioritizing the traffic flows in [10]. In [11], pox controller is used, which is less capable of managing traffic but achieves higher throughput and improved response times. In CAMD [12], the switch selected for migration is determined using a migration indexing factor, which results in improved response time, reduced migration cost, and higher throughput. Balcon [13] introduces a mechanism that balances the received requests among SDN controllers through optimal switch migration, thereby maintaining balanced performance under high traffic scenarios. A reinforcement learning-based switch migration method is proposed in [14], where the most heavily loaded controller is identified and one of its switches, chosen through a reward mechanism is reassigned to a different controller. The Weighted Round Robin approach [15] distributes requests using predefined static weights based on server capacity, which may not suit real-time traffic conditions. The Least Time-Based Weighted Round-Robin method [16] statically assigns weights to the servers based on the real-time link delay. The DWRS (Dynamic Weighted Random Selection) algorithm [17] serves as a dynamic load balancing technique designed for environments with a single SDN controller. Each server is allocated a weight based on its current real-time load. Consequently, the server with the lowest load is more likely to handle incoming client requests, resulting in improved throughput. Multi-path Load Balancing [18] is a link-based load balancer that shows improvement in response time and transfer rate. In [19], an Artificial Neural Network (ANN) technique is trained to identify the path that has the lowest load. This approach enhances network efficiency by considering factors such as latency and bandwidth. In [20], the load is divided into the network paths and aims at cost of route and cost of services. In another link based load balancer [21], a flexible and smart network management framework is designed to enhance dynamic routing efficiency. A dynamic LB algorithm [22] maintains a table of link costs and it is dynamically updated periodically. Jitter, packet loss and throughput are enhanced in this algorithm. Late Acceptance Neighbor Search [23] is a control plane based load balancer in which the controllers are assigned to switches by considering both the switch load and controller capacity, leading to reduced migrations. In a Distributed controller LB [24], each controller maintains a cost-state database that contains information about the travel costs between switches and their neighboring nodes. Based on the controller load and the cost matrix, the switches are assigned to a controller. It provides better flow set-up time.

Most of the existing works on load balancing in SDN have been focused on proposing various techniques or evaluating the controller for a specific scenario. However, there is a lack of holistic studies that address various implementation hurdles, various static, as well as dynamic load balancing techniques, and the evaluation of various SDN controllers in a cohesive environment. In addition, the implementation hurdles such as controller overhead, scalability of the controller for various traffic scenarios, and the integration of traffic generators in the test environment have not been given much importance in the existing works on SDN-based load balancing techniques. In this regard, the proposed work aims at implementing and analyzing various load balancing techniques in a cohesive environment using the Mininet testbed, along with the evaluation of various SDN controllers using a set of metrics for a better understanding of the various implementation hurdles, thus facilitating the development of efficient load balancing techniques in SDN environments.

III. SDN Load Balancing Algorithms

Software-Defined Networking (SDN) mainly consists of two essential layers: the control plane and the data plane. Applying load balancing mechanisms across these layers plays a crucial role in improving overall network efficiency. Fig. 1 shows the impact of load balancing. It indicates that, without load balancing, congestion on specific routes caused a higher packet drop rate. However, when the SDN load balancer was active, packets were efficiently distributed across multiple paths, reducing packet loss by over 50% in most experiments. Load balancing techniques are generally classified into two types: static algorithms, which function under fixed traffic patterns, and dynamic algorithms, which adapt to changing application requirements. Fig. 2 illustrates the different SDN load balancing approaches.

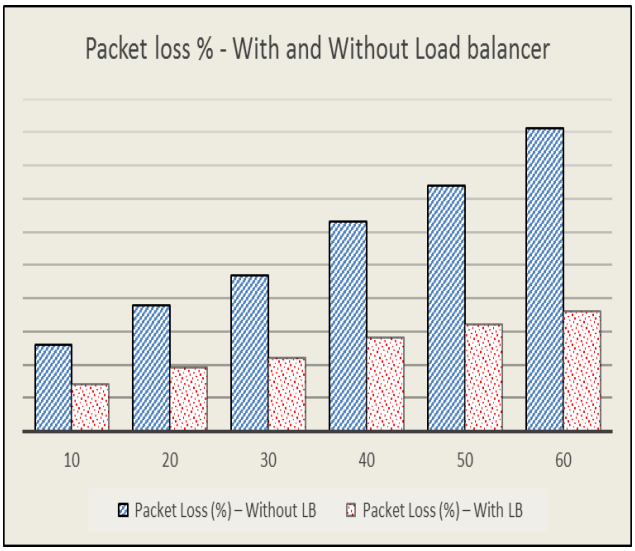

Fig. 1. SDN performance with and without load balancer.

A. Static Load Balancing

Static load balancing requires an in-depth analysis of server capabilities before deployment. The load distribution pattern is predetermined and does not adjust according to real-time traffic fluctuations. This method is best suited for environments where workloads remain relatively constant. Common static load balancing techniques include Round- Robin [25], Weighted Round-Robin, Random Selection [26] and Hashing-based load balancing [27].

1) Weighted Round Robin: Round Robin is a classic and straightforward load-balancing method that evenly distributes incoming requests among available servers in a rotating sequence. The standard Round Robin approach has been

enhanced into the Weighted Round Robin (WRR) method, which distributes client requests according to predefined server weights. These weights are determined based on factors such as current traffic load, server response times, available resources, or overall processing capacity. WRR allocates the load request according to each server's capacity. As a result, it provides improved performance, particularly in environments with heterogeneous servers. For example, in Fig. 3, Server 1, Server 2, and Server 3 are given weights of 3, 2, and 1, respectively. Based on the assigned weights, Server 1 manages the first three incoming requests, Server 2 handles the next two, and Server 3 receives the sixth request. After this sequence, the pattern begins again.

2) Random Selection: In this Random Selection strategy, each incoming task or request is allocated to a server picked at random from the available options. As it does not require complex computations or tracking of server states, the process is fast and has very low resource overhead. This simplicity makes it well-suited for environments with a limited number of incoming requests. However, as the volume of requests increases, the lack of awareness about server load, performance capacity, or network delays can lead to imbalances.

3) Hashing-based load balancing: Hashing-based load balancing provides a simple yet powerful mechanism for achieving load balancing in networks. These methods apply a hash function to certain characteristics, such as IP addresses, port numbers, or protocol types to generate a value that determines how the flow is routed. The resulting hash guides the selection of the appropriate path, server, or switch port, ensuring an even distribution of traffic. One of the key benefits is the speed and consistency of decisions. Once the hash function is established, routing decisions can be made quickly and reliably. Additionally, the technique incurs minimal overhead, as it does not require ongoing monitoring of network conditions. This makes it especially well-suited for large and complex topologies, where traffic must be efficiently spread across multiple available resources.

B. Dynamic Load Balancing

In dynamic load balancing, incoming requests are distributed from heavily loaded servers to those with lighter loads. This procedure necessitates the actual current load status of all servers across the network, allowing workloads to be shifted from overutilized to underutilized servers to maintain balance. Compared to static load balancing, dynamic methods offer improved efficiency by adapting to fluctuations in the load requests during runtime. They are particularly effective for applications with frequently changing load requests. However, implementing dynamic load balancing can be complex. There are three main approaches to dynamic load balancing: non-distributed, distributed, and semi-distributed. In the non-distributed approach, a single centralized node handles all incoming requests and assigns them to various servers. The distributed model allows all nodes to collaborate and share the task of request distribution. In the semi-distributed approach, the network is organized into several clusters, each overseen by a central node responsible for managing the distribution of requests within that cluster.

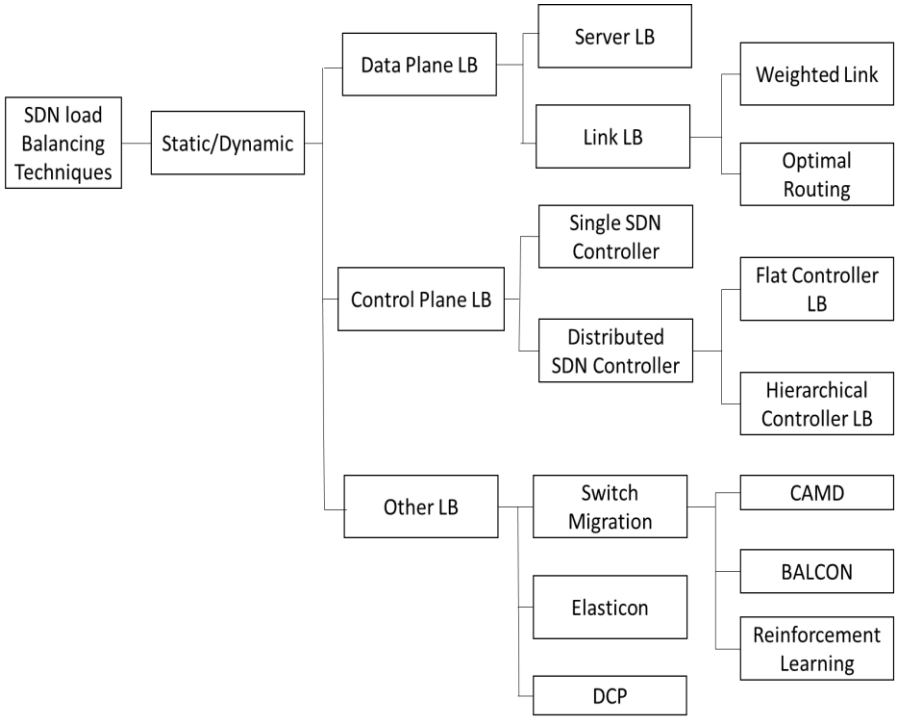

Fig. 2. Taxonomy of load balancing algorithms.

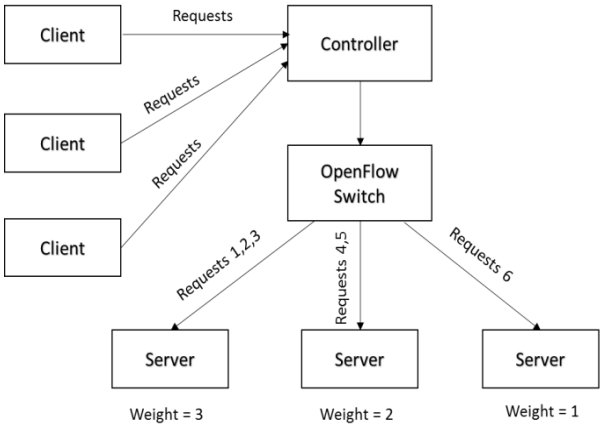

Fig. 3. Weighted Round Robin.

1) Least Connection algorithm: The Least Connection algorithm [28] is a dynamic load balancing method that routes incoming client requests to the server having the fewest active connections. This approach is illustrated in Fig. 4. For instance, consider a scenario with three servers: Server1 has two active connections, Server2 has four, and Server3 has none. In this case, the first two client requests are directed to Server3, as it has no ongoing transactions. Once Server1 and Server3 both have two active connections, the next request (the third) is assigned to Server1, and the fourth to Server3. From that point onward, the requests alternate between Server1 and Server3, similar to the Round Robin approach. Essentially, each new request is assigned to the server with the fewest active transactions; if two or more servers have an equal number of active connections, the Round Robin method is used to distribute the load. The limitation of the Least Connection algorithm is that servers can become overloaded if they handle long-duration connections. Moreover, this method is best-suited for environments with homogeneous server configurations.

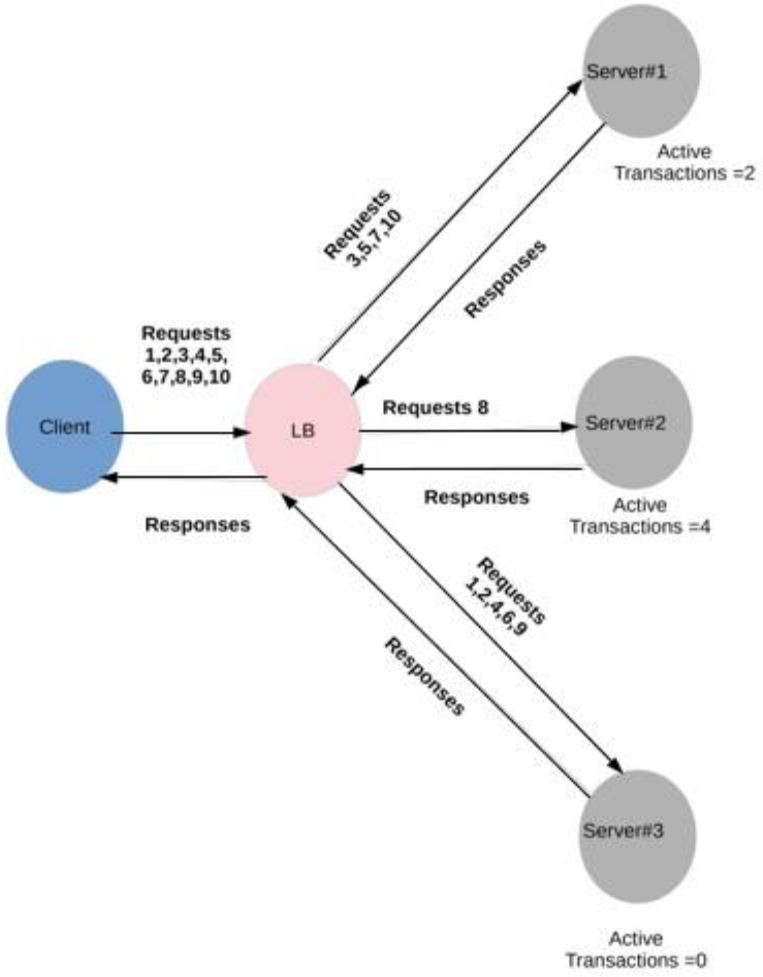

Fig. 4. Least Connection algorithm.

2) Dynamic Weight Random Selection (DWRS): The DWRS load balancing method enhances the standard Weighted Random Selection (WRS) algorithm by dynamically assigning weights to servers based on their current workload.

Although all servers may still receive the incoming requests, the server with the lowest load of requests has a greater chance of receiving additional requests, thereby reducing its idle time compared to other servers. The controller in the DWRS strategy continuously monitors each server's load and translates it into a value of weight using Eq. (1):

$$\text{Server Weight} = 10 - (\text{load at server}/10) \quad (1)$$

The server load, measured by the count of requests it receives, typically varies from 1 to 100. As described by Eq. (1), the server's weight is restricted to a maximum value of 10. This weight represents the server's relative probability of being chosen for incoming requests. Consequently, servers with higher loads are less likely to be selected as the target server. The controller keeps track of the weights assigned to each server. When a request is received, the controller first sums the weights of all servers to determine the total weight. A random number R that is smaller than this total weight is generated. A variable V is set to 0 and is gradually incremented by adding each server's weight in sequence of selection. As soon as V exceeds R, the process stops, and the request is forwarded to the server whose weight was recently added.

C. Data Plane Load Balancing

The data plane (DP) is responsible for directing packets to their appropriate destinations. It is also commonly referred to as the forwarding plane. The centralized SDN controller describes how the data plane is to behave in the network. The routing policies assigned to each flow are clearly established, although in some cases, multiple policies may apply to a single flow. Therefore, there might be many packets which use the particular route, and results in an increase in the volume of traffic. This surge in traffic leads to congestion along the path, which in turn raises delays and reduces the network's overall performance. To avoid congestion in the data plane, the path with the least load is chosen for the new incoming requests. Thus, a load balancer is required with predefined balancing policies. The data plane load balancing can be either of two types: Server LB or Link LB.

1) Server LB: The current existing traffic in the network is growing day-by-day, the service providers experience serious problem in server overload which results in network congestion leading to degraded performance and poor user experience. Therefore, the server load balancing plays a major role in network. This process distributes incoming requests across multiple servers to prevent any single server from becoming bottleneck. Server load balancing can be performed either in static or in dynamic algorithms. The both static and dynamic server LB algorithms are deliberated in [29].

2) Link LB: In SDN network, load balancing in multiple paths or links prevents congestion and enhances bandwidth, throughput and performance of the network. The link LB is achieved by two ways: a) Weighted link and b) Optimized routing.

a) Weighted links: Weights are assigned to the links [30] according to their current load, and packet distribution is carried out in proportion to these assigned weights. The controller monitors the link and, based on the load, delay, and packet loss as impact factors, load balancing is performed. A score is assigned to each link depending on its load, and the link with the highest score is chosen to carry the traffic. This load balancing with respect to the weighted links prevents the link to be congested, therefore reduces the packet loss and hence provides better network performances.

b) Optimal routing: Link load balancing can also be achieved by optimal routing by generating optimal routing tables. Optimal routing can be done by different algorithms. The LB mechanism uses a shortest path algorithm to determine the shortest route for each flow and therefore balances the traffic. When a switch receives a new flow request, it queries the controller for routing instructions. The controller calculates the best path and responds by updating the switch's flow table accordingly. Each switch maintains a collection of flow tables containing entries for each source-destination pair. The multi-flow load balancing algorithm determines alternative shortest paths between these source and destination nodes. This load balancing approach uses the source-destination pair as its input. The algorithm also gathers network topology information using protocols JSON and REST APIs, enabling it to map ports, links, MAC and IP addresses, and their associations with switches and ports.

D. Control Plane Load Balancing

The control plane can be of single or multiple controllers working together. The control plane (CP) defines the routing table for forwarding the data packets. The SDN Controller serves as a key element within the control plane, overseeing the management of switches and leveraging software applications to enable intelligent network operations. The load balancing at the Control plane can be done in two ways: 1) Single or centralized controller, and 2) Distributed controller.

1) Single controller LB: A centralized controller plays a crucial role in the control plane's performance by overseeing all network-related functions. The load balancing can be performed either in static or in dynamic method. In static LB the controller use the load balancing algorithms and drives the connected servers [31]. In a dynamic algorithm, a central controller regularly collects load information from all connected servers and uses this information to allocate incoming tasks accordingly. It is simple to implement and less overhead of network management. But when the flow in the network increases, the single centralized controller are overloaded with huge number of requests from switches and controller may develop bottleneck problem. This results in lower performance of the whole network. Data centers and service provider networks are typical examples of such network scenario. Data center network includes thousands of switching elements which are expected to create vast number of control events which led to overload the single centralized controller. Use of Multiple controllers is the solution to meet the traffic characteristics within the large-scale networks.

2) Distributed SDN controller LB : The distributed SDN controllers include set of controllers which collaborate with

each other to manage all the underlying network devices. Distributed controllers enhance the system's reliability and scalability problems. Load balancing mechanism is needed in control plane to distribute or share the loads among the multiple controllers [32] and also to transfer the loads from overloaded controller to underutilized controllers. This balancing of load amid the multiple controllers, improves the performance of the entire network. In a distributed controller setup, the control plane's load balancing component can be organized in either a flat or hierarchical structure.

a) Flat or horizontal controller LB: In this LB, each controller in the architecture has the same tasks. The controllers communicate with each other via specialized channels by using east-west interfaces to share the network information. The multiple controllers are coordinated by maintaining network topology information in the datacenter, which helps enhance the scalability of SDN networks [33]. Onix is one of the examples of distributed controllers that manage multiple controllers and various switches [34].

b) Hierarchical or vertical controller LB: In a vertical or hierarchical architecture, controllers are organized across several levels in a top-down structure. The local controllers are placed in the bottom layers, whereas the root controller or super controller is placed in the top layer. The super controller oversees a comprehensive overview of the load status across all lower-level controllers. This LB is time-consuming as it is to manage multiple controllers in multiple layers. Lamen is a system with supervisor controllers [35] that manages the other controllers, similar to a centralized architecture.

E. Other Techniques of Cp/Dp Load Balancing

As the real-time communication networks lead to an increase in traffic dynamically, various types of dynamic load balancing techniques exist both in data and control planes. In recent research, load balancing is done by techniques like switch migration, reinforcement learning, ElastiCon, distributed controller placement, resource-based LB, etc.

1) Switch migration: As the load in a network is dynamic, static assignments of switch to controller are not efficient. Hence, to achieve load balancing, it becomes necessary to transfer switches from heavily loaded controllers to less burdened ones. This is accomplished by continuously monitoring the switches and identifying those with the highest traffic flow, which are then dynamically reassigned to a lightly loaded controller. In Mininet, switch migration denotes reassigning a switch to a different controller. As mininet is a network emulator, this process generally involves reconfiguring the control plane connections rather than relocating actual hardware. This mechanism improves the controller availability, but selecting the switch to get migrated in balancing the load is a challenging task in this mechanism. In [36], the load balancing is improved by 14% compared to the existing approach. The migration cost is reduced in the load balancer proposed in [37].

2) Reinforcement Learning: Reinforcement Learning (RL) is a branch of machine learning where an agent learns to make optimal decisions by interacting with its environment and receiving evaluative feedback in the form of rewards or penalties. When applied to load balancing tasks, RL provides an adaptive and intelligent approach in distributing workloads across servers or network paths, with the goal of improving overall system efficiency and responsiveness.

In RL-based load balancing systems, the agent acts as a smart controller that continuously observes system metrics such as server utilization, traffic patterns, response time and latency. Based on these observed metrics, it determines the appropriate allocation of incoming tasks or data flows. Each action is evaluated by examining its system efficiency by focusing on reducing latency and improving throughput. Through iterative learning and real-time feedback, the agent refines its strategy to achieve optimal performance.

This approach is particularly advantageous in complex and dynamic environments, such as cloud platforms, edge networks, and SDN. Unlike traditional static or rule-based methods, RL-based load balancing can adapt to fluctuating workloads and network conditions, offering a more flexible and robust solution. The cost of migration of flows between the controller and switch is reduced in [38]. The RL-based load balancer [39] improved performance by accurate decision-making of switch migration.

3) Elastic SDN controller: An Elastic Software-Defined Networking (SDN) controller refers to a dynamic and adaptable control plane mechanism that can scale in response to fluctuating network demands. Unlike traditional fixed controllers, an elastic SDN controller setup is capable of automatically adjusting the number of active controller instances based on factors such as traffic load, network size, and application requirements.

This adaptability is made possible through approaches like deploying multiple controllers across the network, distributing traffic loads efficiently, and allowing for the seamless placement or relocation of controller entities. The primary objective is to maintain system reliability, ensure low latency, and optimize network performance, regardless of traffic variations. The load balancer proposed in [13] and [40] implements an elastic SDN controller and shows improved performance with dynamic traffic change. The LB [41] is designed to support dynamic addition and removal of controllers.

4) Resource-based load balancing: Resource-based load balancing within the Mininet environment involves intelligently distributing network traffic across various paths or nodes by considering the current state and usage of system resources. In this method, factors such as CPU utilization, memory load, bandwidth consumption, and queue occupancy are used to guide forwarding decisions.

In practice, this strategy is implemented by configuring SDN controllers like Ryu, POX, or ONOS to actively monitor the network's performance. These controllers gather real-time data on resource usage and use it to adjust flow rules on OpenFlow-enabled switches. This enables the network to adapt

dynamically by rerouting traffic away from congested links or overloaded devices toward less-utilized alternatives, ensuring better load distribution and improved performance. The adaptive resource allocation scheme in [42] shows effective even in large-scale SDN.

5) Distributed Controller Placement (DCP): The DCP method determines both the number of SDN controllers needed and where they should be positioned within the network. This method is completely topology dependent. To solve the problem of the availability and scalability of a single SDN controller, distributed controller placement is employed. To improve the SDN network, the positioning of the controller needs to be considered. Given the positions of the switches, the Controller Placement Problem (CPP) selects optimal locations for controllers to reduce the latency between switches and controllers. The approach described in [43] addresses both delay reduction and load balancing. This method achieves a well-balanced load distribution with only a slight increase in latency. In [44], the number of required controllers is chosen with respect to support the load and the locations of controller is determined with the aspect of communication latencies.

IV. SDN Simulators and Emulators

SDN simulators use abstract representations of network topologies and protocols, making them ideal for evaluating algorithms and testing scalability. Emulators, on the other hand, recreate real network environments by running actual software like SDN controllers and OpenFlow switches, providing realistic behavior and supporting real-time interaction.

The comparison table of simulator and emulator tools is shown in Table I. Mininet [45] enables real-time interaction with actual SDN controllers, whereas simulators provide abstract models, often needing additional modules for OpenFlow support and lacking the ability to run real controller code. Unlike EstiNet, which is powerful and may require licensing, Mininet is free, open-source, and user-friendly. While GNS3 focuses on traditional network and device emulation, Mininet is specifically designed for SDN, making it more effective for SDN-focused experimentation. Hence, Mininet is widely regarded as the leading tool for SDN, particularly in research and development. As a lightweight emulator, it enables users to create virtual networks on a single machine or virtual machine, allowing for interaction with virtual hardware and testing of SDN applications. Fig. 5 illustrates the percentage of usage for various simulators, where the graph indicates that Mininet is utilized more extensively than the other tools. Table I describes the features of various SDN simulators.

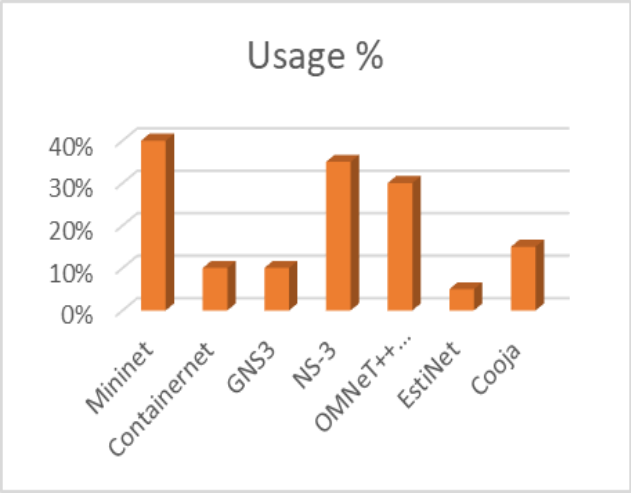

Fig. 5. Usage percentage of different simulators.

TABLE.I Types of Simulators and Emulators

Tools	Type	OpenFlow Support	Emulates Real network interfaces	Scalability	User-friendly interface	Features
Mininet	Emulator	Yes	Yes	100 – 1000 nodes	Excellent	Fast prototyping, integrates with real controllers
Containernet	Emulator	Yes	Yes	Hundreds of containers	Good	Docker containers as hosts; good for app testing
GNS3	Emulator	Partial (via VM integration)	Yes	50–100 nodes	Moderate	Real router images, GUI-based setup
NS-3	Simulator	Yes(via OFSwitch13)	No	Tens of thousands of nodes	Poor	Detailed protocol simulation, high scalability
OMNeT++ (INET)	Simulator	Indirect (requires modules)	No	Thousands of nodes	Poor	Modular simulation, protocol-level granularity
EstiNet	Sim/Emu	Yes	Yes	Thousands of virtual nodes	Moderate	Commercial-grade GUI, supports hybrid simulation
Cooja	Simulator	Limited (IoT-focused)	Yes	100–1,000 nodes (depending on mote type)	Moderate	SDN for WSN, energy-aware routing support

V. SDN Controllers

In SDN, the controller acts as the centralized brain of the network. The SDN controller resides in the control plane and manages the network by dynamically adjusting routing, load balancing, and access policies using software-based logic. In essence, the SDN controller functions as the decision-making hub of the network. It discovers and maps the network, defines how traffic should be handled, applies control policies, monitors activity, and facilitates communication between the control logic and the underlying hardware. These tasks collectively allow for flexible, programmable, and adaptive network management.

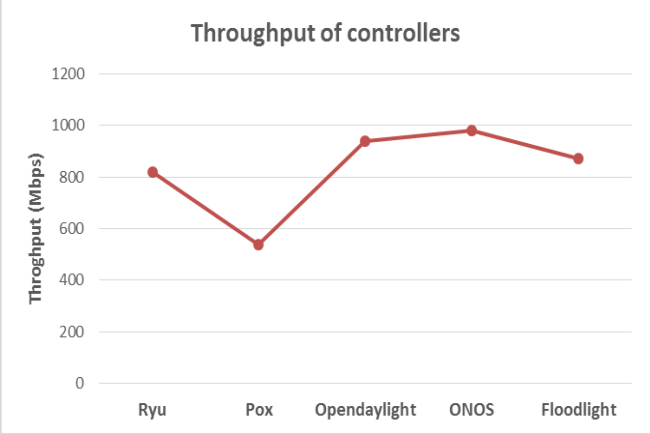

Fig. 6. Throughput of SDN controllers.

In Mininet, a few controllers are available for SDN simulation. In comparative analysis of SDN controllers, Ryu [46], POX, OpenDaylight (ODL), Open Network Operating System (ONOS), and Floodlight are evaluated based on throughput and Round-trip time (RTT). Throughput for SDN controllers is determined by measuring the volume of data successfully delivered within a given time frame using traffic generation tools such as iPerf. As illustrated in Fig. 6, the POX controller demonstrates significantly lower throughput compared to the other controllers. Round-trip time (RTT) for each controller is evaluated using the ping command, with the results shown in Fig. 7. The analysis reveals that the POX controller records the highest RTT, whereas the ONOS controller achieves the lowest. Table II differentiates the different types of controllers and its features.

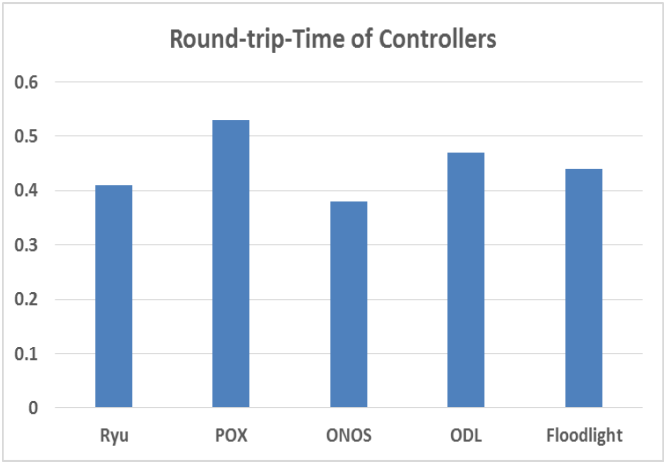

Fig. 7. Round-trip time of different controllers.

TABLE.II TYPES OF CONTROLLERS AND THEIR FEATURES

Name of Controller	Supported Platform	Supported Programming Language	Features
Nox	Linux	Python	First OpenFlow controller. The new version of Nox is released with multithreaded manner
Pox	Linux, Mac, Windows	Python	Upgraded version of Nox. It requires less memory but throughput performance is less.
Floodlight	Linux	Java	Designed as Concurrent systems. It achieves better performance in Organizational networks and Datacenters
Trema	Linux	C, Ruby	Designed for Cloud Infrastructure
Mastero	Linux, Mac, Windows	Java	Designed as Concurrent systems
Beacon	Linux, Mac, Windows	Java	Supports Multithreading and achieves better performance
Ryu	Linux	Python	Component-based Framework. Can create new network management and control applications. Designed for Cloud infrastructure.
OpenDaylight	Linux, Mac, Windows	Java	The recent version supports for the Internet of Things(IoT)
ONOS	Linux, Mac, Windows	Java	Includes Distributed network operating system. Increases availability and scalability naturally.

TABLE.III TYPES OF TRAFFIC GENERATORS AND THEIR FEATURES

Traffic Generator	Supported Traffic Types	Flow Scalability	Performance Metrics Measured	Advantages	Limitations
Iperf / Iperf3	TCP, UDP	Moderate (up to 20k with parallel mode)	Throughput / Bandwidth, Total Data Transferred, Jitter (UDP), Packet Loss	Simple to use, widely supported, reliable performance metrics	Limited application diversity, CPU-bound for large flows
D-ITG	TCP, UDP, ICMP, VoIP, video, gaming	High (up to 50k)	Throughput / Bandwidth, Total Data Transferred, Jitter (UDP), Packet Loss, End-to-End Delay (Latency)	Supports diverse application profiles, good scalability	Slightly complex configuration compared to Iperf
Pktgen	Raw Ethernet packets	Very high (50k–100k+)	Throughput / Bandwidth, Total Data Transferred	Extremely fast, low-level packet control	Requires kernel tuning, less suited for application-level testing
Ostinato	Layer 2–7 protocols	Low-Moderate (10k)	Throughput / Bandwidth, Total Data Transferred, Jitter (UDP), Packet Loss , End-to-End Delay (Latency)	GUI interface, flexible packet design	GUI overhead, not ideal for bulk flow generation
Mininet Built-in CBR/UDP	UDP	Very low (2k)	Throughput / Bandwidth, Total Data Transferred, Jitter (UDP), Packet Loss	Lightweight, no extra installation	Limited functionality, not scalable

VI. Traffic Generators in Mininet

Mininet supports a variety of tools [47] to generate and analyze network traffic, aiding in performance testing and simulation of real-world conditions.

1) Ping: The ping command is used in Mininet to generate basic network traffic by sending ICMP Echo Request packets from one host to another. For example, the command h1 ping h2 sends requests from host1 to host2, which replies with ICMP Echo Reply packets. This tool is useful for verifying connectivity and measuring round-trip time (RTT) between hosts.

2) IPerf: IPerf is a powerful and flexible tool for generating traffic and assessing network performance over TCP and UDP. Operating in a client-server model, it measures metrics like interval, data transfer rate, and bandwidth with iperf-TCP, while with iperf-UDP, along with those details it also provides jitter, packet loss statistics, and loss percentages, making it suitable for testing under various traffic conditions.

3) D-ITG (Distributed Internet Traffic Generator): D-ITG is a comprehensive tool for simulating a broad spectrum of traffic types using TCP, UDP, ICMP, SCTP, and DCCP protocols. It provides precise control over various traffic characteristics, including packet size, arrival intervals, transmission rate, and duration. Operating in a client-server model, ITGSend generates traffic, ITGRecv captures it, and ITGDec analyzes log files to extract metrics such as jitter, throughput, packet loss and delay. D-ITG is ideal for detailed research and controlled testing.

4) Mgen (Multi-generator): Developed by the U.S. Naval Research Laboratory, Mgen is a script-based traffic generator designed for real-time network traffic simulation. It supports both TCP and UDP, including unicast and multicast. Users define traffic flows via scripts specifying parameters like start/stop time, destination, port, packet size, and rate. Mgen runs as two components: mgen input to generate traffic and mgen output to log performance. It provides metrics such as latency, jitter, packet loss, and packet delivery ratio, making it ideal for QoS evaluation and repeatable network experiments.

Table III summarizes a few more types of traffic generators and their features.

VII. Performance Metrics for Evaluating SDN Load Balancer

The evaluation of an SDN load balancer requires analyzing various performance metrics that indicate the network's efficiency, utilization of resources, and overall responsiveness. The essential metrics and tools listed below are used to evaluate these parameters when conducting simulations in environments like Mininet.

1) Throughput: It measures the rate at which data packets are successfully delivered over the network. It is calculated using Eq. (2):

$$\text{Throughput} = \frac{Dp * Rp}{\text{Total duration of simulation}} \quad (2)$$

where, D_p represents the count of packets delivered and P_s denotes the size of a packet. IPerf tool can be used to measure the throughput.

```
# On the receiving host (e.g., h2)
h2 iperf -s

# On the sending host (e.g., h1)
h1 iperf -c h2
```

Fig. 8. IPerf commands.

The commands in Fig. 8 test can be repeated across various load balancing algorithms to compare throughput performance.

2) Latency (End-to-end delay): It measures the time a data packet takes to travel from the source to the destination. To measure latency, ping command can be used. The average RTT value from the output reflects the average latency.

3) Packet loss rate: This metric indicates the proportion of packets that fail to reach their intended destination, which may occur due to congestion or link failure. Ping command can be used and from the output summary packet loss percentage can be calculated as Eq. (3):

$$\text{Loss Packet\%} = \frac{Dp - Rp}{\text{Dp}} * 100 \quad (3)$$

where, Dp =Count of delivered packets and Rp = Count of received packets.

4) Packet Delivery Ratio (PDR): PDR measures the ratio of packets successfully received to those originally sent, as determined using Eq. (4). A higher PDR reflects better reliability in packet transmission.

$$\text{PDR} = \frac{\text{Total packets Received}}{\text{Total packets Sent}} \quad (4)$$

5) Response time: This is the time interval between sending a request and receiving a response, especially relevant for application-layer communications. Application-level logging or timestamping (e.g., with curl, custom web apps, or controller logging) can be used to determine response duration [see Eq. (5)].

$$\text{Server Utilization\%} = \frac{\text{Server Active Time}}{\text{Total Simulation Time}} * 100 \quad (5)$$

6) Quality of Service (QoS) metrics: Bandwidth Allocation can be analyzed by comparing the expected allocated bandwidth versus the actual utilized bandwidth using tools like iperf and network monitoring scripts.

7) Scalability: It checks how the load balancer performs as the network size or traffic increases. In Mininet, the number of hosts or switches is increased, and the key metrics (throughput, latency, controller load) can be monitored to determine how much they vary with scale.

VIII. Static versus Dynamic Load Balancer

Static algorithms, including RR, RS, LTWRR, and WRR, together with the dynamic algorithm DWRS [9] [48], were

implemented and their performance was systematically evaluated within the Mininet environment. The iperf-udp utility is employed to simulate network traffic. The configuration consists of three servers and three clients, where all clients simultaneously transmit iperf traffic to the servers. Fig. 9 presents the results obtained from the iperf-UDP tool. To assess the performance of each load balancing algorithm, two traffic conditions were emulated, consisting of 100 and 200 client requests directed to the servers.

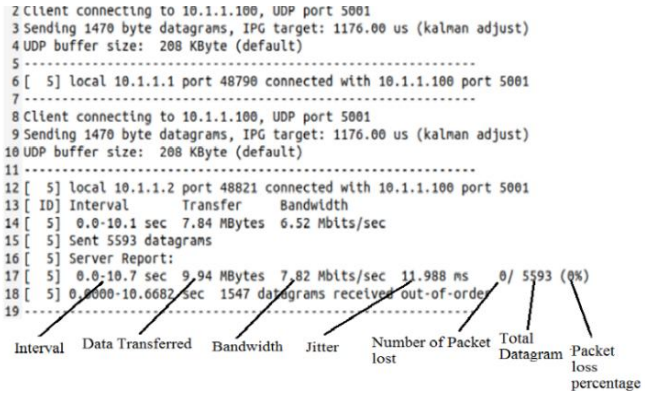

Fig. 9. Output of iperf-udp traffic generator.

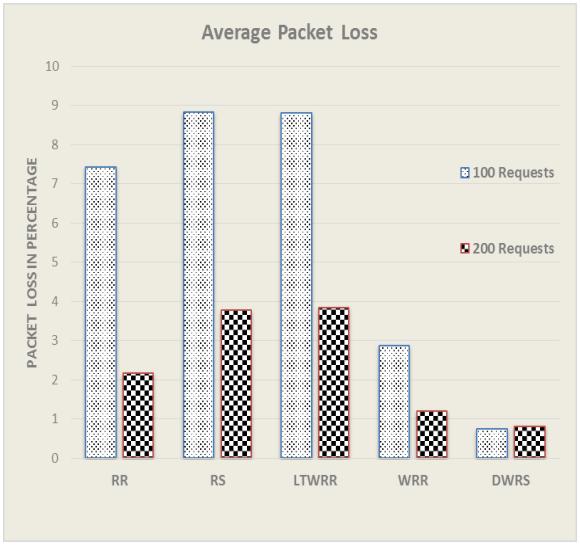

Fig. 10. Packet loss percentage.

Fig. 10 illustrates the packet loss percentages for both static algorithms and the Dynamic Weighted Random Selection (DWRS) approach. According to the graph, the Round Robin (RR) and Random Selection (RS) algorithms exhibit the highest packet loss rates. The LTWRR algorithm, a static method focused primarily on link delay, also shows significant packet loss. Although WRR is a static algorithm, it assigns weights based on server capacity, resulting in a lower packet loss rate than LTWRR. DWRS outperforms the other methods in minimizing packet loss by dynamically modifying weights based on real-time network load. The graph clearly indicates that dynamic load balancing techniques yield better performance compared to static algorithms. Additionally, the average response time of the load balancing techniques is measured using the curl command, and the outcomes are illustrated in Fig. 11. Among the tested approaches, DWRS achieves a lower response time compared to the other load balancers.

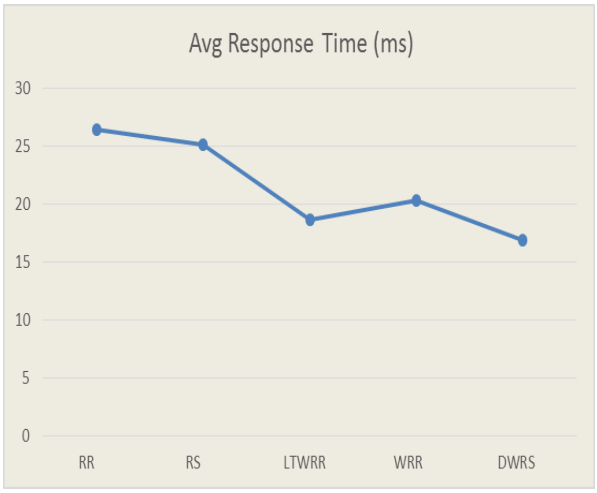

Fig. 11. Average response time of load balancers.

Both static and dynamic approaches are simulated under the same conditions, allowing for an objective assessment of the solution. The comparison of the controller and the traffic generators is the key, which points out the benefits, as well as its drawbacks, especially with regard to flexibility, efficiency, and scalability, which is crucial in validating the solution as well as positioning it with regard to the related work, thus adding value to the study itself. A range of SDN optimization approaches is examined in [49], including a comparative analysis of algorithms designed for routing (both static and dynamic), packet forwarding, load balancing, traffic management, and reducing forwarding delays.

IX. Future Enhancements

The results of this study indicate that no single load balancing strategy is best in the context of Software Defined Networking environments. While static load balancing is best due to its simplicity and less control overhead, the strategy is less effective due to its lack of adaptability to changing network environments. On the other hand, the dynamic nature of the load balancing strategy is more effective in terms of its responsiveness to changing network environments; however, the strategy also incurs additional overhead at the controller level.

In terms of the practical implications of the study, the use of the Mininet tool is effective in terms of its flexibility and cost, however, the tool may not accurately represent real-world environments. Therefore, the results of the study are valuable if further tested in real-world and large scale environments. This study also highlights the importance of using the right load balancing strategy in SDN environments and the need to develop more effective solutions that are adaptive to changing network environments. Future work can focus on hybrid approaches that balance performance and overhead.

After reviewing some of the existing load balancing algorithms, few open issues are analyzed and summarized below:

1) Early detection of controller overload: There is a growing need for mechanisms that can identify potential overload conditions in SDN controllers before they occur. A promising direction for improvement involves the application of machine learning and artificial intelligence to forecast traffic patterns and facilitate adaptive, real-time traffic distribution. These intelligent techniques can utilize both past and present network data to mitigate congestion proactively and ensure efficient use of resources.

2) Unified framework: The existing algorithms addresses either server-side or controller side load distribution but do not provide a unified framework that balances both controller load and server load simultaneously.

3) Scalability through distributed architectures: Many existing dynamic load balancing techniques depend on a centralized controller, which poses scalability and fault-tolerance challenges in large-scale deployments. To address these limitations, distributed and hierarchical controller architectures can be introduced. These models help eliminate bottlenecks and single points of failure, offering better scalability and improved reliability while maintaining efficient load balancing.

4) Security-aware load balancing: Finally, the existing load balancing algorithms neither considered about the security issues in forwarding the requests. Incorporating security-aware mechanisms into load balancing strategies will be essential to ensure that traffic distribution upholds data integrity and confidentiality, especially in complex and rapidly changing network environments.

X. Conclusion

SDN has become a major focus of academic and industrial research in recent years. This work focuses on implementing load balancing algorithms within the Mininet simulation environment using SDN. The study addresses the challenges in designing efficient load balancing solutions and outlines the distinct features of various SDN controllers. Both static and dynamic load balancing techniques were examined, and their comparative performance was analyzed to demonstrate the trade-offs between simplicity, responsiveness, and scalability. By analyzing different load-balancing strategies and evaluating multiple SDN controllers, the work provides practical insights into their strengths, limitations, and suitability under varying network conditions. The findings highlight the importance of adaptive and scalable approaches to address real-world challenges such as dynamic traffic and controller overhead. The results indicate that while static algorithms provide lightweight and predictable performance, dynamic algorithms are more adaptive to fluctuating network conditions but introduce higher complexity. The study also provides an overview of traffic generators available in Mininet, highlighting their features and the performance metrics that can be evaluated using them. By bringing together these findings, the study not only evaluates current strategies but also identifies open challenges that demand further exploration. The outcomes of this study provide valuable guidance for researchers and practitioners aimed at creating more resilient, adaptive, and scalable load balancing strategies for SDN environments.

References

[1] M. Keerthi Prabhu and S. Veena, "Implementation of Load Balancing Algorithms in Software Defined Networks using OpenDaylight," 2024 4th International Conference on Intelligent Technologies (CONIT), Bangalore, India, 2024, pp. 1-6, doi: 10.1109/CONIT61985.2024.10627596.

[2] Omer, Yassin Abdulkarim Hamdalla, Amin Babiker A. Mustafa, and Ashraf G. Abdalla. "Performance analysis of round robin load balancing in SDN." 2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE). IEEE, 2023.

[3] Prabhu, M. Keerthi, and S. Veena. "Implementation of Load Balancing Algorithms in Software Defined Networks using OpenDaylight." 2023 4th International Conference on Intelligent Technologies (CONIT). IEEE, 2024.

[4] Sharma R, Reddy H 2019 Effect of load balancer on software-defined networking (SDN) based cloud. IEEE 16th India Council International Conference (INDICON) (pp. 1-4). IEEE.

[5] Alssaheli, Omran MA, et al. "Software defined network based load balancing for network performance evaluation." International Journal of Advanced Computer Science and Applications 13.4 (2022).

[6] Kumar, Vishwas, Sandeep Jangir, and Deven G. Patanvariya. "Traffic load balancing in SDN using round-robin and Dijkstra based methodology." 2022 International Conference for Advancement in Technology (ICONAT). IEEE, 2022.

[7] Singh, Irengbam Tilokchan, Thounaojam Rupachandra Singh, and Tejmani Sinam. "Server load balancing with round robin technique in sdn." 2022 International Conference on Decision Aid Sciences and Applications (DASA). IEEE, 2022.

[8] Tache, Maria Daniela, Ovidiu Păscuțoiu, and Eugen Borcoci. "Optimization algorithms in SDN: Routing, load balancing, and delay optimization." Applied Sciences 14.14 (2024): 5967.

[9] Wijaya, Chandra, et al. "Load balancing algorithm in a software-defined network environment with round robin and least connections." International Conference on Smart Grid and Internet of Things. Cham: Springer Nature Switzerland, 2023.

[10] Joshi, Neha, and Deepak Gupta. "Application layer load balancing in software defined networking using priority based round robin scheduling algorithm." Wireless Personal Communications 136.2 (2024): 759-772.

[11] Manasa, B. ., and A. R. . Babu. "Dynamic Weighted Round Robin Approach in Software-Defined Networks Using Pox Controller". International Journal on Recent and Innovation Trends in Computing and Communication, vol. 11, no. 5, May 2023, pp. 304-10, doi:10.17762/ijritcc.v11i5.6618.

[12] Sahoo, Kshira Sagar, and Bibhudatta Sahoo 2019 CAMD: a switch migration based load balancing framework for software defined networks." IET Networks 8, no. 4: 264-271.

[13] Cello, Marco, Yang Xu, Anwar Walid, Gordon Wilfong, H. Jonathan Chao, and Mario Marchese. 2017 BalCon: A distributed elastic SDN control via efficient switch migration. IEEE International Conference on Cloud Engineering (IC2E), pp. 40-50. IEEE.

[14] Yeo, Sangho, Ye Naing, Taeha Kim, and Sangyoon Oh. 2021 Achieving Balanced Load Distribution with Reinforcement Learning-Based Switch Migration in Distributed SDN Controllers. Electronics 10, no. 2: 162.

[15] Vyakaranal, Shashidhara B., and Jayalaxmi G. Naragund. 2019 Weighted round-robin load balancing algorithm for software-defined network. In Emerging Research in Electronics, Computer Science and Technology, pp. 375-387. Springer, Singapore, 2019.

[16] Kaur, Karamjeet, Sukhveer Kaur, and Vipin Gupta 2017 Least time based weighted load balancing using software defined networking. Advances in Computing and Data Sciences: First International Conference, ICACDS 2016, Ghaziabad, India, November 11-12, 2016, Revised Selected Papers 1. Springer Singapore.

[17] Chiang, Mei-Ling, Hui-Sheng Cheng, Hsien-Yi Liu, and Ching-Yi Chiang 2021 SDN-based server clusters with dynamic load balancing and performance improvement. Cluster Computing 24: 537-558.

[18] Nkosi, Mpho C., Albert A. Lysko, and S. Dlamini. 2018 Multi-path load balancing for SDN data plane. International Conference on Intelligent and Innovative Computing Applications (ICONIC), pp. 1-6. IEEE, 2018.

[19] Ruelas, A.M., Rothenberg, C.E., 2018. A load balancing method based on artificial neural networks for knowledge-defined data center networking, Proceedings of the 10th Latin America Networking Conference, pp. 106–109.

[20] Jamali, S., Badirzadeh, A., Siapoush, M.S., 2019. On the use of the genetic programming for balanced load distribution in software defined networks. Digital Communications and Networks 5, 288–296.

[21] Sun, P., Lan, J., Guo, Z., Xu, Y., Hu, Y., 2020b. Improving the scalability of deep reinforcement learning-based routing with control on partial nodes, ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE. pp. 3557–3561.

[22] Lucas Soares da Silva, Lucas Soares, Carlos Renato Storck, and Fátima de LP Duarte-Figueiredo. 2019 A Dynamic Load Balancing Algorithm for Data Plane Traffic. LANOMS. 2019.

[23] Shang, Zhihao, Han Wu, Guang Peng, and Katinka Wolter. 2019 Dynamic load balancing in the control plane of software-defined networks. IEEE 19th International Conference on Communication Technology (ICCT), pp. 947-953. IEEE.

[24] El Kamel, Ali, and Habib Youssef. 2020 Improving Switch-to-Controller Assignment with Load Balancing in Multi-controller Software Defined WAN (SD-WAN). Journal of Network and Systems Management: 1-23.

[25] Kaur, Sukhveer, Krishan Kumar, Japinder Singh, and Navtej Singh Ghumman. "Round-robin based load balancing in Software Defined Networking." In 2015 2nd international conference on computing for sustainable global development (INDIACom), pp. 2136-2139. IEEE, 2015.

[26] Li, Jun, Xiangqing Chang, Yongmao Ren, Zexin Zhang, and Guodong Wang. 2014 An effective path load balancing mechanism based on SDN. IEEE 13th international conference on trust, security and privacy in computing and communications, pp. 527-533. IEEE.

[27] Hsu, Shih-Wen, et al. "Design a hash-based control mechanism in vSwitch for software-defined networking environment." 2015 IEEE international conference on cluster computing. IEEE, 2015.

[28] Shengsheng, Yu, et al. 2005 Least-Connection Algorithm based on variable weight for multimedia transmission. http://www.wseas.us/elibrary/conferences/skiathos2002/papers/447-144.pdf. Accessed on January. Vol. 7.

[29] Zhang, H., Guo, X., 2014. SDN-based load balancing strategy for server cluster, in: Cloud Computing and Intelligence Systems (CCIS), IEEE 3rd International Conference on, IEEE. pp. 662– 667.

[30] Wang, Chunzhi, Gang Zhang, Hui Xu, and Hongwei Chen. 2016 An ACO-based link load-balancing algorithm in SDN. In 2016 7th International Conference on Cloud Computing and Big Data (CCBD), pp. 214-218. IEEE.

[31] R. Curtis, J. C. Mogul, J. Tourrihes, and P.Yalagandula.2011 Scaling Flow Management for High-Performance Networks. In Proc. SIGCOMM.

[32] Hu, Yannan, Wendong Wang, Xiangyang Gong, Xirong Que, and Shiduan Cheng. 2012 Balanceflow: controller load balancing for openflow networks. IEEE 2nd international conference on cloud computing and intelligence systems, vol. 2, pp. 780-785. IEEE.

[33] Tam, A.S.W., Xi, K., Chao, H.J., 2011. Use of devolved controllers in data center networks, in: Computer Communications Workshops (INFOCOM WKSHPS), IEEE Conference on, IEEE. pp. 596–601.

[34] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L.,Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., et al.,2010. Onix: A distributed control platform for large-scale production networks. OSDI, pp. 1–6.

[35] Nayyer, A., Sharma, A.K., Awasthi, L.K., 2019 A supervisor controller based scalable framework for software defined networks. Computer Networks 159, 125–134.

[36] Al-Tam, Faroq, and Noélia Correia. "On load balancing via switch migration in software-defined networking." IEEE Access 7 (2019): 95998-96010.

[37] Wang, Chuan'an, Bo Hu, Shanzhi Chen, Desheng Li, and Bin Liu 2017 A switch migration-based decision-making scheme for balancing load in SDN. IEEE Access 5 (2017): 4537-4544.

[38] Kumari, Abha, Arghyadip Roy, and Ashok Singh Sairam. "Optimizing SDN controller load balancing using online reinforcement learning." IEEE Access (2024).

[39] Yeo, Sangho, et al. "Achieving balanced load distribution with reinforcement learning-based switch migration in distributed SDN controllers." Electronics 10.2 (2021): 162.

[40] Dixit, Advait Abhay, et al. "Elasticon: an elastic distributed sdn controller." Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems. 2014.

[41] V. Yazici, M. O. Sunay, and A. O. Ercan, 2014 controlling a software-defined network via distributed controllers arXiv preprint arXiv: 1401.7651.

[42] Lin, Frank Po-Chen, and Zsehong Tsai. 2019 Hierarchical edge-cloud SDN controller system with optimal Lin, Frank Po-Chen, and Zsehong Tsai. 2019 Hierarchical edge-cloud SDN controller system with optimal adaptive resource allocation for load-balancing. IEEE Systems Journal 14, no. 1: 265-276.

[43] Hu, Ying, Tao Luo, Wenjie Wang, and Chunxue Deng. 2016 On the load balanced controller placement problem in Software defined networks." In 2016 2nd IEEE International Conference on Computer and Communications (ICCC), pp. 2430-2434. IEEE.

[44] Talhar, Priyanka, and Amol P. Bhagat.2018 An adaptive approach for controller placement problem in software defined networks. International Conference on Research in Intelligent and Computing in Engineering (RICE), pp. 1-11. IEEE, 2018.

[45] Khan, Sikandar, et al. "Emulating software defined network using mininet-ns3-WIFI integration for wireless networks." Wireless Personal Communications 118.1 (2021): 75-92.

[46] Albu-Salih, Alaa. (2022). Performance Evaluation of Ryu Controller in Software Defined Networks. Journal of Al-Qadisiyah for Computer Science and Mathematics. 14. 1. 10.29304/jqcm.2022.14.1.879.

[47] Khudhair, Tabarak, and Omar Athab. "Recent Tools of Software-Defined Networking Traffic Generation and Data Collection." Al-Khwarizmi Engineering Journal 21.2 (2025): 93-105.

[48] Shona, M and Rinki Sharma. "Implementation and Comparative Analysis of Static and Dynamic Load Balancing Algorithms in SDN." 2023 International Conference for Advancement in Technology (ICONAT) (2023):1-7.

[49] Tache, Maria Daniela, Ovidiu Păscuțoiu, and Eugen Borcoci. "Optimization algorithms in SDN: Routing, load balancing, and delay optimization." Applied Sciences 14.14 (2024): 5967.