Design and Deployment of a Dynamic Weighted Round-Robin SDN Load Balancing Mechanism with Distributed Controllers

M. Shona
Ramaiah University of Applied Science, Bengaluru, Karnataka, India
mshona.cs.et@msruas.ac.in (corresponding author)

Rinki Sharma
Ramaiah University of Applied Science, Bengaluru, Karnataka, India
rinki.cs.et@msruas.ac.in

Received: 16 June 2025 | Revised: 18 September 2025 and 15 October 2025 | Accepted: 18 October 2025

 DOI: https://doi.org/10.48084/etasr.12773

ABSTRACT

Software-Defined Networking (SDN) represents an innovative paradigm in network design that improves manageability, scalability, and adaptability. Within SDN, load balancing serves as a key component in optimizing network performance. SDN relying on a single centralized controller can lead to performance issues, especially under high traffic or excessive requests. To address this challenge, this study introduces an improved SDN load balancer that uses a Dynamic Weighted Round Robin algorithm with Distributed Controller (DWRR-DC), implemented using the Ryu controller. The proposed algorithm was implemented along with some existing load balancing methods, such as Random Selection (RS), Round Robin (RR), Least Time Weighted Round Robin (LTWRR), Weighted Round Robin (WRR), and Dynamic Weighted Random Selection (DWRS), emulated using the Mininet platform with a huge volume of simulated network traffic. Their performance was analyzed with respect to throughput and packet loss percentage, and the results were compared. The response time of the proposed and existing algorithms was also compared using the generated HTTP requests. Finally, the optimal number of controllers for the proposed algorithm was also obtained.

Keywords-dynamic load balancing; Software Defined Networking (SDN); Ryu controller; Mininet

I. INTRODUCTION

Software-Defined Networking (SDN) introduces a transformative approach by decoupling the control plane from the data plane in networking devices. This separation allows for centralized and flexible network management through software-based controllers. Within this framework, an SDN load balancer plays a crucial role in optimizing the distribution of traffic among multiple servers or network paths. As Internet technologies continue to advance rapidly, servers are faced with increasingly heavy workloads. Thus, it is essential to distribute this load effectively across servers to ensure reliable and efficient service delivery to end users. If the load is not evenly distributed, it can significantly degrade the overall performance of the network. This centralized control results in better resource allocation, enhanced network performance, and simpler scalability compared to conventional hardware-based load balancers. However, existing Round-Robin (RR) SDN load balancing approaches struggle to accommodate dynamic traffic and face scalability limitations, making them less suitable for real-time applications.

This paper proposes DWRR-DC, a dynamic load balancing strategy designed to enhance throughput and response time, effectively addressing the scalability limitations typically faced by SDN controllers. The functionality of the proposed SDN load balancer was tested using Mininet [1]. Some existing algorithms, such as Random Selection (RS), RR, Least Time Weighted Round-Robin (LTWRR), Weighted Round-Robin (WRR), and Dynamic Weighted Random Selection (DWRS), are also implemented and compared with the proposed algorithm.

In [1], the importance of SDN load balancing was explored. In [2], the advantages of dynamic SDN load balancing compared to static load balancing were summarized. The LTWRR method [3] allocates incoming traffic to the servers by assigning weights based on the latency of their respective links. Another WRR variant [4] distributes workloads by statically assigning weights according to each server's capacity. A Dynamic Weighted Random Selection (DWRS) approach [5] leveraged a single SDN controller with multithreading to dynamically manage traffic allocation. In the case of dynamic load balancing by switch migration [6], the system identifies

overloaded servers and initiates balancing when response time exceeds a predefined threshold. DAF (Distributed Adaptive and Fast) [7] is a dynamic algorithm that continuously monitors server loads to allocate traffic according to the current status of each server. A multipath forwarding strategy [8] can further improve load-balancing efficiency. In addition, a hierarchical control plane structure, featuring a super controller and several master controllers [9], implements a load notification policy to manage load balancing across multiple controllers. The received requests can be assigned to virtual machines using fuzzy logic [10]. In [11], an RR load-balancing approach was implemented using a single Floodlight controller. In [12], Dijkstra's algorithm was integrated with RR, achieving enhanced performance in fat-tree topologies. In [13], a Mininet-based setup with the POX controller was presented, where load allocation was dynamically regulated based on server availability and health status. The method in [14] enabled SDN controllers to automatically redirect traffic, ensuring sustained performance without manual intervention. In [15], the least connections algorithm was proposed, assigning traffic to the server with the fewest active connections and demonstrating better efficiency compared to RR. In [16], it was emphasized that throughput, response time, and availability could be improved by prioritizing traffic flows. Table I presents the performance benefits and constraints of RR variants.

TABLE I. PROS AND CONS OF EXISTING RR LOAD BALANCING APPROACHES

Ref.	Strengths	Limitations
[4]	Provides improved throughput and reduced transmission time compared to basic RR.	Performance degrades due to dynamic traffic, as weights are assigned based on only server capacity.
[11]	Enhances throughput and is simple to implement	Performs poorly under heterogeneous or dynamic traffic conditions.
[12]	The integration of the RR approach with Dijkstra's algorithm facilitates effective traffic distribution and path optimality for more balanced routing.	Frequent shortest path calculations increase controller overhead and may lag under rapidly changing traffic conditions.
[13]	Offers acceptable response times and is easy to implement.	Does not adjust to server load differences, limiting effectiveness in heterogeneous environments.
[14]	Effective for uniform workloads	Lack of performance guarantees in varied topologies.
[15]	Maintains performance under high request volumes without major degradation.	Least connections requires continuous monitoring of active sessions, adding operational complexity.
[16]	Prioritizes the traffic flows and then applies RR, boosting the performance for higher priority traffic.	May fail to adapt if traffic dynamics shift. Low-priority flows can be penalized even when the high-priority load is minimal.
[17]	Achieves higher throughput and improved response times compared to RR/WRR in Mininet tests	Used pox controller, which is less capable of managing traffic.

This review indicates that both the standard RR approach and its extensions typically depend on a single controller, allocating client requests to servers in a fixed cyclic order without considering the actual workload on each server. This can lead to inefficient resource utilization and overload requests on some servers while others remain underutilized. In contrast, the proposed DWRR-DC algorithm improves on this by introducing dynamic weight calculations that adapt to the real-time load on each server. Furthermore, this algorithm is designed to work with three controllers, significantly improving the scalability and overall performance of the network.

II. THE DWRR-DC ALGORITHM

The key novelty of the proposed algorithm is the improvement of the RR load balancing technique by employing three controllers and distributing incoming requests based on dynamically updated server weights. The proposed Dynamic Weighted Round Robin with Distributed Controller (DWRR-DC) mechanism aims to achieve effective traffic balancing across both servers and controllers by dynamically modifying their weights according to the observed load. Unlike the traditional static weighted RR, where weights remain unchanged, the DWRR-DC approach continually updates weights in response to variations in request handling. The proposed algorithm operates in three stages:

1. Threshold determination of requests,
2. Dynamic weight assignment,
3. Multi-controller management.

A. Threshold Determination of Requests

In the initial stage, the maximum request capacity of a single controller is determined to prevent performance degradation. This is accomplished by measuring response times with the ping command under varying request loads. The response time versus request curve is analyzed to identify the point where latency increases sharply. Two rounds of experiments are conducted to observe how the response time changes as the number of requests increases. The experimental findings indicate that system performance starts to decrease noticeably around 59 requests; therefore, the proposed method sets the threshold value at 50 requests.

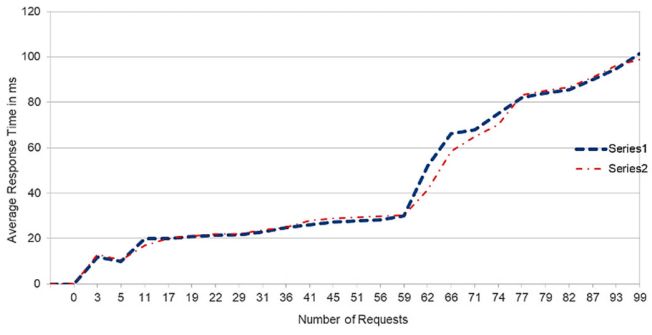

Fig. 1. Threshold response-time determination.

B. Dynamic Weight Assignment

In this phase, the server weights are updated dynamically according to the load managed by each server, as described in the following process. This approach assigns lower weights to heavily burdened servers and higher weights to lightly loaded ones, thereby maintaining balanced traffic distribution. The parameters considered are: W denotes the server weights, L denotes the server loads, n is the number of servers, WT is the total weight of n servers, and LT is the total load of n servers.

1) Step 1: Initialization

Assume n servers, each assigned an initial weight W_i. When client requests arrive, the corresponding server records its current load L_i.

2) Step 2: Compute Total Weight and Total Load

The total weight is the sum of the weights of the servers, calculated as:

$$WT = \sum_{i=1}^{n} W_i \quad (1)$$

The Total Load (LT) is the sum of all server loads, calculated as:

$$LT = \sum_{i=1}^{n} Li \quad (2)$$

These values represent the overall capacity and work done by all servers.

3) Step 3: Calculate the Ratio for Each Server

For each server, a ratio (R_s) is calculated, which compares the actual load a server is handling against the load it should handle based on its weight. The R_s for each server is given by:

$$R_s = \frac{L_i * WT}{LT * Wi} \quad (3)$$

4) Step 4: Find the Average Ratio

This value represents the average balance point among all servers, calculated as:

$$R_{Avg} = \frac{\sum_{i=1}^{n} R_S}{n} \quad (4)$$

5) Step 5: Adjust the Weights Dynamically

If $R_s > R_{Avg}$, the load is lighter and the weight needs to be increased. Otherwise, the load is heavier and the weight needs to be decreased. If $R_s = R_{Avg}$, then there is no need to change the weight. Dynamic weights are updated using:

$$W_i = W_i - \left[1 - \frac{\mathrm{R_S}}{\mathrm{R_{Avg}}}\right] \quad (5)$$

C. Multi-Controller Management

In this phase, the distribution of the load across multiple controllers is systematically handled, as detailed in Algorithm 2. In the proposed topology, three controllers (C_0, C_1, and C_2) are utilized. Initially, C_0 manages incoming requests until it reaches the specified threshold, after which, subsequent traffic is directed to C_1. Once C_1 attains its threshold, requests are forwarded to C_2. This process continues in an RR fashion among the controllers, thereby preventing any single controller from becoming a performance bottleneck.

For example, suppose a simulation of 100,000 requests with 5 clients and 3 servers. Initially, servers may have weights $W_1 = 3$, $W_2 = 4$, $W_3 = 5$. Requests are distributed proportionally: $3:4:5$. When the clients send requests to a controller, and when the number of received requests is less than or equal to 50, the received requests are forwarded to the servers based on the assigned server weights. If the number of requests reaches the threshold, the received traffic is migrated to the next controller. After a batch of requests, i.e., 1000, the loads L_1, L_2 and L_3 are measured. WT, LT, R_s, and R_{Avg} are calculated. Finally, the weights for the servers are updated using (5). The working process of the proposed algorithm is outlined in the flowchart shown in Figure 2.

```
Algorithm 1: Dynamic Weight of Servers
Inputs: Server weights (W), Server Loads
(L), Number of servers (n)
Dynamic Weight(W, L, n)
//Calculate total weight (WT) and total
load (LT)
1:  WT := 0
2:  LT := 0
3:  for i=1 to n do
4:     WT:= WT + W[i]
5:     LT := LT + L[i]
6:  end for
// calculate ratio (R_S) and Average Ratio
// (R_Avg)
7:  RS := 0
8:  for i = 1 to n do
9:     RS[i] := (L[i] * WT ) / (LT * W[i])
10:    RS:= RS + RS[i]
11: end for
12: R_Avg:= RS / n
//  update the weight of each server
13: for i = 1 to n do
14:    if R_S[i] ≠ R_Avg  then
15:    W_i:= W_i − (1 − R_S/R_Avg )
16:    end if
17: end for
18: end
```

```
Algorithm 2: Multi-Controller management
// Initialize a list of controllers with
// their IP addresses and Port numbers
//Set a Threshold Load Value
1:  Migrate Switch()
//  Migrates the switch to the next
//  available controller in an RR manner
2:  while (true) do:
3:  for each controller IP
4:    if total_load >= threshold:
5:       Migrate_switch()
6:    end if
7:  end for
8:  end
```

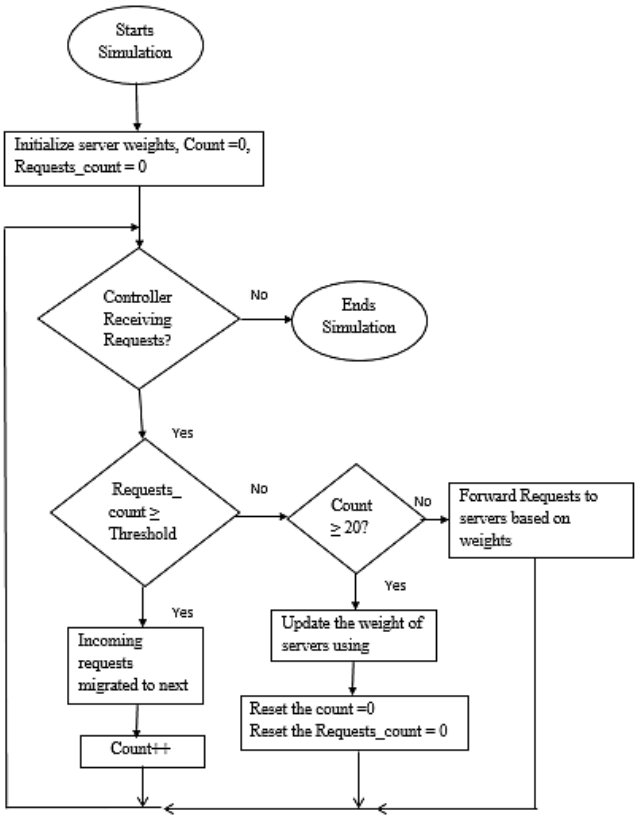

Fig. 2. Flowchart of DWRR-DC.

III. EXPERIMENTAL DESIGN

The proposed algorithm was validated using the Mininet emulation tool [18] in a topology with 5 clients and 3 servers. The load balancer architecture consists of three controllers tasked with managing the distribution of network traffic. These controllers receive requests from clients and, using the proposed algorithm, direct them to the appropriate servers. In this setup, a Ryu controller [19] is utilized, interfacing with an OpenFlow switch that connects to multiple servers via its ports. Both the servers and clients are assigned static IP addresses, while the controllers are allocated virtual IPs. Client requests are sent to this virtual IP, allowing the controller to determine the optimal server to handle each request according to the proposed load balancing strategy. The switch then modifies the destination virtual IP address in the packet to match the chosen server's IP address before forwarding the packet. Figure 2 represents the DWRR-DC setup for the simulation. The clients send the requests to controllers in an RR fashion according to Algorithm 2, and the requests from the controllers are sent to a particular server according to Algorithm 1.

Figure 3 shows the execution of the topology in Mininet. Three controllers with seven switches were configured. Hosts $h1$, $h2$, $h3$, $h4$, $h5$ were considered clients and $h6$, $h7$, $h8$ were the servers. Various traffic generation tools that are compatible with Mininet, such as curl, ping, iperf, Multi-Generator (MGEN), and Distributed Internet Traffic Generator (DITG), have been analyzed for their performance in previous studies [20]. To evaluate the proposed DWRR-DC algorithm, iperf was used to generate the necessary network traffic. Traffic from all five clients was simultaneously directed to the servers using iperf-UDP. The corresponding output of the iperf-UDP is redirected to a log file, as shown in Figure 4. Using iperf-UDP, the intervals, transfer rate, bandwidth, count of received packets, count of packet-loss, and packet-loss percentage are recorded.

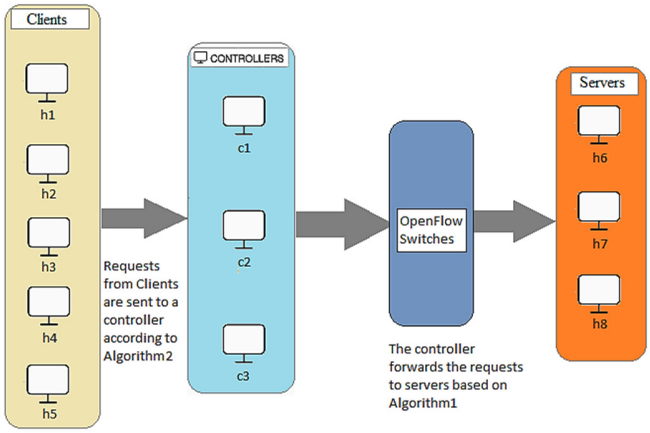

Fig. 3. DWRR-DC.

```
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(10.00Mbit) (10.00Mbit) (s1, s2) (10.00Mbit) (10.00M
) (10.00Mbit) (s3, s6) (10.00Mbit) (10.00Mbit) (s3,
it) (s5, h3) (10.00Mbit) (10.00Mbit) (s5, h4) (10.00
it 11ms delay) (10.00Mbit 11ms delay) (s7, h7) (10.0
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding controller
*** Starting controller
c0 c1 c2
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...(10.00Mbit) (10.00Mbit) (10.
) (10.00Mbit) (10.00Mbit) (10.00Mbit) (10.00Mbit) (1
 12ms delay)
*** Waiting for switches to connect
s1 s2 s3 s4 s5 s6 s7
*** Starting network
*** Starting controllers
Starting the servers in h6, h7, h8
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
```

Fig. 4. Execution of the topology.

```
1 ------------------------------------------------------------
2 Client connecting to 10.1.1.100, UDP port 5001
3 Sending 1470 byte datagrams, IPG target: 1176.00 us (kalman adjust)
4 UDP buffer size:  208 KByte (default)
5 ------------------------------------------------------------
6 [  5] local 10.1.1.1 port 48790 connected with 10.1.1.100 port 5001
7 ------------------------------------------------------------
8 Client connecting to 10.1.1.100, UDP port 5001
9 Sending 1470 byte datagrams, IPG target: 1176.00 us (kalman adjust)
10 UDP buffer size:  208 KByte (default)
11 ------------------------------------------------------------
12 [  5] local 10.1.1.2 port 48821 connected with 10.1.1.100 port 5001
13 [ ID] Interval       Transfer     Bandwidth
14 [  5]  0.0-10.1 sec  7.84 MBytes  6.52 Mbits/sec
15 [  5] Sent 5593 datagrams
16 [  5] Server Report:
17 [  5]  0.0-10.7 sec  9.94 MBytes  7.82 Mbits/sec  11.988 ms    0/ 5593 (0%)
```

Fig. 5. Output of iperf-UDP.

IV. EXPERIMENTAL RESULTS

The effectiveness of the proposed DWRR-DC algorithm was compared against five existing algorithms: RR, RS, WRR, LTWRR, and DWRS. The RR technique is a traditional and simple load-balancing approach that distributes incoming requests sequentially and evenly across all available servers. The Random Selection (RS) method assigns each request to a server chosen randomly from the available servers. The conventional RR mechanism has been refined into the WRR algorithm, where requests are allocated based on predefined weights assigned to each server. In the LTWRR algorithm, higher weights are given to servers with the lowest link delay, allowing them to handle a greater number of requests. In the DWRS strategy, the controller constantly monitors the load on each server and converts this data into a weight value as:

$$ServerWeight = 10 - (load\ at\ server/10) \quad (6)$$

This computed weight determines the probability of a server being selected to process a new request. The controller maintains a record of all server weights. When a new request arrives, it calculates the total of all weights and generates a random number (R) less than this total. A variable (V) is initialized to zero and is incremented sequentially by each server's weight. Once V exceeds R, the corresponding server is chosen, and the request is assigned to it.

All algorithms, including the proposed one, were implemented within the same experimental environment to ensure a fair and consistent comparative analysis. The evaluation considers three huge traffic levels of 10,000, 50,000, and 100,000 client requests to assess load-balancing performance. Key performance metrics include packet loss percentage and throughput.

- Throughput measures how quickly packets are sent through the network, calculated as

$$Throughput = (Dp * Ps)/Total\ simulation\ time \quad (7)$$

where Dp is the count of delivered packets and Ps is the packet size.

- Packet loss is determined using:

$$PacketLoss\% = (Dp - Rp)/Dp * 100 \quad (8)$$

where Dp is the count of packets delivered and Rp count of packets sent.

In the RR algorithm, the received requests are distributed among the servers irrespective of the server condition. In the RS algorithm, the load is distributed by randomly selecting the servers. Hence, with an increase in load, a huge packet loss is obtained in these algorithms. In LTWRR, static weights are assigned with respect to the delay identified in the link. As static weights are assigned, packet loss is greater in the case of a dynamic increase in traffic load. In WRR, the weights are assigned statically only once with respect to server capacity. Therefore, a dynamic increase in traffic can cause packet loss. In DWRS, the weights are dynamically assigned based on the current amount of requests on the servers, and the request is distributed based on the dynamic weights. Hence, packet loss is less compared to static algorithms. In the proposed DWRR-DC, three controllers are used. Figure 5 shows that the proposed algorithm offers a 25.7% increase in throughput, and Figure 6 shows that the DWRR-DC has a 12.4% decrease in average packet loss than the DWRS algorithm. Table II presents the comprehensive statistical evaluation of the performance metrics for the analyzed algorithms.

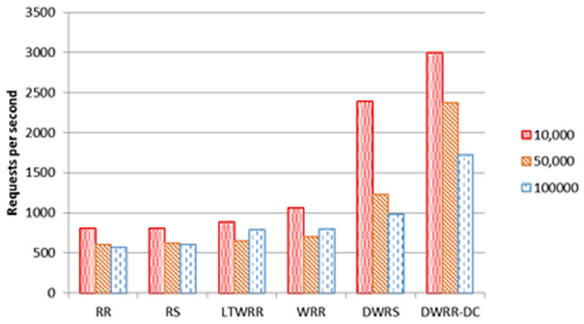

Fig. 6. Throughput comparison.

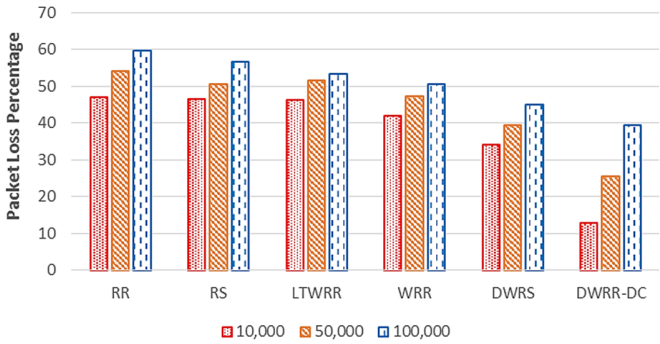

Fig. 7. Packet loss percentage comparison.

TABLE II. STATISTICAL EVALUATION OF PERFORMANCE METRICS BETWEEN PROPOSED AND EXISTING ALGORITHMS

Algorithms	Throughput (Req/sec)	Packet loss %	HTTP response time (s)	Overall Performance
RR	700–900	45–60%	7–9	Weak scalability and high losses
RS	700–850	45–58%	8–9	Unpredictable, not stable
LTWRR	800–950	45–55%	7–9	Slight improvement over RR/RS
WRR	900–1100	40–50%	5–7	Balanced but not optimal
DWRS	1500–2400	35–45%	2–4	Better throughput and low delay
DWRR-DC	2000–3000	15–40%	1–3	Good scalability, lowest loss, fast response

To validate the functionality of the developed load balancing algorithm, HTTP requests were generated and sent to the servers, and the corresponding response times were measured and analyzed. This evaluation was performed for both the proposed and existing load balancing algorithms to enable a comprehensive performance comparison. Figure 8 shows that the proposed algorithm achieves a shorter response time than the other algorithms considered. Additionally, the number of controllers deployed within the network topology significantly influences overall system performance. An insufficient number of controllers may lead to congestion and delayed response times, while an excessive number may introduce unnecessary complexity and resource overhead. Hence, identifying the optimal controller count is essential to achieve a well-balanced environment. Therefore, to assess the impact of controller count on network efficiency for the proposed DWRR-DC algorithm, the round-trip times for varying numbers of controllers were analyzed. This helped identify the most suitable number of controllers to ensure low latency and efficient handling of traffic in the network. Figure 8 shows that the optimal count of controllers for the DWRR-DC algorithm is 1-4. Performance decreases when using more than four controllers.

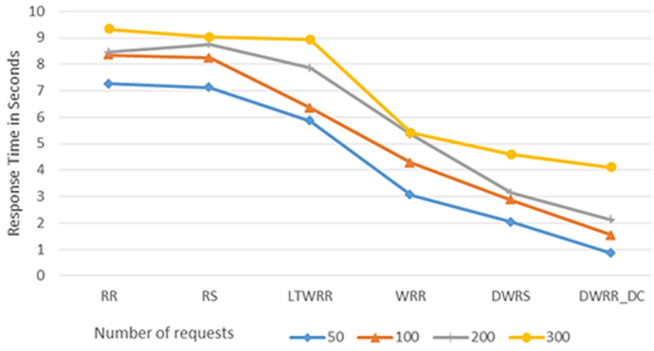

Fig. 8. Validation of proposed DWRR-DC.

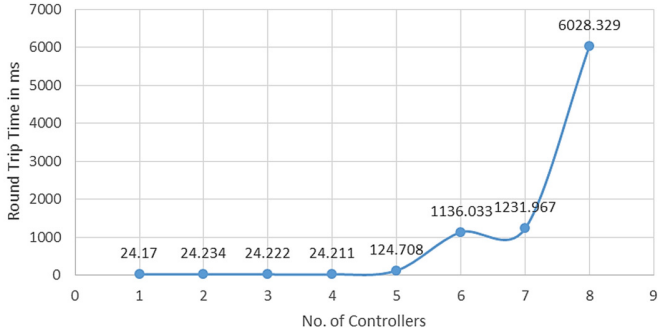

Fig. 9. Optimal number of controllers for DWRR-DC.

V. CONCLUSION

Conventional RR and its improved variants face challenges in adapting to dynamic traffic, as they depend on fixed server weights and a single centralized controller. To address these issues, the DWRR-DC algorithm is introduced as an advanced extension of the RR scheme. This approach integrates three controllers and assigns requests to servers based on dynamically updated weights that reflect real-time server load. The algorithm is designed to efficiently balance traffic between controllers. For fair comparison, the proposed strategy and conventional ones were tested under identical client and server conditions. The experimental findings reveal that DWRR-DC improves throughput by 25.7% and reduces packet loss by 12.4% compared to the DWRS method. Additional experiments with HTTP workloads show that DWRR-DC provides faster response times than other algorithms. Moreover, round-trip time analysis suggests that deploying between one to four controllers yields optimal efficiency, while performance declines with more than five controllers. Results clearly indicate that the proposed DWRR-DC achieves significant improvements compared to prior algorithms, excelling in several performance dimensions.

VI. DWRR-DC: LIMITATIONS AND POTENTIAL FUTURE ENHANCEMENTS

In the DWRR-DC algorithm, the threshold value of the response time is derived manually from the plotted graph, rather than being obtained through an automated process. This can be identified by AI algorithms in future research. The method presumes that all servers are homogeneous, an assumption that may not reflect real-world conditions. The algorithm can be extended to effectively handle heterogeneous server configurations. The server weight assignment considers only the current load on the servers, while other factors, such as link delay, server capacity, and bandwidth, could also be incorporated for more accurate allocation. Machine Learning (ML) algorithms can be integrated into the load-balancing framework to intelligently and dynamically adjust the weight values assigned to different servers or paths based on real-time network traffic conditions. By continuously analyzing parameters such as traffic volume, latency, packet loss, and server utilization, ML models can learn patterns and make informed decisions to optimize traffic distribution. This adaptive approach can not only enhance performance and resource efficiency but can also improve the responsiveness of the load balancer to sudden changes or anomalies in the network.

REFERENCES

[1] R. Sharma and H. Reddy, "Effect of Load Balancer on Software-Defined Networking (SDN) based Cloud," in *2019 IEEE 16th India Council International Conference (INDICON)*, Rajkot, India, Dec. 2019, pp. 1–4, https://doi.org/10.1109/INDICON47234.2019.9030327.

[2] M. Shona and R. Sharma, "Implementation and Comparative Analysis of Static and Dynamic Load Balancing Algorithms in SDN," in *2023 International Conference for Advancement in Technology (ICONAT)*, Goa, India, Jan. 2023, pp. 1–7, https://doi.org/10.1109/ICONAT57137.2023.10080430.

[3] K. Kaur, S. Kaur, and V. Gupta, "Least Time Based Weighted Load Balancing Using Software Defined Networking," in *Advances in Computing and Data Sciences*, Ghaziabad, India, 2017, pp. 309–314, https://doi.org/10.1007/978-981-10-5427-3_33.

[4] S. B. Vyakaranal and J. G. Naragund, "Weighted Round-Robin Load Balancing Algorithm for Software-Defined Network," in *Emerging Research in Electronics, Computer Science and Technology*, 2019, pp. 375–387, https://doi.org/10.1007/978-981-13-5802-9_35.

[5] M. L. Chiang, H. S. Cheng, H. Y. Liu, and C. Y. Chiang, "SDN-based server clusters with dynamic load balancing and performance

improvement," *Cluster Computing*, vol. 24, no. 1, pp. 537–558, Mar. 2021, https://doi.org/10.1007/s10586-020-03135-w.

[6] J. Cui, Q. Lu, H. Zhong, M. Tian, and L. Liu, "A Load-Balancing Mechanism for Distributed SDN Control Plane Using Response Time," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1197–1206, Sep. 2018, https://doi.org/10.1109/TNSM.2018.2876369.

[7] H. Sufiev, Y. Haddad, L. Barenboim, and J. Soler, "Dynamic SDN Controller Load Balancing," *Future Internet*, vol. 11, no. 3, Mar. 2019, Art. no. 75, https://doi.org/10.3390/fi11030075.

[8] Y. C. Wang, Y. D. Lin, and G. Y. Chang, "SDN-based dynamic multipath forwarding for inter–data center networking," *International Journal of Communication Systems*, vol. 32, no. 1, 2019, Art. no. e3843, https://doi.org/10.1002/dac.3843.

[9] J. Yu, Y. Wang, K. Pei, S. Zhang, and J. Li, "A load balancing mechanism for multiple SDN controllers based on load informing strategy," in *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Kanazawa, Japan, Oct. 2016, pp. 1–4, https://doi.org/10.1109/APNOMS.2016.7737283.

[10] S. F. Issawi, A. A. Halees, and M. Radi, "An Efficient Adaptive Load Balancing Algorithm for Cloud Computing Under Bursty Workloads," *Engineering, Technology & Applied Science Research*, vol. 5, no. 3, pp. 795–800, Jun. 2015, https://doi.org/10.48084/etasr.554.

[11] Y. A. H. Omer, A. B. A. Mustafa, and A. G. Abdalla, "Performance Analysis of Round Robin Load Balancing in SDN," in *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, Khartoum, Sudan, Feb. 2021, pp. 1–5, https://doi.org/10.1109/ICCCEEE49695.2021.9429662.

[12] V. Kumar, S. Jangir, and D. G. Patanvariya, "Traffic Load Balancing in SDN Using Round-Robin and Dijkstra Based Methodology," in *2022 International Conference for Advancement in Technology (ICONAT)*, Goa, India, Jan. 2022, pp. 1–4, https://doi.org/10.1109/ICONAT53423.2022.9725862.

[13] I. T. Singh, T. R. Singh, and T. Sinam, "Server Load Balancing with Round Robin Technique in SDN," in *2022 International Conference on Decision Aid Sciences and Applications (DASA)*, Chiangrai, Thailand, Mar. 2022, pp. 503–505, https://doi.org/10.1109/DASA54658.2022.9765287.

[14] M. D. Tache (Ungureanu), O. Păscuțoiu, and E. Borcoci, "Optimization Algorithms in SDN: Routing, Load Balancing, and Delay Optimization," *Applied Sciences*, vol. 14, no. 14, Jan. 2024, Art. no. 5967, https://doi.org/10.3390/app14145967.

[15] C. Wijaya, R. Wiryasaputra, C. Y. Huang, J. Tanato, and C. T. Yang, "Load Balancing Algorithm in a Software-Defined Network Environment with Round Robin and Least Connections," in *Smart Grid and Internet of Things*, TaiChung, Taiwan, 2024, pp. 148–157, https://doi.org/10.1007/978-3-031-55976-1_15.

[16] N. Joshi and D. Gupta, "Application Layer Load Balancing in Software Defined Networking Using Priority Based Round Robin Scheduling Algorithm," *Wireless Personal Communications*, vol. 136, no. 2, pp. 759–772, May 2024, https://doi.org/10.1007/s11277-024-11273-2.

[17] B. Manasa 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, pp. 304–310, May 2023, https://doi.org/10.17762/ijritcc.v11i5.6618.

[18] S. Khan, A. Akram, H. Alsaif, and M. Usman, "Emulating Software Defined Network using Mininet-ns3-WIFI Integration for Wireless Networks," *Wireless Personal Communications*, vol. 118, no. 1, pp. 75–92, May 2021, https://doi.org/10.1007/s11277-020-08002-w.

[19] A. T. Albu-Salih, "Performance Evaluation of Ryu Controller in Software Defined Networks," *Journal of Al-Qadisiyah for Computer Science and Mathematics*, vol. 14, no. 1, Feb. 2022, https://doi.org/10.29304/jqcm.2022.14.1.879.

[20] T. Khudhair and O. Athab, "Recent Tools of Software-Defined Networking Traffic Generation and Data Collection," *Al-Khwarizmi Engineering Journal*, vol. 21, no. 2, pp. 93–105, Jun. 2025, https://doi.org/10.22153/kej.2025.06.002.