IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Journal Européen des Systèmes Automatisés
Vol. 59, No. 5, May, 2026, pp. 1461-1476
Journal homepage: http://iieta.org/journals/jesa

A Queueing Theory-Based Dynamic Load Balancing Algorithm for Optimizing Software-Defined Networking Performance in Homogeneous and Heterogeneous Environments

Maghrib Abidalreda Maky Alrammahi[1*], Mohanad Yahya Al-hamami[1], Ali Mohammed Taher[2]

[1] Information Technology Research and Development Centre, University of Kufa, Najaf 54001, Iraq
[2] Department of Mathematics, Faculty of Basic Education, University of Kufa, Najaf 54001, Iraq

Corresponding Author Email: maghrib.alramahi@uokufa.edu.iq

https://doi.org/10.18280/jesa.590522

Received: 6 March 2026
Revised: 3 May 2026
Accepted: 19 May 2026
Available online: 31 May 2026

Keywords:
Software-Defined Networking, load balancing, least Control Process Unit, least Random Access Memory, Least Connection, Python OpenFlow eXtensible, Mininet, queueing theory

ABSTRACT

Software-Defined Networking (SDN) is one of the most prominent modern technologies in network management because it separates the control plane from the data plane, enabling improved performance and more dynamic resource management. Load balancing among servers is a major challenge in SDN networks because it directly affects service quality and network efficiency. To address this issue, a dynamic load balancing with utilizes queueing theory (DLBQT) algorithm is proposed to distribute traffic among servers with the aim of improving network performance. Simulations were conducted in Mininet with a Python OpenFlow eXtensible (POX) SDN controller using real traffic traces from Facebook data centers. Three representative load levels were evaluated: 1,000, 100,000, and 1,000,000 requests. For each algorithm and environment (homogeneous and heterogeneous), the experiments were repeated three times under identical request sequences, and the performance metrics were averaged. The reported percentage gains in packet loss, response time, and throughput are computed using a standard percentage-improvement metric over the aggregated results across the three load levels. Within the evaluated SDN setup, DLBQT reduces packet loss and response time by up to 37.5% and 8.61%, respectively, and improves throughput by 6.61% in the homogeneous environment, while in the heterogeneous environment, the corresponding improvements reach 39.2%, 11.4%, and 18.4%, respectively.

1. INTRODUCTION

Software-Defined Networking (SDN) is a recent development in network management, providing more flexibility since it decouples the data plane and the control plane. This division allows programmability in the control center, optimum utilization of resources, and better performance of the network [1]. The SDN architecture comprises three main layers. The application layer defines the network policies and services. The control layer contains the network controller, which processes traffic and decides how to route flows. The data plane consists of forwarding devices that route traffic according to the controller's instructions [2].

Despite these benefits, one of the most significant SDN aspects is load balancing [3]. The distribution of loads across servers should be done in an efficient manner to minimize congestion and the loss of packets, to enhance response times, and to enhance the rate of data transfer across servers [4]. Network environments are either homogeneous or heterogeneous [5].

In a homogeneous environment, servers have similar hardware specifications, such as Control Process Unit (CPU) and Random Access Memory (RAM) capacity. In contrast, a heterogeneous environment contains servers with different CPU and memory capacities [6]. Heterogeneous environments are closer to real-world deployments, where resource variability helps accommodate diverse workloads and optimize utilization.

The load balancing techniques can be classified as both static and dynamic [7]. In static algorithms, the decision parameters are fixed during execution and do not change in response to real-time network conditions. As a result, static schemes cannot adapt to dynamic traffic and often lead to inefficient resource usage [8]. Conversely, dynamic algorithms can be configured to use the current measurements of server load, response time, and data traffic status to make decisions and therefore can be more adaptive and offer a higher degree of performance [9]. It has been shown previously that the dynamic approaches are more effective, especially when operating within heterogeneous environments.

In this paper, proposes a load balancing algorithm, namely dynamic load balancing with utilizes queueing theory (DLBQT) to optimize load distribution in SDN networks. The algorithm is designed to achieve more accurate decision-making and a balance between performance and decision complications.

This research included the following key contributions:

- Proposal of a dynamic DLBQT algorithm based on the queueing theory concept to improve load balancing in SDN networks.

- Evaluation of the algorithm in homogeneous and heterogeneous environments using Mininet and the Python OpenFlow eXtensible (POX) controller.

- Comparing performance with the Least CPU (LCPU), Least Connection (LC), and Least RAM (LRAM) algorithms using packet loss, response time, and throughput.

- Providing a practical solution to improve the performance of dynamic networks and reduce congestion.

The remainder of this paper is organized as follows. Section 2 reviews previous studies in the field of load balancing. Section 3 provides an explanation of performance metrics and the improvement method. Section 4 explains the proposed methodology, flowchart, and general diagram. Section 5 explains and describes the proposed network and discusses the results. Section 6 discusses and analyses the work achieved for the proposed algorithm. Finally, Section 7 presents the conclusion and future work.

The main abbreviations used throughout this paper are summarized in Table 1 for ease of reference.

Table 1. Abbreviations used in the paper

Abbrevia	Description
SDN	Software-Defined Networking
DLBQT	Dynamic Load Balancing based on Queueing Theory
LC	Least Connection
LCPU	Least CPU
LRAM	Least RAM
QoS	Quality of Service
CI	Confidence Interval
SD	Standard Deviation
MM1	M/M/1 queueing model
R	Number of experimental runs per configuration

2. RELATED WORK

This section reviews previous studies on dynamic load balancing in SDN networks, their key techniques, results, and weaknesses.

In the study [10], the Adaptive Lowest Load Ratio (ALLR) algorithm was suggested, and it approximates server load according to the monitoring and adaptive computations. It was also compared with the Dynamic Least Connection (DLC) and Dynamic Least Bandwidth (DLB) algorithms and had better response time (13.37%) and throughput (8%) and low CPU consumption. This method helps with decentralized decision-making, but it is based upon periodic updates, and this can lead to problems when the load is high. In the study [11], a dynamic weighting method of the Weighted Round Robin (WRR) algorithm was suggested that requires fewer communications. This method balances the weights based on the actual data rate that enhances the performance of load balancing and resource consumption. Findings showed an increase in the response time and packet loss, although it is not easy to compute the optimal weights because of unexpected data traffic variations, which can lead to inefficiencies in the algorithm's performance and affect overall network reliability. In the study [12], the Load Balancing based on Resource Utilization (LBORU) algorithm was developed on a hybrid SDN network, using different measurements, including CPU, I/O, and network speed. The algorithm achieved high load balancing while maintaining a CPU variance of 10%. Even though it is a better use of resources and less latency, it is more complex and has not been demonstrated to be effective in practice, particularly in real-world scenarios where varying workloads and network conditions can impact performance. In the study [13], a dynamic load balancing algorithm was implemented in SDN data centers and executed in Mininet and Floodlight controllers. This method diverts the streams of data to less overloaded routes, enhancing throughput (as much as 50 percent) and reply time. Nevertheless, the routing decision-making and controller load can be higher when data updates are high in the network, which may lead to increased latency and reduced overall performance of the data center operations. In the study [14], Enhanced_Conn, which is a combination of dynamic and static factors, was used as the extended LC algorithm. It showed better load balancing and reduced response times in comparison to the RR and LC algorithms. The need to run the data process continuously complicates its computation, even though it is efficient, particularly when compared to the static nature of the RR and LC algorithms, which do not require continuous data processing.

In this study [15], four dynamic load balancing algorithms (namely LRAM, LCPU, Least Connection-Least CPU-RAM (LCLCPURAM), and Least CPU-RAM (LCPURAM) were proposed and tested to compare them with the famous LC algorithm in a scenario that includes the servers with different capacities. The experiments were performed with Mininet, an OpenFlow switch, and a POX controller. The findings revealed that the LCLCPURAM algorithm would have a great latency and waiting time improvement, although the LC algorithm had better outcomes in service time. In general, the study proved the efficacy of dynamic approaches to the enhancement of performance in a heterogeneous environment.

In this study [16], a deep learning-based multi-label classification method with an adaptive load balancing mechanism was proposed for distributed computing systems. The method is based on a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) architecture for feature extraction and traffic classification, and then a Deep Reinforcement Learning (DRL) agent is used to optimize the load distribution dynamically. The proposed model showed remarkable improvements; the average throughput rose from 85-115 requests/s to 120-150 requests/s, and the average latency reduced from 95-120 ms to 65-95 ms. It has been observed, however, that the model has some misclassifications for some traffic types (such as video streaming), and the combination of several deep learning components makes the model more complex, potentially restricting its use in real time in resource-limited SDN environments.

In this study [17], DeepBalance, a DRL framework for dynamic load balancing in SDNs, was proposed. A Deep Q-Network (DQN) agent is continuously observing real-time network states, such as link utilization, queue sizes, and flow statistics, and learns optimal routing policies using a multi-objective reward function, which consists of three components: load distribution (60%), congestion avoidance (30%), and throughput-to-latency efficiency (10%). Experimental results demonstrated that the variance of link utilization was reduced by 37%, the throughput was increased by 28%, and the latency was decreased by 42% compared to shortest-path routing. Even with its excellent performance, DeepBalance needs lots of training time and computation power, and the initial exploration period may result in sub-

optimal early routing decisions. Additionally, the high state-action space complexity makes it challenging to scale up to hundreds of nodes.

In this study [18], a decentralized Quality of Service (QoS)-aware load balancer for edge and cloud environments was proposed, namely, QEdgeProxy. The load balancing problem is stated as a Multi-Player Multi-Armed Bandit (MP-MAB) problem with heterogeneous per-client QoS rewards. The load balancers work independently and use Kernel Density Estimation (KDE) to estimate the QoS success probability of each service instance while also using an adaptive exploration mechanism to cope with performance changes and non-stationary traffic patterns. The system outperformed the other two baselines (proximity and reinforcement learning) in terms of per-client QoS satisfaction. This approach is for compute-continuum and IoT environments and is decentralized, unlike classical SDN server-side load balancing, and the bandit-based model is not analytically guaranteed like a queueing-theoretic formulation.

In summary, the dynamic load balancing algorithms presented in the literature (e.g., ALLR, WRR with dynamic weights, LBORU, Enhanced_Conn, and the LCPU/LRAM family) have many merits compared to static algorithms but still have some disadvantages in real SDN scenarios. These have the following limitations: (i) they are less accurate when sudden traffic bursts occur or when server capacities vary (single or loosely coupled metrics, such as CPU or connections only); (ii) they have high control overhead and decision complexity because they need frequent state updates; and (iii) they have not been fully validated with realistic traffic traces in both homogeneous and heterogeneous environments.

In particular, the LCPU/LRAM family of algorithms introduced by reference [15] in 2026 represents a recent state-of-the-art baseline for dynamic server-side load balancing in SDN, demonstrating clear advantages over the classic LC strategy in heterogeneous environments. In this work, these drawbacks are explicitly addressed and empirically demonstrated by comparing the proposed DLBQT algorithm, which is based on queueing theory, with three widely used dynamic algorithms (LCPU, LC, and LRAM) under identical network topologies, traffic loads, and performance metrics (packet loss, response time, and throughput), as detailed in Sections 5 and 6. It should be noted that this comparison is conducted in terms of the algorithms' mechanisms of operation, implementation, and efficiency in load distribution within the specific simulation environment of this research, without relying on the quantitative results, network design, or operating conditions reported in previous studies. Instead, the study is based on the network design proposed in this work, evaluated under two distinct environments (homogeneous and heterogeneous) and using a real-world dataset adopted for generating packets and requests. This ensures that the comparison provides a current and realistic assessment of the DLBQT algorithm's ability to enhance SDN performance under practical network conditions.

The proposed DLBQT algorithm explicitly models each server as a queueing system, where the arrival rate (λ), service rate (μ), utilization (ρ), and average waiting time are continuously estimated in real time. Based on these parameters, the controller computes a weight for each server and always selects the server with the lowest weight for incoming requests, while assigning a high penalty weight when the utilization approaches saturation. This analytically grounded decision mechanism allows DLBQT to proactively avoid congestion and to distribute load more accurately than heuristics that rely only on CPU usage, RAM, or connection counts.

The recent state-of-the-art studies in SDN load balancing have continued to seek dynamic and hybrid schemes. Some of the publications provided better dynamic weighted round robin and hybrid dynamic-static algorithms that combine heuristic measurements and metaheuristics or intelligent controllers to enhance response time and throughput in SDN systems. Multi-parameter decision algorithms and load balancing at the controller level have been proposed in other work. In large-scale SDN deployments, researchers have used metrics such as CPU, memory, connection utilization, and QoS limitations to reallocate load across controllers or servers. Recent surveys and reviews (2018–2026) have also noted a trend towards multi-metric, QoS-conscious, and AI-assisted load balancing solutions, as well as the importance of accurate traffic modeling and realistic testing environments. Nevertheless, most of these methods still depend on empirically tuned metrics or heuristic combinations rather than an analytically founded queueing model to make decisions. They are often evaluated under limited traffic conditions or without a direct comparison between homogeneous and heterogeneous settings. In contrast to the above AI-driven and QoS-aware approaches, which rely on empirically trained models, heuristic reward functions, or bandit-based probabilistic policies, the proposed DLBQT algorithm provides an analytically grounded decision mechanism based on M/M/1 queueing theory. This allows for interpretable, closed-form weight computation in real time without requiring training data or exploration phases, while achieving competitive performance improvements in both homogeneous and heterogeneous SDN environments.

3. PERFORMANCE METRICS AND IMPROVEMENT METHOD

This section describes the definitions of performance requirements in the measurement of efficiency in the network, mathematic approaches used in calculations, and optimization of results. Furthermore, the transmission of packets and requests is simulated in the simulation environment using real-world data, which enables the evaluation of the proposed algorithm to become more relevant and real. Network analysis identifies three types of incoming requests (small, medium, and large) based on repetition and data volume [19]. Small requests are those that have a low arrival rate, which is usually less than 1,000 requests per second, and they can add only a small increment to the system load. Medium requests are between 1,000 and 499,999 requests per second and take a huge share in the network load. Large requests are defined as those with arrival rates of 500,000 requests per second or more and can be up to 1,000,000 requests per second or more. This categorization makes sure that all the possible request rates are covered without gaps or overlap and represent the various load levels used in the experiments (1,000, 100,000, and 1,000,000 requests). This classification is useful in distributing resources, enhancing performance, and minimizing congestion.

3.1 Percentage improvement

The given metric holds the utmost importance during the

evaluation, as the given ratio is used to compare the suggested DLBQT algorithm to the rest of the load balancing algorithms. These calculations allow proving the high level of the proposed algorithm performance compared with other algorithms [20]. The comparison is made by the sum of the values at all the test load levels, which are 1,000, 100,000, and 1,000,000 requests. The sum of the values (results) obtained at each load level is calculated as a total value for each algorithm. The average is then calculated for the proposed algorithm. For parameters where lower values are better (e.g., packet loss), the formula is defined in Eq. (1):

$$PI = \frac{Original\ Number\ \ - New\ Number}{Original\ Number} \times 100 \qquad (1)$$

where,

Original Number: Value recorded by the other algorithm.

New Number: Value recorded by the DLBQT algorithm.

In this paper, the percentage improvement values reported in the abstract and in the result tables correspond to this metric applied to the aggregated performance across the three evaluated load levels (1,000, 100,000, and 1,000,000 requests) for each pair of algorithms.

This technique is widely used in network performance analysis due to its ease of measurement and simplicity; it improves performance by aggregating the outputs of different algorithms under various load conditions without the need for statistical normalization or weighting.

3.2 Real-world datasets

In this paper, real-world datasets were used to test the results in order to obtain a realistic and accurate evaluation of the proposed algorithm. The evaluation was conducted using a research paper that collected network traffic statistics from Facebook data centers [21]. To examine traffic patterns in an SDN load balancing simulation, the data was separated into 16 real-world-size text files. Light to heavy traffic loads, the file sizes were varied (1 KB to 6.8 MB) to enable a thorough test of network behavior with different loads of traffic. The 1 KB and 2 KB files represent light traffic, while the larger files, ranging from 196 KB to 6,836 KB, are classified as heavy traffic conditions [22]. This distribution allows the testing of the suggested algorithm on diverse real-life network scenarios.

3.3 Performance metrics

These are the most significant fundamental measures employed to determine the overall network performance, and the main focus is on the calculation of packet loss, response time, and throughput.

A) Packet Loss (PL): It is the loss of data packets sent across the network because of congestion in the network, loss of packets during transmission, or interruption of connections. The result of a high packet loss rate is delays in execution and response time, resulting in poor network performance and QoS [23]. Optimal performance is attained when the loss of packets is low. This metric is mathematically expressed as follows in Eq. (2) [24]:

$$PL\ (\%) = \frac{\mathrm{N_{lost}}}{\mathrm{N}_{sent}} \times\ 100 \qquad (2)$$

where,

PL (%): Packet Loss.

N_{lost}: Number of lost packets.

N_{sent}: Total number of packets sent

B) Response Time (RT): This is the amount of time it takes for a system to respond to a request in second. This time begins when the request is sent and ends when it is fully received. This metric helps measure system performance and the user experience [25]. Low response times indicate more responsive systems [26]. The mathematical formula for this metric in Eq. (3) [27]:

$$Response\ Time\ =\ Tend\ -\ T\ Start \qquad (3)$$

where,

RT: It takes seconds to respond.

T_{end}: The time the service ends or the response is received.

T_{Start}: The time the request or transmission begins.

C) Throughput (TH): Refers to the actual rate at which data is transferred over a given period of time across the network as (bits/s), the higher the throughput, the better the performance [27]. The mathematical formula for this metric in Eq. (4) [28]:

$$Throughput = \frac{Total\ Data\ Transferred\ (bits)}{Total\ Time\ (s)} \qquad (4)$$

where,

Throughput (bits/s): The rate at which data is transferred through a system or network, in bits per second.

Total Data Transferred (bits): The total amount of data transferred during the measured period, in bits.

Total Time (s): In seconds, the total time taken to transfer that data.

4. PROPOSED METHODOLOGY

This section explains the proposed dynamic model for load balancing in SDN environments. It begins by clarifying the general design framework associated with SDN, then proposes the DLBQT model and its flowchart, which illustrates the steps and mechanism of operation, in addition to the proposed methodology and pseudocode for the model.

4.1 Proposed model

The proposed model for dynamic load balancing in SDN, shown in Figure 1 consists of three layers: the application layer, the control layer, and the data layer. The application layer refers to the multiple users or requests that will be sent, as previously mentioned, in three types. The control layer, on the other hand, represents the SDN controller integrated with the proposed load balancing algorithm at the core of the architecture, which serves as the central hub for policy formulation and the determination of the proposed algorithms for application and load distribution. Using the concept of queueing theory, the final layer is the data unit, which operates dynamically by selecting the server requested by each server in the network based on a set of mathematical calculations.

The schematic illustrates the proposed Dynamic Load Balancing based on Queueing Theory (DLBQT) in an SDN network, which aims to make real-time decisions, as shown in Figure 2. The term “multiple requests” refers to the transmission of several requests from the application layer to

the SDN controller, which incorporates the proposed DLBQT model. Through this model, decisions are made to efficiently distribute data traffic among available servers at the data plane, leveraging the real-time monitoring capability built into the controller. This controller uses the arrival rate (λ) and service rate (μ), which are used to determine the server used (p) and the average waiting time (W). Using these parameters, the controller can make the appropriate decision and dynamically reroute data traffic away from congested paths to optimize server load. With this configuration, dynamic load balancing can be performed proactively by analyzing data based on the above parameters. This ensures network responsiveness, reducing latency and improving resource utilization across widely distributed server networks.

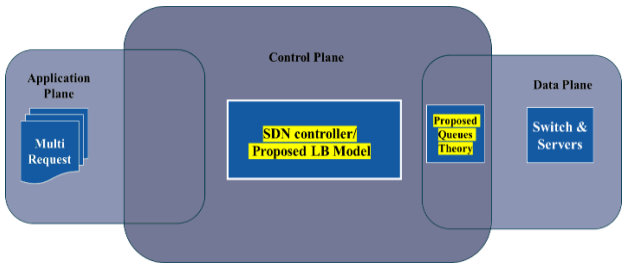

Figure 1. The diagram illustrates the architectural design of the proposed model

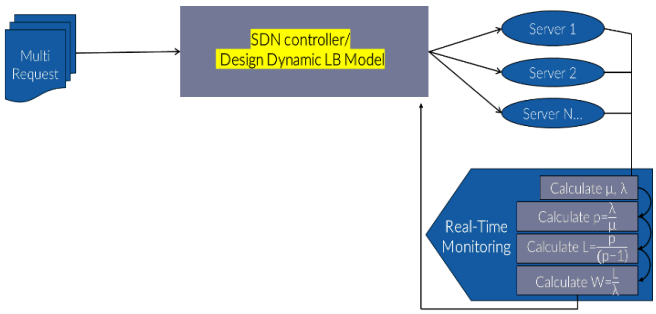

Figure 2. Diagram proposed algorithm dynamic load balancing with utilizes queueing theory (DLBQT)

4.2 Methodology

The proposed DLBQT algorithm can be summarized at four levels as follows: The SDN controller maintains a continuous record of incoming requests and the service rate for each server in the first step. Secondly, it calculates the utilization and the corresponding queueing theory weight for each server. Third, it chooses the server with the lowest weight for every new request but gives a very high weight to saturated servers. Lastly, the controller updates the statistics after every time window and continues the cycle for the next requests. This conceptual perspective is summarized in a flowchart (Figure 3) and in Table 2.

Figure 3 illustrates a simplified conceptual flow of the DLBQT algorithm. Incoming requests are first observed at the controller, which updates the arrival and service rates for each server over a sliding time window. From these measurements, the utilization and weight of each server are computed using the M/M/1 queueing model. The controller then selects the server with the minimum weight for the next request, while assigning a large penalty weight to servers whose utilization is close to saturation. After the request is processed, the QoS metrics are updated and the process repeats. Table 3 summarizes the main functional modules of the DLBQT controller and clarifies how the parameters λ, μ, ρ, and W are computed and used in the decision process.

Table 3 summarizes the main parameters used in the proposed DLBQT algorithm and clarifies their meanings and units. These parameters are used by the SDN controller to estimate server load conditions and support the routing decision process in real time.

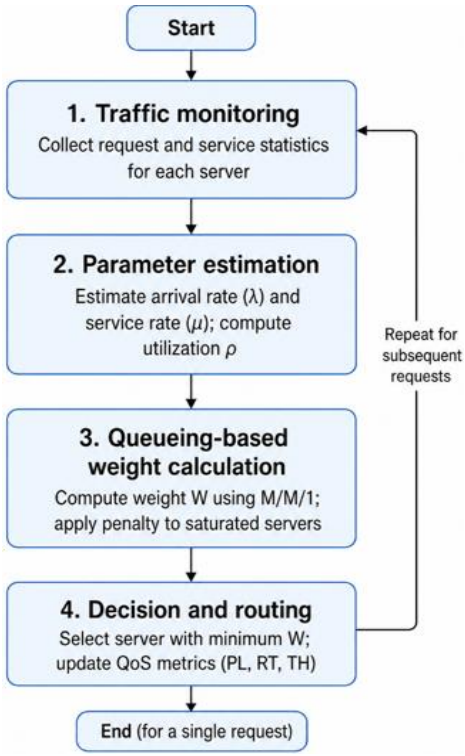

Figure 3. Simplified conceptual flow of the proposed dynamic load balancing that utilizes the queueing theory (DLBQT) algorithm

Table 2. Main modules and parameters of the dynamic load balancing with utilizes the queueing theory (DLBQT) controller

Module	Input Data	Main Computation	Output / Role
Traffic monitoring	Packet/flow counters per server, time	Compute arrival rate λ and service rate μ	Updated λ and μ for each server
Utilization estimator	λ, μ	Compute utilization ρ = λ/μ	ρ for each server
Weight calculator	P, queueing model (M/M/1)	Compute weight W, apply penalty if ρ ≈ 1	W for each server
Decision and routing	W for all servers	Select server with minimum W, break ties deterministically	Server choice for each incoming request
QoS statistics collector	Completed requests, timestamps, packet counts	Compute packet loss, response time, throughput	Performance metrics for evaluation

The DLBQT model was developed to distribute network traffic efficiently among servers by calculating key parameters based on queueing theory, where the final result depends on

the weight of each server (W) and the selection of the server with the lowest weight, which determines the routing decision. This technique aims to optimize resource utilization to achieve optimal and balanced performance. The DLBQT model continuously monitors task metrics in real time.

Table 3. Main symbols and units used in the dynamic load balancing with utilizes queueing theory (DLBQT) algorithm

Symbol	Meaning	Unit
λ	Arrival rate of incoming requests/tasks to a server	requests/s
μ	Service rate of completed requests/ tasks at a server	requests/s
ρ	Server utilization, computed as λ/μ	dimensionless
W	Estimated average time a request spends in the system	s
Twin	Observation window length	s

It relies on two basic parameters, both of which are considered inputs calculated in real time for each request. The first is the arrival rate (λ), which is the rate at which requests or tasks arrive at the system per unit of time, and second, the service rate (μ), which is the number of requests or tasks the system can process or serve per unit of time. These are mathematically formulated in Eqs. (5) and (6):

$$\lambda = \frac{Number\ of\ incoming\ requests}{Elapsed\ time} \tag{5}$$

$$\mu = \frac{Number\ of\ completed\ requests}{Elapsed\ time} \tag{6}$$

where, λ is the arrival rate and μ is the service rate are measured in requests per second (requests/s).

Each server in the proposed paradigm is modeled as an M/M/1 queueing system. Our Mininet configuration does not strictly enforce a Poisson arrival process; rather, the traffic generator generates a huge number of separate flows with randomized inter-arrival periods and file sizes. However, the aggregate behavior of many such separate random flows has been commonly modeled by M/M/1-type assumptions in networking and SDN studies. Under this approximation, the server utilization is ρ, the mathematical expression is shown in Eq. (7):

$$\rho = \lambda / \mu \tag{7}$$

where, ρ is the server utilization ratio and is dimensionless.

Finally, the weight (W) is calculated as the average time a request spends in the system, based on the utilization rates of each server and its service. The server selected to process incoming requests or tasks is determined by the lowest weight (W) value, according to the mathematical formula shown in Eq. (8):

$$W = P / (\mu * (1 - P)) \tag{8}$$

where, W denotes the estimated average system time in seconds (s).

This amount is used as the queueing-theoretic routing criterion in DLBQT: at each decision period, the controller chooses the server with the smallest estimated W, thus minimizing the predicted time a request spends in the system. Although actual SDN traffic may not conform to the idealized Poisson and exponential assumptions, the M/M/1 based models can nonetheless give valuable approximations and design guides for SDN and other networking systems. The randomized traffic patterns in our simulation environment lead to aggregate server loads that are well approximated by this M/M/1 approximation, and therefore the expression for W is not simply an intuitive heuristic, but a theoretically grounded approximation of the expected system time in the simulated SDN setting.

4.3 Online estimation of queueing parameters

In the implementation, each server keeps two counters: one for newly arrived requests and one for completed requests. At any time when the controller needs to take a routing decision, it looks at the time elapsed since the last update and uses this period as an observation window. This window is never shorter than one second, even if decisions are taken more frequently.

For each server, the algorithm stores a short history of the last ten observation windows. At every decision step, it computes:

- the average number of arrivals over the last ten windows,
- the average number of completions over the last ten windows,
- the arrival rate λ as "average arrivals ÷ window length",
- the service rate μ as "average completions ÷ window length", with a small lower bound on μ to avoid division by zero,
- the utilization P as the ratio between λ and μ.

Using these quantities, the weight W is calculated from the M/M/1 queueing model, so that W represents the estimated average time a request spends in the system (waiting plus service). After each update, the per-window counters are reset and the start time of the next window is set to the current time. This procedure smooths out short-term fluctuations and bases the decision on recent load history rather than on a single instantaneous measurement.

4.4 Routing, tie-breaking, and penalty policy

Each routing decision involves the controller updating the values of λ, μ, P, and W of all live servers as indicated above. The new request is then sent to the server with the minimal weight W, i.e., the server with the minimum estimated average system time.

When the estimated utilization of a server is one ($P \geq 1$) or more, the algorithm sets its weight to a very large value ($W = 1000$). This "penalty" makes the server very unlikely to be selected and effectively avoids sending new requests to a saturated server. In case two or more servers have the same smallest W value, tying is deterministically resolved by selecting the first server in the internal list.

4.5 Flowchart of proposed model dynamic load balancing with utilizes queueing theory

This section outlines the flowchart proposed in Figure 4 within an SDN environment, utilizing queuing theory and real-time performance analysis. The diagram starts by sending a

request; it then decides the rate of arrival (λ) and the rate of service (μ) offered by the servers and then computes the utilization factor (ρ) to evaluate the level of busyness of the server. Then, the weight (W) is computed, which is the average time that a request is in the system; in case W is more than the threshold, a high default weight is placed on the request to be redirected to the less busy servers and maintain the QoS. The system identifies the server that has the lowest weight to attain optimal distribution, and the QoS measures of packet loss, response time, and throughput are computed. This process is repeated until the request is fulfilled, and the network is made efficient, stable, and scalable according to the SDN principles.

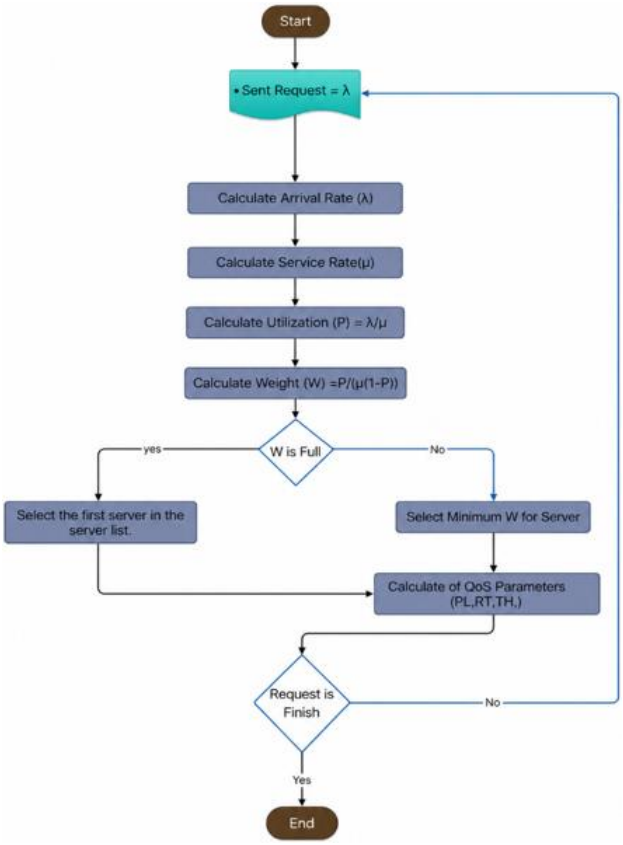

Figure 4. Flowchart proposed algorithm dynamic load balancing with utilizes queueing theory (DLBQT)

Algorithm 1: Proposed Dynamic Load Balancing based on Queueing Theory

Input: Arrival Rate (λ), Service Rate (μ)
Output: Weight (W), Selected Server
1. Begin
2. While (true) do:
3. For each server, calculate Arrival Rate (λ) based on Eq. (5)
4. For each server, calculate Service Rate (μ) based on Eq. (6)
5. For each server, calculate Utilization (P) based on Eq. (7)
6. For each server, calculate Weight (W) based on Eq. (8)
7. If P < 1:
    W = P / (μ * (1 - P))
    Select a server with a minimum W
8. Else:
    W = 1000
    Select the first server in the server list.
9. If the request is finished:
    Return QoS metrics
10. Else:
    Continuing processing requests
11. End If
12. End While
13. End

The pseudocode for the algorithm implements dynamic load balancing in an SDN environment using queueing theory, as illustrated in the flowchart above. The controller continuously checks the status of each server and calculates the arrival rate (λ) and service rate (μ) for each server to determine the weight (W). Requests are directed to the least-loaded server, and a request is discarded when the weight (W) exceeds the maximum limit, while measuring (QoS) metrics and upon request completion (Algorithm 1).

5. NETWORK SETUP AND EXPERIMENTAL METHODS

This section explains the general network structure, the types of environments and their characteristics, along with server specifications, and finally discusses the results of the proposed algorithm compared with other algorithms.

5.1 Network structure

The simulation environment was implemented using the Mininet simulator. Figure 5 illustrates the proposed topology in an SDN environment, known as the "Single Topology," in which a group of hosts (host1 through host4) send data traffic to a central switch. The switch, in turn, routes this traffic to Server1 or Server2 depending on the rules configured within the network controller. The controller, his type being POX, dynamically manages flow rules to achieve optimal data traffic distribution and load balancing among the servers.

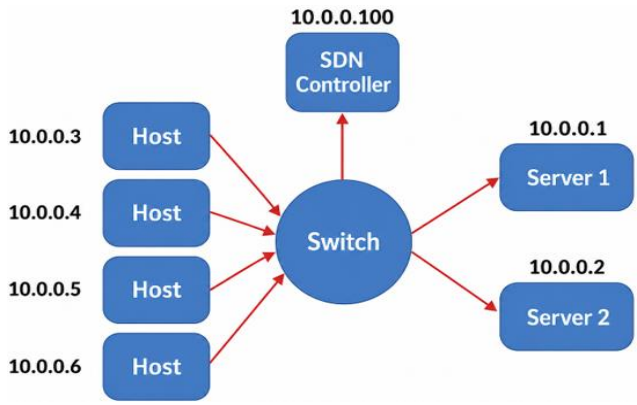

Figure 5. The proposed network topology in Software-Defined Networking (SDN)

5.2 Environments

The proposed models were tested in two different environments: a homogeneous environment and a heterogeneous environment. A homogeneous environment refers to one in which servers have the same specifications and computational resources, providing a stable and balanced testing environment, while the heterogeneous environment reflects the reality of real-world networks, where server capabilities vary in terms of processor and memory, posing a

challenge to load-balancing algorithms in terms of adaptation and effective load distribution. As shown in Table 4, the table details the full specifications of the servers for both environments as well as the computer specifications.

Table 4. Server specifications for homogeneous and heterogeneous experimental environments

Environment	Server	CPU Model	CPU Cores	Random Access Memory (RAM) (GB)
Homogeneous	Server 1	Intel i7-10850H	12	32
	Server 2	Intel i7-10850H	12	32
Heterogeneous	Server 1	Intel i7-10850H	3	8
	Server 2	Intel i7-10850H	1	3

To establish a baseline fairness protocol, the algorithms LC, LCPU, and LRAM are implemented and configured exactly as in our previous work, where they were introduced and evaluated as server-side load balancing schemes for SDN and collectively used as baseline algorithms. In this study, we reuse the same parameter settings (e.g., CPU and RAM sampling intervals and thresholds for LCPU and LRAM, and the standard configuration for LC) to ensure a fair and consistent comparison, while integrating them into the same POX controller module as DLBQT. All load balancing algorithms (LC, LCPU, LRAM, and DLBQT) are implemented in the same POX controller module and use the same event-driven update mechanism. Table 5 summarizes the decision metrics and main configuration parameters of each algorithm, helping to ensure that performance differences arise from the decision rules rather than from different monitoring frequencies or controller settings.

Table 5. Configuration of load balancing algorithms

Algorithm	Decision Metric(s)	Update Mechanism	Key Parameters
LC	Number of active connections	On new-flow installation	Configured as in this study [15]
LCPU	CPU utilization	Event-driven (same as DLBQT)	Configured as in this study [15]
LRAM	RAM utilization	Event-driven (same as DLBQT)	Configured as in this study [15]
DLBQT	Queueing Weight (W)	Event-driven at each routing decision using λ,μ from last 10 windows	$T_{\{win\}} \geq 1\ s,$ $W = 1000$

Note: LC = Least Connection; LCPU = Least CPU; LRAM = Least RAM; DLBQT = Dynamic Load Balancing based on Queueing Theory; CPU = Control Process Unit; RAM = Random Access Memory

Table 6. Unified experimental configuration for the Software-Defined Networking (SDN) simulations

Category	Parameter	Description / Value
SDN platform	Simulator	Mininet
SDN platform	Controller	POX (single centralized controller)
Topology	Network structure	Single switch with multiple hosts sending traffic to two servers (as in Figure 5)
Environments	Homogeneous environment	Two servers with identical CPU and RAM specifications (see Table 4)
Environments	Heterogeneous environment	Two servers with different CPU cores and RAM capacities (see Table 4)
Server specs	CPU model	Intel i7-10850H for all servers
Load-balancing schemes	Compared algorithms	LC, LCPU, LRAM, and DLBQT are implemented in the same POX module
Dataset	Traffic traces	16 real-world trace files derived from Facebook data-center traffic
Dataset	File-size range	1 KB to 6.8 MB, covering light and heavy objects
Traffic generation	Request pattern	Hosts repeatedly request files using a pseudo-random permutation of the 16 traces
Traffic generation	Inter-arrival times	Uniformly distributed random inter-arrival times over a fixed interval
Randomization	Random seed	Fixed seed per run, shared across all algorithms for fair comparison
Randomization	Additional seeds	Different recorded seeds used when multiple runs are executed
Load levels	Number of requests	1,000; 100,000; and 1,000,000 total requests per experiment
Run configuration	Number of runs	3 runs per configuration (results reported as averages)
Evaluation metrics	Performance metrics	Packet loss, response time, and throughput

Note: POX = Python OpenFlow eXtensible; LCPU = Least CPU; LRAM = Least RAM; DLBQT = Dynamic Load Balancing based on Queueing Theory; CPU = Control Process Unit; RAM = Random Access Memory

All simulations were executed on a dedicated Linux workstation equipped with an Intel Core i7-10850H CPU, 32 GB of RAM, and a 512 GB SSD, running Ubuntu 22.04 LTS. Mininet version 2.3.0 and the POX controller (git snapshot 2023_01_15) were used as the SDN emulation platform. The main configuration parameters of the experimental setup are summarized in Table 6.

5.2.1 Traffic generation from real-world dataset

The 16 real-world trace files of sizes 1 KB to 6.8 MB are used to generate traffic. In both experiments, hosts request these files over and over again on the servers. The pseudo-random permutation of the set of 16 files is used to select the file index of each new request so that all the 16 files are exercised with different load levels. This process makes sure that in the traffic mix, there are light (1-2 KB) and heavy (196-6836 KB) objects in the traffic mix that will reflect realistic content diversity.

5.2.2 Randomization and reproducibility

To make the experiments reproducible, all pseudo-random choices in the traffic generator (file selection and inter-arrival times) are driven by a fixed random seed at the start of each run. Using the same seed for different algorithms guarantees

that DLBQT, LCPU, LC, and LRAM are evaluated under identical request sequences. When additional runs are performed, different seeds are used, but the seeds are recorded so that the entire sequence of requests can be replayed if needed.

5.2.3 Inter-arrival time distribution

Inter-arrival times between consecutive requests are generated from a uniform distribution over a fixed time interval. This uniform randomization models bursty yet bounded request arrivals and, combined with the diversity of file sizes, produces a wide range of instantaneous loads on the servers. The same inter-arrival time sequence (determined by the fixed random seed) is used for all algorithms, ensuring a fair comparison under identical traffic conditions.

5.2.4 Number of runs and averaging

Each experimental condition (algorithm, environment, and total number of requests 1,000, 100,000, and 1,000,000) is executed R = 3 times with different random seeds. For each metric (packet loss, response time, throughput), the value reported in Tables 7-12 at each load level is the average across the R runs. The "Improving" rows in these tables are then computed using the percentage-improvement metric in Eq. (1) on the aggregated results across the three load levels for DLBQT versus each baseline algorithm.

5.3 Results and Discussion

For all metrics, the results at each load level (1,000, 100,000, and 1,000,000 requests) represent the averages over three independent runs per algorithm and environment, while the percentage improvements in the last row summarize the overall gain of DLBQT using Eq. (1).

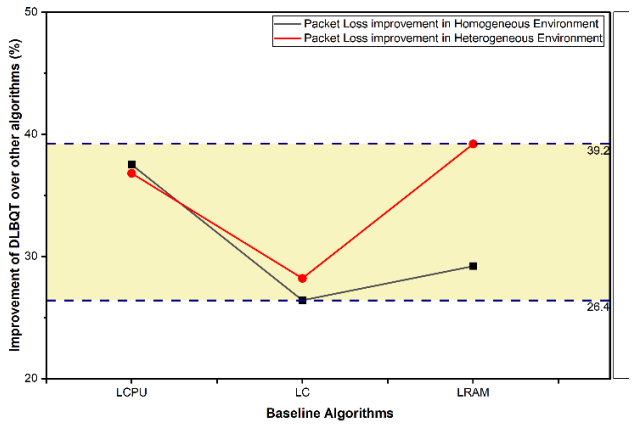

Figure 6. Packet loss in homogeneous and heterogeneous environments

The packet-loss results for both homogeneous and heterogeneous environments are summarized in Tables 7 and 8 and illustrated in Figure 6. Tables 7 and 8 present the results of the DLBQT algorithm compared to the LCPU, LC, and LRAM algorithms in terms of their ability to reduce packet loss in homogeneous and heterogeneous environments at different load levels. No packet loss was observed at low loads (1,000 requests), while loss increases as the load increases for all algorithms. The DLBQT algorithm achieved the lowest packet loss rates (200 and 1848 packets), and its improvement rates were 37.5%, 26.4%, and 29.2% higher, respectively, compared to the other algorithms. In the heterogeneous environment, DLBQT achieved the lowest packet loss values (172 and 1730 packets), representing substantially better performance than the other algorithms, with improvement rates of 36.8%, 28.2%, and 39.2%, respectively. This demonstrates its high efficiency in reducing packet loss and its strong ability to adapt to variable resources in the heterogeneous environment.

Table 7. Packet loss (%) in homogeneous environment

Algorithms Total Requests	LCPU	LC	LRAM	DLBQT
1000	0.00	0.00	0.00	0.00
100000	301	245	251	200
1000000	3160	2819	2987	1848
Improving	37.5%	26.4%	29.2%	-

Note: LCPU = Least CPU; LC = Least Connection; LRAM = Least RAM; DLBQT = Dynamic Load Balancing based on Queueing Theory

The response-time results in both environments are given in Tables 9 and 10 and visualized in Figure 7. Tables 9 and 10 show the response times of the DLBQT algorithm compared to the LCPU, LC, and LRAM algorithms in homogeneous and heterogeneous environments at different load levels. At low load (1,000 requests), the DLBQT algorithm recorded the lowest response times in both environments (0.0426 and 0.0388 s), and these values increase as the load increases for all algorithms. The DLBQT algorithm demonstrated its best performance at medium and high loads, recording 0.3663 s at 100,000 requests and 0.3756 s at 1,000,000 requests, with improvements of 8.61%, 7.78%, and 6.94% over LCPU, LC, and LRAM, respectively. In the heterogeneous environment, it performed best, achieving the lowest response times at 100,000 and 1,000,000 requests (0.3648 and 0.3822 s), with improvement rates of 11.4%, 7.75%, and 6.93% compared to LCPU, LC, and LRAM, respectively, demonstrating its excellent ability to adapt to changes in server resources.

Table 8. Packet loss (%) in heterogeneous environment

Algorithms Total Requests	LCPU	LC	LRAM	DLBQT
1000	0.00	1.00	0.00	0.00
100000	261	226	260	172
1000000	2861	2567	3117	1730
Improving	36.8%	28.2%	39.2%	-

Note: LCPU = Least CPU; LC = Least Connection; LRAM = Least RAM; DLBQT = Dynamic Load Balancing based on Queueing Theory

Table 9. Response time (s) in homogeneous environment

Algorithms Total Requests	LCPU	LC	LRAM	DLBQT
1000	0.0484	0.0511	0.0475	0.0426
100000	0.3902	0.3725	0.3816	0.3663
1000000	0.4070	0.3956	0.4016	0.3756
Improving	8.61%	7.78%	6.94%	-

Note: LCPU = Least CPU; LC = Least Connection; LRAM = Least RAM; DLBQT = Dynamic Load Balancing based on Queueing Theory

The throughput results for the homogeneous and heterogeneous environments are reported in Tables 11 and 12 and depicted in Figure 8. Tables 11 and 12 show the throughput performance of the DLBQT algorithm compared to the LCPU, LC, and LRAM algorithms in homogeneous and

heterogeneous environments at different load levels. The values at low loads (1,000 requests) indicate a close match between the algorithms, with a slight advantage for the DLBQT algorithm in the heterogeneous environment (19.62), although performance improves as the load increases. The DLBQT algorithm achieved the best throughput performance at medium and high loads (51.46 and 51.31) and improved performance by 4.34%, 1.33%, and 6.61% over LCPU, LC, and LRAM, respectively, in the homogeneous environment. In the heterogeneous environment, DLBQT similarly achieved the best throughput performance at medium and high loads (52.75 and 50.97), with higher improvement rates of 17.1%, 8.4%, and 18.4% over LCPU, LC, and LRAM, respectively, indicating its high efficiency in resource utilization and data transfer rate improvement.

Table 10. Response time (s) in heterogeneous environment

Algorithms Total Requests	LCPU	LC	LRAM	DLBQT
1000	0.0463	0.0422	0.0417	0.0388
100000	0.3948	0.3892	0.3918	0.3648
1000000	0.4268	0.4196	0.4107	0.3822
Improving	11.4%	7.75%	6.93%	-

Note: LCPU = Least CPU; LC = Least Connection; LRAM = Least RAM; DLBQT = Dynamic Load Balancing based on Queueing Theory

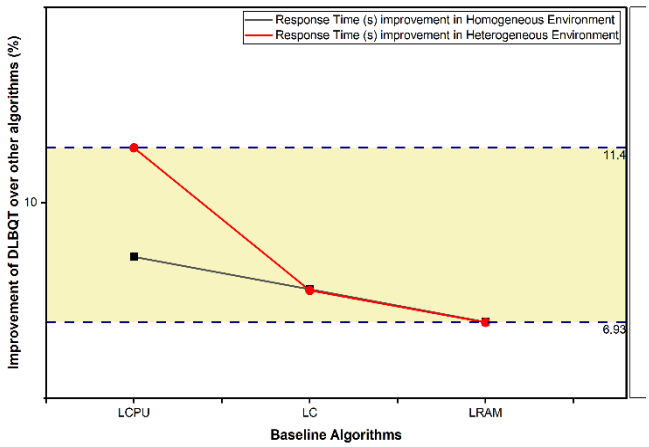

Figure 7. Response time in homogeneous and heterogeneous environments

Table 11. Throughput (bits/s) in homogeneous environment

Algorithms Total Requests	LCPU	LC	LRAM	DLBQT
1000	18.42	17.99	15.99	15.84
100000	46.34	49.09	47.75	51.46
1000000	44.24	46.18	45.41	51.31
Improving	4.34%	1.33%	6.61%	-

Note: LCPU = Least CPU; LC = Least Connection; LRAM = Least RAM; DLBQT = Dynamic Load Balancing based on Queueing Theory

Table 12. Throughput (bits/s) in heterogeneous environment

Algorithms Total Requests	LCPU	LC	LRAM	DLBQT
1000	16.30	19.36	15.22	19.62
100000	46.71	47.92	47.27	52.75
1000000	43.18	44.77	44.39	50.97
Improving	17.1%	8.4%	18.4%	-

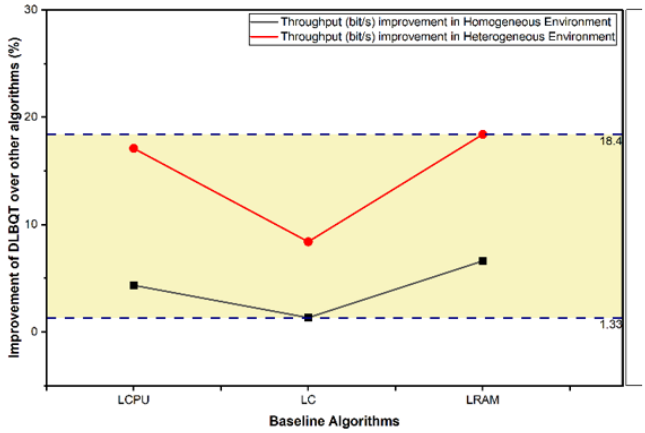

Figure 8. Throughput in homogeneous and heterogeneous environments

It is important to note that, although several state-of-the-art load balancing schemes were proposed in 2018–2026, most of them employ different traffic models, controller platforms, or network topologies, which makes a direct numerical comparison (using the same traces and code base) impractical. For this reason, our quantitative evaluation focuses on three widely used and well-documented dynamic baselines (LCPU, LC, and LRAM), while the relationship to more recent state-of-the-art methods is discussed conceptually in terms of their design choices, performance goals, and identified limitations.

5.4 Limitations and scope of the evaluation

More specifically, the current evaluation is limited in three main aspects. First, the study does not include the effects of load balancing, failover behavior, and inter-controller communication overhead, which are essential in large-scale, multi-controller deployments. Second, the dual-server configuration emphasizes server-side decision-making and makes the study of queuing behavior easy, but it is not a complete representation of a large data-center cluster of tens or hundreds of heterogeneous servers. Third, the three load levels (1,000, 100,000, and 1,000,000 requests) cover a controlled range of light, medium, and heavy traffic situations but do not include highly bursty traffic, long-term diurnal variations, or adverse traffic situations. The restrictions encourage the future work that is described in Section 7, in which DLBQT will be tested with larger server pools, multi-controller architectures, and more varied and realistic traffic profiles.

6. DISCUSSION

This paper analyses the performance of the proposed DLBQT algorithm using a small sample size of n = 3, corresponding to three request load levels (1,000, 100,000, and 1,000,000). In this study, the effect size between DLBQT and each baseline algorithm (LCPU, LC, and LRAM) was computed using Cohen's d for paired differences. For each metric and environment, cohen's d based on paired differences between the baseline and DLBQT at each load level was first calculated based on the values reported in Tables 7-12, and these differences were then used to obtain the corresponding effect-size estimates. Also, effect sizes (Cohen's d) are therefore computed over three representative load levels (n =

3) for each scenario. Consequently, the reported values should be viewed as descriptive indicators of the magnitude of the observed differences rather than as the basis for formal statistical inference. A more robust inference would require repeated runs per operating point and additional load levels to estimate variability and confidence intervals (CI). The mean and Standard Deviation (SD) of these differences were obtained as:

$$Mean_{of\ Differences} = \frac{(sum\ of\ (LCPU_i - DLBQT_i))}{n} \quad (9)$$

$$SD_{differences} = sqrt\left(\frac{sum\left(\left(D_i - Mean_{of\ Differences}\right)^2\right)}{(n-1)}\right) \quad (10)$$

and Cohen's d was then computed using:

$$d = \frac{Mean_{of\ Differences}}{SD_{differences}} \quad (11)$$

where, n = 3 is the number of load levels. This paired-difference formulation quantifies the strength of the performance gap between DLBQT and each baseline across the three tested load levels according to which the values are small (around 0.2 to 0.49), medium (around 0.5 to 0.79), and large (around 0.8 and more) between the proposed algorithm and the other algorithms, as shown in Table 13 (Packet Loss), Table 14 (Response Time), and Table 15 (Throughput).

Concerning the packet loss, Table 13 outcomes show that in a homogeneous environment, the effect size (Cohen's d based on paired differences) of DLBQT is always large (d > 1.3), and similarly in a heterogeneous environment. This indicates that there is a significant decrease in the loss of packets, and the better results are shown in the case of heterogeneous ones, which proves that the proposed algorithm is robust. This can also be exemplified in Figure 9 for packet loss comparison, whereby the DLBQT is always lower in the packet loss rate in all the request levels.

Table 13. Effect size for packet loss (both environments)

Comp.	Homo. (d)	Interp.	Hetero. (d)	Interp.
DLBQT vs LCPU	1.43	Large	1.55	Large
DLBQT vs LC	1.32	Large	1.40	Large
DLBQT vs LRAM	1.38	Large	1.60	Large

Regarding response time, Table 14 demonstrates that the effect sizes (d > 1.3) in either environment are large in all the comparisons. This shows that DLBQT has a substantial decrease in latency, and its performance will improve in a stable and changing network environment. In Figure 10 for response time comparison, with DLBQT having the smallest values of response time.

Concerning the throughput, as illustrated in Table 15, the effect sizes are large in all the situations (d ≥ 1.05). These findings suggest that DLBQT is effective in improving throughput with a minor increase in the heterogeneous, which suggests its effectiveness in diverse network environments. Figure 11 for throughput comparison supports these findings, showing that the highest throughput was achieved by the DLBQT in all cases.

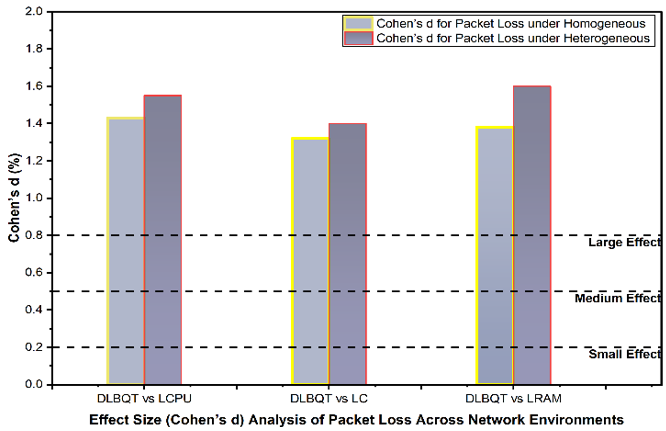

Figure 9. Effect size (Cohen's d) for packet loss

Table 14. Effect size for response time (both environments)

Comp.	Homo. (d)	Interp.	Hetero. (d)	Interp.
DLBQT vs LCPU	1.35	Large	1.50	Large
DLBQT vs LC	1.42	Large	1.48	Large
DLBQT vs LRAM	1.38	Large	1.55	Large

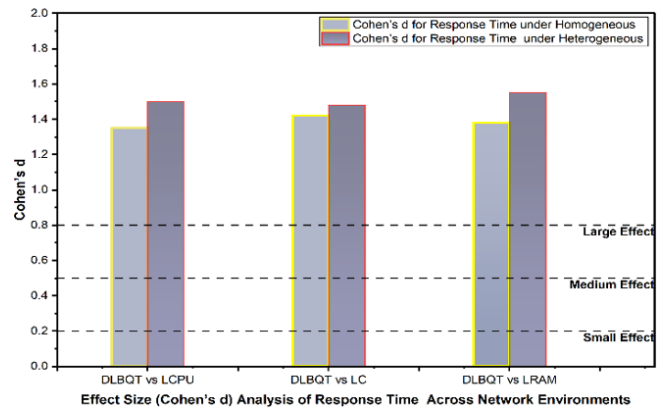

Figure 10. Effect size (Cohen's d) for response time

In general, despite the relatively small sample size (n = 3), the results consistently demonstrate large effect sizes across all evaluated metrics. The results indicate that the suggested DLBQT algorithm provides significant improvements in the areas of packet loss, response time, and throughput. More so, the fact that the effect sizes are always higher in the heterogeneous environment proves the strength and versatility of the proposed approach in more complicated and dynamic conditions of a network.

DLBQT can achieve the same objectives by an explicit analytical model, in contrast to recent state-of-the-art approaches, which often pursue QoS-conscious and multi-metric load balancing using heuristic or AI-based decision mechanisms. DLBQT can be used to refer to various open problems that were identified in recent surveys, including the accurate prediction of performance at high load and under real evaluation conditions.

Table 15. Effect size for throughput (both environments)

Comp.	Homo. (d)	Interp.	Hetero. (d)	Interp.
DLBQT vs LCPU	1.10	Large	1.25	Large
DLBQT vs LC	1.05	Large	1.10	Large
DLBQT vs LRAM	1.15	Large	1.30	Large

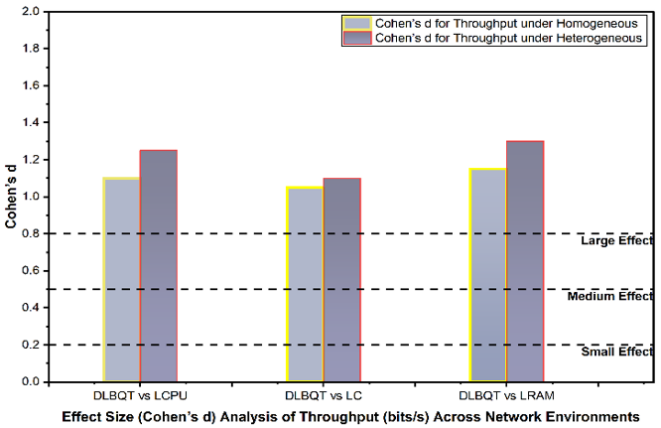

Figure 11. Effect size (Cohen's d) for throughput

Based on the improvement percentages reported in Tables 7-12, additional descriptive statistics were computed to further quantify the behavior of DLBQT relative to the baseline algorithms. Values represent the percentage improvement of DLBQT over LCPU, LC, and LRAM across the three evaluated load levels (1,000, 100,000, and 1,000,000 requests). For each load level and environment, the experiments were executed R = 3 times and the improvement values were computed from the averaged results, so that N denotes the number of improvement values per environment. Using OriginPro 2026 (OriginLab, Northampton, MA, USA), the mean, Standard Deviation (SD), and 95% CI of these improvement values were calculated for each performance metric and environment, providing a concise summary of the distribution of the observed improvements across the evaluated load levels and baseline algorithms. In addition, the performance figures for packet loss, response time, and throughput include error bars representing ±1 SD around the mean at each load level, which visually depict the variability across the R = 3 runs and further support the stability of the observed gains of DLBQT.

The improvement percentages of DLBQT over the baseline algorithms show a consistent positive gain in both environments (Figures 12 and 13). The mean improvement is 31.03% with an SD of 5.77%, and the 95% CI is [18.89%, 43.75%], showing that the packet loss reduction is still significant for the evaluated operating points; the moderate SD is due to the natural variation in the degree of improvement across the three baseline algorithms that differ in their congestion-handling mechanisms, while the consistently positive CI bounds confirm that DLBQT reliably outperforms all baselines under uniform server conditions. The mean improvement is 34.73% (SD 5.78%, 95% CI 20.29%–49.17%) in the heterogeneous environment, due to the higher inefficiency of heuristic-based baselines in this case, where DLBQT's queueing-theory model is able to identify and avoid overloaded servers and thus achieve larger packet loss reductions. The positive lower CI bound in both environments supports the consistency of the advantage of DLBQT and not just due to isolated operating conditions. Normality tests were also applied to the improvement percentages, and the P values obtained were 0.338 for the homogeneous environment and 0.279 for the heterogeneous environment, both of which exceeded the conventional significance level of 0.05, suggesting that there was no significant deviation from normality and that the distribution of the observed improvements could be described using parametric summaries (mean and SD) and 95% confidence intervals (Table 16).

Table 16. Statistics of packet loss improvement

Environment	N	Mean (%)	SD (%)	CI = 95%	Normality P-Value
Homogeneous	3	31.03	5.77	16.8 - 45.3	0.338
Heterogeneous	3	34.73	5.78	20.3 - 49.1	0.279

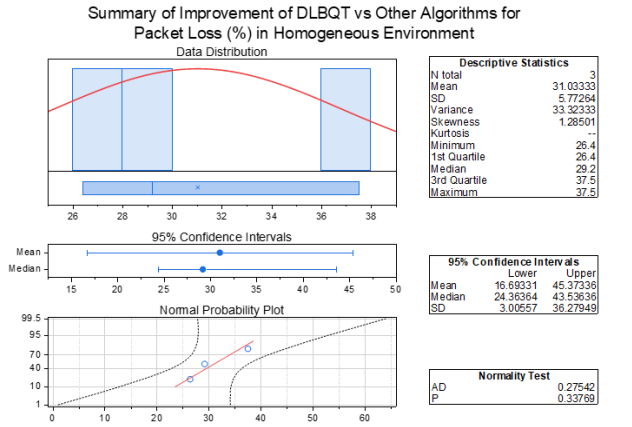

Figure 12. Statistics of packet loss improvement (homogeneous)

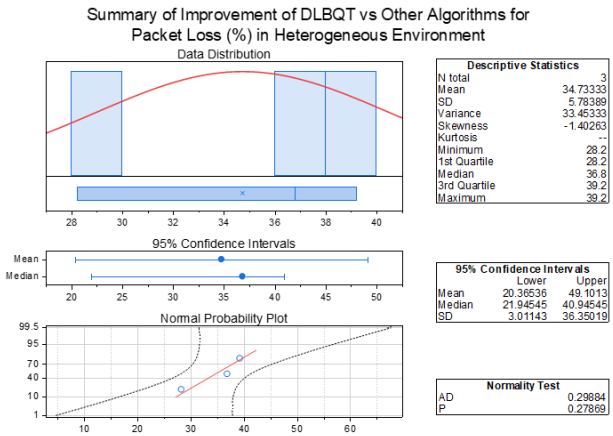

Figure 13. Statistics of packet loss improvement (heterogeneous)

Table 17. Statistics of response time improvement

Environment	N	Mean (%)	SD (%)	CI = 95%	Normality P-Value
Homogeneous	3	7.78	0.84	5.70 - 9.85	0.631
Heterogeneous	3	8.69	2.38	2.78 - 14.6	0.224

Note: SD = Standard Deviation; CI = confidence intervals; R = Number of experimental runs per configuration

The improvement percentages in response time for DLBQT

over the baseline algorithms show a consistent decrease in response time in both environments (Figures 14 and 15). The mean improvement is 7.78% in the homogeneous environment with an SD of 0.84%, and the 95% confidence interval is from 5.70% to 9.85%, which shows that the queueing-theory-based scheduling of DLBQT is stable and reliable for the different operating points evaluated in the homogeneous environment; the small SD indicates that the scheduling of DLBQT is consistent across uniform server capacities. The mean improvement is 8.69% (SD 2.38%) with a 95% confidence interval of 2.78%–14.60% in the heterogeneous environment due to the larger variability in baseline performance across the different server types, which led to a larger range of improvements observed. Nevertheless, the lower bound of the CI is positive in all cases, indicating that DLBQT is always superior to all three baseline algorithms in all environments. Normality tests for both the homogeneous and heterogeneous environments also yielded P-values above 0.05, with values of 0.631 and 0.224, respectively, indicating that there is no significant deviation from normality and that parametric summaries and 95% confidence intervals can be used to describe the distribution of the improvements observed in both environments (Table 17).

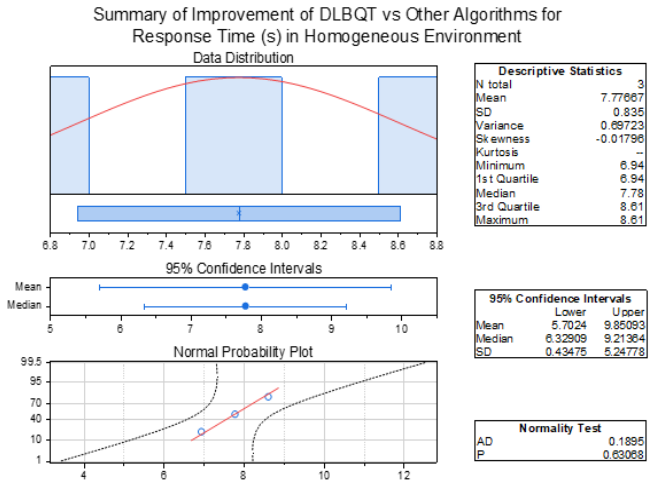

Figure 14. Statistics of response time improvement (homogeneous)

In terms of throughput, the enhancement percentages of DLBQT over the baseline algorithms are clearly in favor of DLBQT, especially in the heterogeneous environment (Figures 16 and 17). The mean improvement is 4.09% (SD 2.65%), and the 95% confidence interval is (−2.48%, 10.66%) for the homogeneous environment, suggesting that the throughput gains of DLBQT over the baselines are fairly small and may fluctuate around zero, depending on the baseline algorithm. This behavior is because in traditional dynamic heuristics, the load is already well balanced when all servers have similar capacities, and there is little room to gain more throughput. In the heterogeneous environment, the mean improvement increases to 14.63% (SD 5.35%), and the 95% CI is (2.17%, 27.09%) with a positive lower bound. This result indicates that DLBQT is especially effective in taking advantage of the capacity differences between the servers: instead of simply counting connections or CPUs, it can send more connections to high-capacity servers and prevent overloading of low-capacity servers, resulting in significant and statistically stable throughput gains over all three baseline algorithms. The normality tests for the improvement values gave P-values of 0.598 for the homogeneous environment and 0.154 for the heterogeneous environment, both of which were greater than 0.05, indicating that there was no significant difference between the distribution of the improvement values and the normal distribution and that parametric statistics and 95% confidence intervals could be used to characterize the distribution of throughput improvements (Table 18).

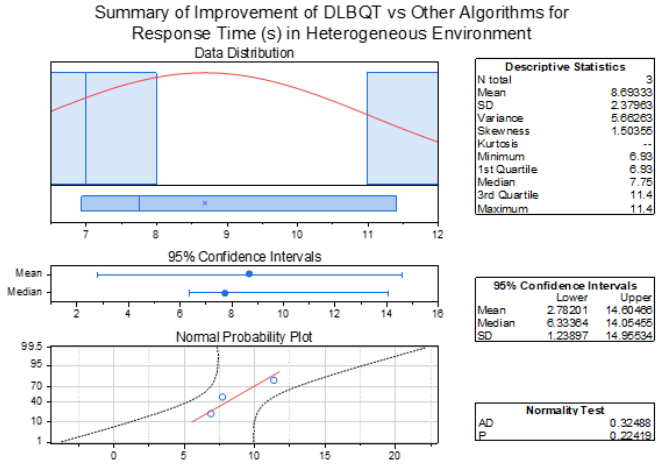

Figure 15. Statistics of response time improvement (heterogeneous)

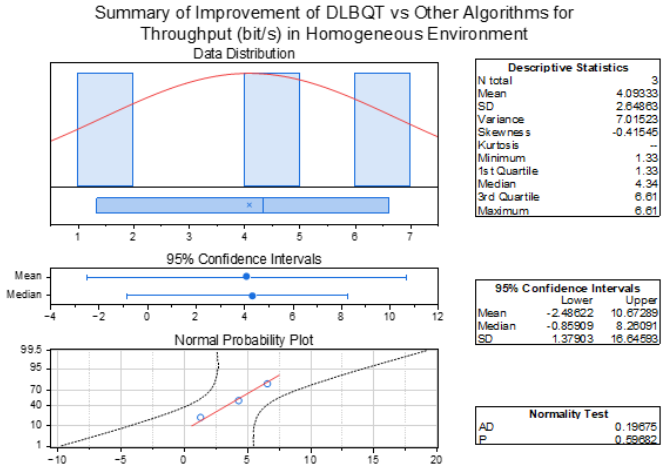

Figure 16. Statistics of throughput improvement (homogeneous)

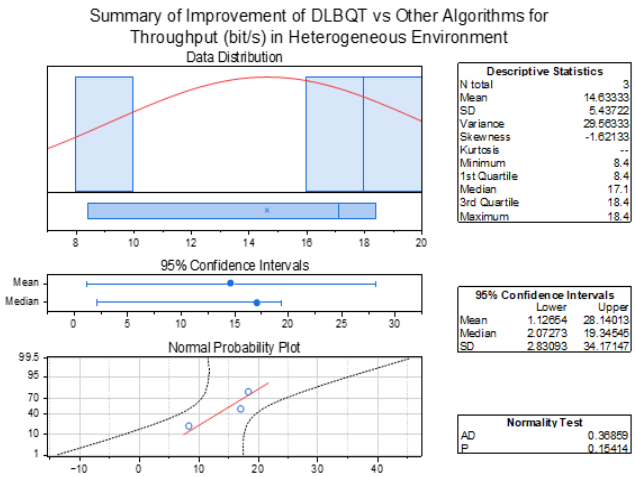

Figure 17. Statistics of throughput improvement (heterogeneous)

Table 18. Statistics of throughput improvement

Environment	N	Mean (%)	SD (%)	CI = 95%	Normality P-Value
Homogeneous	3	4.09	2.65	−2.48-10.6	0.598
Heterogeneous	3	14.63	5.35	2.17 - 27.09	0.154

Note: SD = Standard Deviation; CI = confidence intervals; R = Number of experimental runs per configuration

6.1 Sensitivity to load levels

To examine the sensitivity of the compared algorithms to different traffic loads, the results in Tables 7-12 and Figures 6-8 were analysed across the three evaluated request levels (1,000, 100,000, and 1,000,000 requests). At low load (1,000 requests), all algorithms show negligible packet loss and very small differences in response time and throughput, indicating that the servers operate far from saturation in both environments. As the load increases to 100,000 and 1,000,000 requests, LCPU, LC, and LRAM exhibit a faster growth in packet loss and latency, especially in the heterogeneous environment, reflecting uneven load distribution and higher utilization of the weaker server. In contrast, DLBQT maintains substantially lower packet loss and shorter response times at medium and high loads, while sustaining higher throughput, because the queueing-theoretic weight steers more traffic towards underutilized, high-capacity servers and penalizes nearly saturated servers. The error bars in Figures 12-17 further show that the variability of DLBQT remains moderate across the three load levels, whereas the baseline algorithms exhibit larger fluctuations at high load, particularly in the heterogeneous scenario, confirming that DLBQT is less sensitive to load increases in terms of both server utilization and end-to-end latency.

7. CONCLUSION

In this paper, a DLBQT was proposed and evaluated for SDN environments using the Mininet simulator and the POX controller. The algorithm models each server as a queueing system and uses real-time estimates of arrival rate, service rate, and utilization to compute a weight that guides the routing of incoming requests. Within the evaluated SDN setup, experimental results in both homogeneous and heterogeneous environments show that DLBQT achieves consistent performance improvements over the baseline algorithms LCPU, LC, and LRAM in terms of packet loss, response time, and throughput.

These improvements are further supported by descriptive statistics computed over the three evaluated load levels, including positive mean percentage gains, moderate SD, and 95% confidence intervals that are mostly above zero for packet loss and throughput, as well as normality P-values above 0.05. Together with the large effect sizes (Cohen's d ≈ 1.05–1.60), these results indicate that the performance advantages of DLBQT are stable across the tested operating conditions, while still being limited by the small sample size (n = 3) and the specific single-controller, two-server SDN scenario considered in this study. Thus, in the evaluated SDN scenario, the queueing-theory-based controller enables more accurate routing decisions and a better balance of requests across servers than the considered heuristic load-balancing schemes.

Future work will extend DLBQT to a greater number of servers and more complex topologies, integrate machine learning approaches to automatically tune algorithm parameters, and test the methodology under more real-world traffic statistics and various controller platforms. Therefore, the conclusions drawn in this work should be regarded as specific to the tested single-controller SDN configuration with two servers and three representative load levels, and further experiments on larger and more complex topologies are required before generalizing the findings to broader SDN deployments.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support provided by the Information Technology Research and Development Centre (ITRDC), University of Kufa, Iraq for this work.

DATA AVAILABILITY STATEMENT

The data, the Python source code and the experimental figures used to verify the results of the study are available and open source in a GitHub repository [29]. The repository contains the complete implementation of the proposed DLBQT algorithm, the configuration of the homogeneous and heterogeneous server environment, and all the experimental figures mentioned in this manuscript.

REFERENCES

[1] Chiti, F., Lotti, M., Picchioni, S., Pierucci, L. (2026). SDN-oriented 6G industrial IoT architecture design and application to optimal RIS placement and selection. Sensors, 26(2): 411. https://doi.org/10.3390/s26020411

[2] Jabbar, M.S. (2025). Enhancing an intelligent model for enhancing software-defined networking (SDN) achievement using fog computing and reinforcement learning for operational performance and dynamic resource management. International Journal of Intelligent Engineering & Systems, 18(5): 347. https://doi.org/10.22266/ijies2025.0630.25

[3] Mirmohseni, S.M., Tang, C., Javadpour, A. (2026). Optimizing network performance through cloud load balancing and virtualization in software-defined networking. Computing, 108(3): 45. https://doi.org/10.1007/s00607-026-01635-y

[4] Das, S., Adamson, J., Lee, A., Vaideswar, V., Kalafatis, S. (2026). Integrated strategies for bottleneck mitigation in SDN: A comprehensive analysis of caching, queuing, and network coding. In 2026 International Conference on Computing, Networking and Communications (ICNC), Maui, HI, USA, pp. 333-338. https://doi.org/10.1109/ICNC68183.2026.11416950

[5] Wei, Z., He, R., Song, C., Chen, X. (2026). Differentiated offloading and resource allocation with energy anxiety level consideration in heterogeneous maritime Internet of Things. IEEE Transactions on Network and Service Management, 23: 2166-2189 https://doi.org/10.1109/TNSM.2026.3655385

[6] Ravinder, B., Haritha, D., Naresh, V. (2026). GCOA: An effective task scheduling for load balancing in the cloud framework. Transactions on Emerging

Telecommunications Technologies, 37(1): e70343. https://doi.org/10.1002/ett.70343
[7] Boussaoud, K., En-Nouaary, A., Ayache, M. (2025). Dual-agent reinforcement learning for adaptive bandwidth allocation and congestion control in SD. International Journal of Intelligent Engineering & Systems, 18(7): 338-361. https://doi.org/10.22266/ijies2025.0831.24
[8] Kumar, S., Malik, A. (2026). Optimized controller placement and load balancing in SDN-enabled IoT networks. Microsystem Technologies, 32(4): 47. https://doi.org/10.1007/s00542-025-06004-x
[9] Dev, A., Rout, S., Patnaik, H.K., Singh, S., Patra, S.S. (2026). Intelligent SDN traffic engineering with MADRL. In 2026 7th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI), Goathgaun, Nepal, pp. 123-129. https://doi.org/10.1109/ICMCSI67283.2026.11412947
[10] Jasim, M.N. (2021). A proposed adaptive least load ratio algorithm to improve resources management in software defined network OpenFlow environment. Karbala International Journal of Modern Science, 7(1): 6. https://doi.org/10.33640/2405-609X.2255
[11] Almhanna, M.S., Murshedi, T.A., Al-Turaihi, F.S., Almuttairi, R.M., Wankar, R. (2023). Dynamic weight assignment with least connection approach for enhanced load balancing in distributed systems. https://doi.org/10.21203/rs.3.rs-3216549/v1
[12] Malbašić, T., Bojović, P.D., Bojović, Ž., Šuh, J., Vujošević, D. (2022). Hybrid SDN networks: A multi-parameter server load balancing scheme. Journal of Network and Systems Management, 30(2): 30. https://doi.org/10.1007/s10922-022-09642-y
[13] Haile, G.B., Zhang, J. (2021). Dynamic load balancing algorithm in SDN-based data center networks. International Journal of Engineering Research & Technology, 10(3): 388-394.
[14] Zhu, L., Cui, J., Xiong, G. (2018). Improved dynamic load balancing algorithm based on Least-connection scheduling. In 2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China, pp. 1858-1862. https://doi.org/10.1109/ITOEC.2018.8740642
[15] Alrammahi, M.A.M., Al-hamami, M.Y., Taher, A.M. (2026). Optimizing server-side dynamic load balancing in SDN using novel algorithms based On CPU, RAM, and connection metrics. International Journal of Advanced Research in Computer Science, 17(1): 21. https://doi.org/10.26483/ijarcs.v17i1.7401
[16] Fachrizal, F., Nugroho, O. (2026). Load balancing optimization through multi-label classification of network traffic using deep learning in distributed computing systems. Engineering, Technology & Applied Science Research, 16(1): 31310-31317. https://doi.org/10.48084/etasr
[17] Ali, A.A., Hussein, G.A., Muter, B.M., Hassen, O.A. (2025). DeepBalance: A deep reinforcement learning framework for dynamic load balancing in software-defined networks. Journal of Intelligent Systems & Internet of Things, 17(1): 279. https://doi.org/10.54216/JISIoT.170120
[18] Čilić, I., Žarko, I.P., Frangoudis, P., Dustdar, S. (2025). QoS-aware load balancing in the computing continuum via multi-player bandits. arXiv preprint arXiv:2512.18915. https://doi.org/10.48550/arXiv.2512.18915
[19] Jiang, Z.M., Hassan, A.E. (2015). A survey on load testing of large-scale software systems. IEEE Transactions on Software Engineering, 41(11): 1091-1118. https://doi.org/10.1109/TSE.2015.2445340
[20] Kenton, W. (2024). Calculate percentage change for finance and investment analysis. Investopedia https://www.investopedia.com/terms/p/percentage-change.asp.
[21] Roy, A., Zeng, H., Bagga, J., Porter, G., Snoeren, A.C. (2015). Inside the social network's (datacenter) network. In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, London, United Kingdom, pp. 123-137. https://doi.org/10.1145/2785956.27874
[22] Iqbal, A. (2021). Oddlab datasets – Cache folder. GitHub. https://github.com/aymeniq/Oddlab/tree/main/Datasets.
[23] Wakaiki, M. (2025). Stabilization of infinite-dimensional systems under quantization and packet loss. IEEE Transactions on Automatic Control, 70(10): 6625-6640. https://doi.org/10.1109/TAC.2025.3564223
[24] Pundir, M., Sandhu, J.K. (2021). A systematic review of quality of service in wireless sensor networks using machine learning: Recent trend and future vision. Journal of Network and Computer Applications, 188: 103084. https://doi.org/10.1016/j.jnca.2021.103084
[25] Gharbaoui, M., Martini, B., Castoldi, P. (2025). A WAN infrastructure manager prototype for QoS-based multi-site SDN-enabled NFV orchestration. Journal of Network and Systems Management, 33(3): 54. https://doi.org/10.1007/s10922-025-09929-w
[26] Cui, J., Lu, Q., Zhong, H., Tian, M., Liu, L. (2018). A load-balancing mechanism for distributed SDN control plane using response time. IEEE Transactions on Network and Service Management, 15(4): 1197-1206. https://doi.org/10.1109/TNSM.2018.2876369
[27] Rawas, S., Samala, A.D. (2026). Bio-inspired AI for adaptive and resilient software-defined networks: A self-healing approach for autonomous optimization and security. Iran Journal of Computer Science, 9(1): 7. https://doi.org/10.1007/s42044-025-00358-1
[28] SystemsArchitect.io (2024). Throughput. https://systemsarchitect.io/docs/requirements/estimates/initial/network-throughput.
[29] Alrammahi, M.A.M, (2026). Proposed Queueing Theory Algorithm– Datasets, Source Code, and Experimental Figures, GitHub repository. https://github.com/maghribalramahi83/sdn-load-balancing/tree/main/Proposed%20Queueing-Theory%20Algorithm.

NOMENCLATURE

Symbol	Meaning	Unit
W	Queueing-theoretic weight / estimated average system time	s
$Twin$	Observation window length	s
d	Cohen's effect size for paired differences	dimensionless
n	Number of evaluated load levels	dimensionless
R	Number of runs per	dimensionless

	configuration	

Greek symbols

λ	Arrival rate of incoming requests/tasks to a server	requests/s
μ	Service rate of completed requests/tasks at a server	requests/s
ρ	Server utilization, computed as λ/μ	dimensionless