I.J. Wireless and Microwave Technologies, **2026, 5, 1-19**
Published Online on October 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijwmt.2026.05.01

Comparative Study of Exploration Strategies in Q-Learning for Multi-Controller SDN Load Balancing

Binod Sapkota
Department of Electronics and Computer Engineering, Thapathali Campus, IoE, TU, Kathmandu, Nepal
E-mail: replybinod@gmail.com
ORCID iD: https://orcid.org/0009-0009-9418-0092

Utkarsha Shukla
Department of Electronics and Computer Engineering, Thapathali Campus, IoE, TU, Kathmandu, Nepal
E-mail: utkar24@gmail.com
ORCID iD: https://orcid.org/0009-0000-8832-5747

Babu R. Dawadi*
Department of Electronics and Computer Engineering, Pulchowk Campus, IoE, TU, Lalitpur, Nepal
E-mail: baburd@ioe.edu.np
ORCID iD: https://orcid.org/0000-0001-6449-399X
*CorrespondingAuthor

Shashidhar R. Joshi
Department of Electronics and Computer Engineering, Pulchowk Campus, IoE, TU, Kathmandu, Nepal
E-mail: srjoshi@ioe.edu.np
ORCID iD: https://orcid.org/0000-0001-9480-0495

Received: May 28, 2026; Revised: July 6, 2026; Accepted: August 3, 2026; Published: October 8, 2026

Abstract: Software Defined Networking enables centralized control and dynamic programmability, but achieving efficient load balancing across distributed controllers remains challenging because of traffic variability and scalability constraints. Despite the significant potential of reinforcement learning for multicontroller load balancing, existing studies primarily focus on developing new learning architectures or switch migration mechanisms, with limited attention given to systematically comparing exploration strategies under identical network conditions. The experimental setup consists of twelve switches for data forwarding and three distributed controllers following east-west communication. This study compares five exploration strategies of Q-learning using throughput, latency, controller response time, load balancing efficiency, fairness, scalability, and congestion-related metrics. Existing studies also often neglect fault tolerance, energy efficiency, and real-world validation. Under congested conditions, SOFT Q-learning records the lowest throughput reduction of 0.92, followed by upper confidence bound with 1.56, epsilon-greedy strategy with 2.34, prioritized experience replay with 3.29, and Boltzmann exploration with 6.06. The epsilon-greedy strategy achieves the highest throughput of 7.88 and fairness of 0.9999, soft reinforcement learning records the lowest latency of 0.0115 seconds, upper confidence bound achieves the fastest controller response time of 0.0285 seconds, and Boltzmann exploration attains the highest load balance ratio of 0.9865.

Index Terms: Boltzmann Exploration, Multicontroller, Load balancing, Reinforcement learning, Software Defined Networking

1. Introduction

Software Defined Networking (SDN) revolutionizes networking by separating the data plane (packet forwarding) from the control plane (network intelligence), enhancing flexibility, efficiency, and programmability [1]. Multiple

This work is open access and licensed under the Creative Commons CC BY 4.0 License.

controllers are crucial in SDN to avoid bottlenecks as network scale and control traffic grow. They enable better load distribution, faster decisions, and improved scalability, with overload prevention as the primary goal.

Static load balancing uses fixed rules like Round Robin, ratio-based, or priority-based methods without considering real-time traffic or server load. While easy to implement, it lacks adaptability, leading to higher latency, poor fault tolerance, and inefficient resource use under dynamic conditions [2]. Dynamic load balancing adaptively distributes traffic using real-time metrics like active connections, response time, and predictive algorithms, without dedicated hardware [3].

SDN and OpenFlow-based methods such as EHLBOF (Extended Health monitoring for Load Balancing in OpenFlow based network), traffic-aware, and response-time-based routing enhance throughput, reduce latency, and optimize resource usage. For instance, EHLBOF improved throughput by 47.39% over Round Robin, while LABERIO (Load Balanced Routing with OpenFlow) reduced transmission time by 13% [4]. Machine learning, particularly Deep Reinforcement Learning (DRL), further enhances adaptability in load balancing. Techniques like DRL-Switch Migration Strategy (SMS), using Double Deep Q-Networks (DDQN), optimize switch migration in multi-controller SDNs, reducing imbalance and improving stability [5]. The Effective slave controller allocation-based load balancing (ESCALB) algorithm balances controller loads in multi-domain IoT by combining ANP (Analytical Network Process) based controller ranking with 0/1 Knapsack-based switch selection. Upon overload, ANP ranks controllers, and the knapsack model selects switches for migration based on traffic and capacity. The Forwarding and Updating Module (FFUM) updates the flow rules in real time, which ensures continuous load balancing and QoS [6].

A balanced multi-controller SDN architecture improves scalability, fault tolerance, and resilience by distributing traffic and tasks across controllers, avoiding bottlenecks and failures. Real-time synchronization ensures stable performance and adaptive load balancing optimizes resource use and responsiveness in dynamic IoT environments [7].

Recent research has used artificial intelligence (AI) optimization methods to achieve load balancing in software-oriented networks. Researchers focus on AI-based load balancing (LB) techniques and divide them into nature-inspired, machine learning, mathematical models, and other LB methods. Their studies deal with the management of controllers and the movement of switches through the use of separate algorithms.

In the field of AI, reinforcement learning (RL) has become an extremely important core technology, and people's attention is also focused on research and application of RL. RL algorithms can explore the surrounding environment through agents and, without prior knowledge, continue to train for many cycles to make the most correct action decision [8]. By learning policies from interactions, RL enables autonomous decision-making in complex and uncertain domains.

Sun et al. [9] proposed a dynamic controller workload balancing scheme, enabling controller load balancing in software-defined networks with multi-agent reinforcement learning (MARVEL), which is based on multiagent reinforcement learning for the generation of switch migration actions, and their experimental results show that MARVEL outperforms competing existing schemes by improving the control plane request processing ability using less processing time.

The goal of this work is to compare five Q-learning exploration strategies for multi-controller SDN load balancing under identical experimental conditions. The study evaluates how each strategy affects throughput, latency, fairness, controller response time, scalability, and congestion resilience.

The key contributions of the proposed system are:

- Systematically compare five Q-learning exploration strategies for multicontroller SDN load balancing under identical experimental conditions.
- A performance analysis of the strategies using throughput, latency, fairness, controller response time, scalability, and congestion-related metrics.
- A performance analysis of the strategies using throughput, latency, fairness, controller response time, scalability, and congestion-related metrics.
- A discussion of trade-offs among the strategies to identify which method is best suited for different SDN operating conditions.

In the following chapters, the second chapter, literature review, discusses existing research on multi-controller SDN architectures and load-balancing methods in related works. The third chapter, methodology, presents the proposed framework, describes the flow of the work, discusses Q-learning load balancing algorithms in multicontroller SDN, hyperparameter tuning, and the congestion scenario. The fourth chapter, results and discussion, evaluates the experimental results and interprets their key insights along with comparative analysis of algorithms. The fifth chapter, conclusion and future work, describes the conclusions and identifies potential directions for further investigation.

2. Literature Review

Srivastava et al. [10] propose a DRL-based rerouting optimization strategy for SDN, dynamically considering switch, server, and controller loads. Tosounidis et al. [11] apply DRL for efficient service request balancing in data centers. Tennakoon et al. [12] introduce a Q-learning-based load balancing approach for 5G SDN, improving per-flow allocation

while reducing unsatisfied users. However, frequent updates led to computational overhead and latency, necessitating more efficient RL techniques.

Chen et al. [13] developed Automatic Load-Balancing Architecture Based on Reinforcement Learning (ALBRL) which is an RL-based SDN load-balancing framework with an enhanced DDPG algorithm, improving throughput and convergence but introducing latency due to prioritization mechanisms. Liang et al. [14] propose a Bayesian network-integrated RL method for congestion prediction in SDN-based IoT, achieving significant delay reduction but facing scalability and complexity challenges.

Ahmed et al. [15] introduce a deep active learning-based framework for intrusion detection and load balancing, doubling performance over traditional methods, yet requiring lightweight AI models for real-time SDN traffic management.

Li et al. [16] design an RL-based multi-controller SDN load balancing mechanism that dynamically migrates switches to reduce controller overload, but its high computational cost limits real-time applicability.

Xiang et al. [5] introduce a DRL-based switch migration strategy for controller load balancing, demonstrating effective load distribution. However, SDN-enabled IoT networks still face issues like static controller placement, high latency, and scalability constraints.

Tache et al. [17] categorized SDN optimization techniques into exact, heuristic, and ML-based approaches, highlighting the trade-offs between scalability, real-time adaptability, and computational costs. These challenges underline the need for hybrid AI-heuristic models to improve SDN efficiency while minimizing overhead.

Semong et al. [18] survey intelligent load balancing in SDNs, emphasizing the benefits of SDN's programmability and centralized control. They categorize existing methods into controller, server, wireless, path selection, and AI-based strategies, analyzing their performance and tools.

Kumar et al. [19] proposed a machine learning-based SDN load balancing method using Kmeans and DBSCAN on twelve flow parameters, enhancing traffic distribution and scalability over prior approaches.

Wang et al. [20] propose RDMAR, a rounding-based algorithm to jointly optimize controller and link load balancing in multi-controller SDNs. It addresses the NP-hard LBR-LC problem and reduces controller response time by 70% in simulations. Singh et al. [21] proposed an energy-efficient, secure load-balancing method for SDN-enabled fog computing using a DBN-based intrusion detection system and region-based fog node clustering. Their approach improves response time by 15%, energy use by 23%, and delay by 10%.

Although RL has advanced SDN load balancing, important challenges remain, including state-space growth, computational overhead, real-time decision constraints, and scalability under dynamic traffic conditions. Recent approaches such as MARVEL and ESCALB show promising progress in controller load balancing, but they use different design assumptions and optimization goals. This paper therefore focuses on a controlled comparison of Q-learning exploration strategies rather than proposing a new controller architecture.

3. Methodology

3.1. Instrumentation

The experimental framework was deployed on a personal server configured with 16 GB of RAM, an Intel® Core™ i5-1235U processor, and Intel® Iris Xe Graphics. This server functioned as the computational platform for training and testing machine learning models. The SDN environment was virtualized using Oracle VirtualBox, with Ubuntu 20.04 installed as the guest operating system. Python scripts were employed to configure the SDN topology, facilitating the simulation and evaluation of various network scenarios. The Ryu controller framework was utilized to implement and assess load-balancing mechanisms, as well as modules addressing controller responsiveness and congestion scenarios, within the SDN-enabled environment.

3.2. Data Collection

The Q-Learning Load Balancer in SDN gathers and processes real-time network data to optimize controller assignments and load distribution.

Data Sources

The experimental framework leverages Ryu controller event handlers and Mininet's traffic generation which collects a wide range of performance metrics within a synthetic, controlled and reproducible environment that accommodates real-world traffic traces. Its modular structure supports comparative analysis against baseline load distribution strategies and theoretical benchmarks, demonstrating the adaptability of the Q-learning approach under diverse network conditions.

Data sources include detailed network traffic metrics such as packet-in events, flow statistics, controller performance, switch-controller interactions, migration records, and latency measurements. Real-time coordination among distributed controllers is achieved through an MQTT-based communication protocol which enhances decision-making processes.

Additionally, performance logs capturing CPU and memory usage, response times, throughput, and latency further inform Q-learning state evaluations and system optimization, enabling dynamic monitoring and adjustment of the network.

3.3. Data Preprocessing

Before training the reinforcement learning agent, the collected SDN network telemetry was preprocessed to improve the quality and consistency of the input data. Let the raw network dataset be denoted as given in Eq.(1)

$$X = \{x_1, x_2, \ldots, x_n\}, \tag{1}$$

where each observation contains network performance metrics such as latency, throughput, controller load, and packet loss. Outliers were removed using the Interquartile Range (IQR) criterion, producing a cleaned dataset defined in Eq.(2)

$$X_{\text{clean}} = \{x_i \in X \mid Q_1 - 1.5(Q_3 - Q_1) \leq x_i \leq Q_3 + 1.5(Q_3 - Q_1)\}, \tag{2}$$

where Q_1 and Q_3 represent the first and third quartiles, respectively. Numerical features were subsequently normalized using Min-Max scaling to ensure that all state variables shared a common numerical range, thereby preventing features with larger magnitudes from dominating the learning process. In addition, temporal fluctuations in continuously monitored network metrics were reduced using moving average and exponential smoothing techniques, producing more stable observations for the reinforcement learning agent. The resulting preprocessed metrics constitute the state information used during controller selection and Q-learning updates. The normalized feature value is computed in Eq.(3)

$$x_i' = \frac{xi - \min(X)}{\max(X) - \min(X)} \tag{3}$$

where x_i denotes the original feature value and x_i' is the corresponding normalized value.

3.4. System design scenario

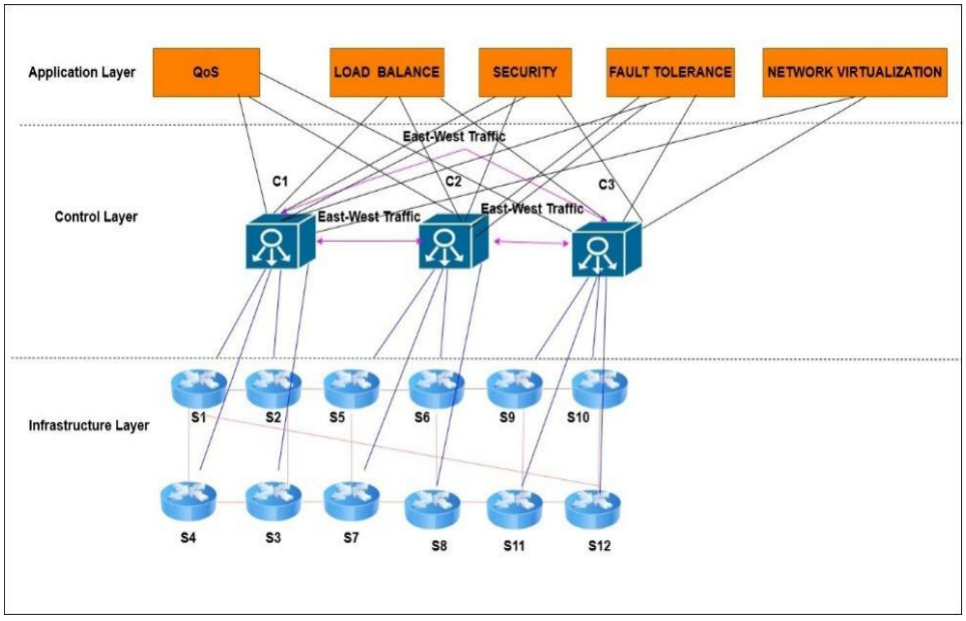


Fig. 1. Proposed system design for multi-controller load balancing

The SDN-enabled network architecture in Figure 1 is organized into three layers to enhance scalability, fault tolerance, and performance.

The Application Layer handles key functions such as Quality of Service (QoS) for traffic prioritization, dynamic Load Balancing, Security enforcement, Fault Tolerance for resilience, and Network Virtualization for flexibility and scalability. These features ensure secure, efficient, and high-performing network operations.

The Control Layer includes distributed controllers (C1, C2, C3) with East-West communication, enabling real-time synchronization, load sharing, and coordinated decisions, eliminating single points of failure.

The Infrastructure Layer consists of redundant switches (S1–S12) for efficient data forwarding. This layered design supports high availability, adaptability, and security, making it ideal for dynamic SDN environments.

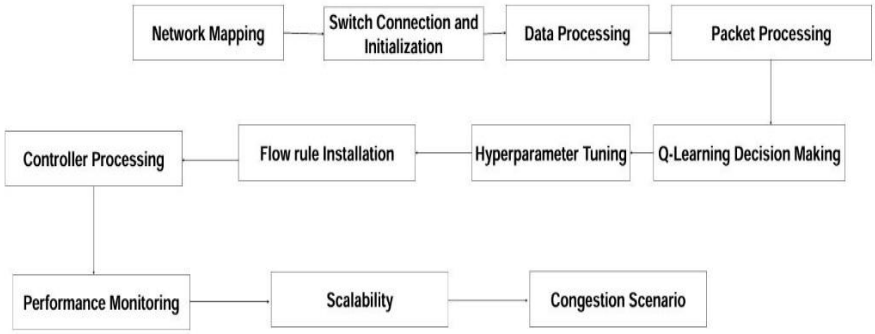


Fig. 2. Work Flow Diagram

The load balancing system in Figure 2 begins with network mapping, identifying switches, controllers, and traffic sources to define the topology. It then moves to switch connection and initialization, establishing links and setting up the Q-learning framework by initializing the Q-table and tuning parameters (ϵ, α, γ).

During packet processing, traffic is analyzed to extract Ethernet and IP details, enabling controllers to make routing and load-balancing decisions. The core Q-learning module uses an epsilon-greedy strategy to select optimal controllers based on current states, updating the Q-table with rewards to improve over time.

Flow rules are installed based on these decisions for efficient traffic handling. Lastly, performance monitoring tracks metrics such as latency, jitter, packet loss, throughput, energy use, and load distribution, while visualizing system performance and updated Q-values.

The controller's response time is determined by four factors: network propagation delay, queueing delay, processing time and decision-making delay. To enhance performance, the system employs weighted load balancing, performance-aware selection, fault tolerance, and resource efficiency metrics to balance energy consumption with performance. The average, minimum, and maximum response times are used to measure how efficiently flow rules are computed and installed.

Scalability is assessed through latency, throughput, number of switches, and convergence time, reflecting the system's ability to handle network growth and dynamic changes.

Congestion is analyzed using throughput reduction and average congestion window size, where larger windows indicate efficient flow control and smaller ones reflect higher congestion and limited throughput.

3.5. Load Balance Algorithms

1. Q-Learning with Exploration-Exploitation Strategy

The ϵ-Greedy strategy (EGS) is commonly used in Q-Learning to balance exploration (trying new actions) and exploitation (choosing the best-known action).

The Q-value for a state-action pair (s,a) is updated using the simplified form in Eq. (4).

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \cdot \max Q(s',a') - Q(s,a)) \tag{4}$$

where α denotes the learning rate ($0 < \alpha \leq 1$), which controls the step size for updating the Q-values; γ represents the discount factor ($0 \leq \gamma < 1$), determining the importance of future rewards; r is the immediate reward received after taking action a in state s; s' is the next state reached; and $\max_{a'} Q(s',a')$ is the maximum estimated Q-value for the next state s' over all possible actions a'.

ϵ-Greedy Action Selection

At each step in Eq.(5), the agent chooses an action a as:

$$a = \begin{cases} \text{argmax}a\ Q(s,a) \text{with probability } 1 - \epsilon \\ \text{random } action\ with\ probability\ \varepsilon \end{cases} \tag{5}$$

A high value of ϵ (e.g., 0.9) encourages exploration by promoting more random actions, allowing the agent to discover new strategies. Conversely, a low value of ϵ (e.g., 0.1) encourages exploitation, where the agent relies more on its learned policy and selects fewer random actions, with ϵ decaying.

In Eq.(6), exploration is reduced gradually, and exploitation is favoured as learning progresses:

$$\epsilon \leftarrow \epsilon \times \text{decay rate} \tag{6}$$

2. Q-Learning with BE(Boltzmann Exploration)

Q-learning is a model-free RL algorithm in which an agent learns optimal actions through interaction with the environment and updates a Q-table. The table estimates long-term rewards for state-action pairs. The Boltzmann exploration strategy smoothly balances exploration and exploitation. State and Action Representation

In SDN-based load balancing, the state s is derived from the switch ID as given in Eq.(7):

$$s = \text{switch id mod num actions} \tag{7}$$

Here, num actions is the number of controllers, and the action a selects which controller to use.

Reward Mechanism

The reward r reflects how well the load is balanced—better distribution earns higher rewards.

Boltzmann Exploration Strategy

Instead of always picking the best-known action, Boltzmann exploration assigns a probability to each action in Eq.(8):

$$\text{P(a | s)} = \frac{e^{Q(s,a)/T}}{\sum_{a'} eQ(s,a')/T} \tag{8}$$

The temperature parameter T controls the balance between exploration and exploitation in action selection. A high value of T encourages exploration by producing more uniform action probabilities, whereas a low value of T favors actions with higher Q-values, leading to greater exploitation. Typically, T is gradually decreased (annealed) over time to shift the agent's behavior from exploration toward exploitation as learning progresses.

This approach helps in dynamic environments like SDN load balancing, where choosing the right controller improves network performance. Combining Q-learning with Boltzmann exploration supports adaptive and intelligent decision making [22–25].

3. Q-learning with PER(Prioritized Experience Replay)

PER improves learning by prioritizing transitions with high temporal difference (TD) errors, which speeds up convergence and enhances performance [26]. The priority of each transition is given from Eq.(9) to Eq.(11):

$$p_i = |\delta_i| + \epsilon \tag{9}$$

where $\delta_i = |r + \gamma \max_{a'} Q(s',a') - Q(s,a)|$. The sampling probability for a transition is given in Eq.(10):

$$\text{P(i)} = \frac{pi^{\alpha}}{\sum_k p_k^{\alpha}} \tag{10}$$

and the update weight is:

$$\text{wi} = (\frac{1}{N.P(i)})\beta \tag{11}$$

PER accelerates learning, handles rare events, and adapts dynamically [27]. Key Components of PER

Replay Buffer: Stores experiences in the form of (s,a,r,s'), prioritized by TD errors.

TD Error: Measures the "surprise" of an experience as given in Eq.(12):

$$\text{TD Error} = |\ \text{r}+ \gamma max Q(s', a') - Q(s, a)| \tag{12}$$

Sampling: Experiences with higher TD errors are sampled more frequently to improve learning efficiency. The sampling probability $P(i)$, defined in Equation 10, prioritizes transitions with larger TD errors. The exponent α controls the strength of prioritization, where $\alpha = 0$ corresponds to uniform sampling and $\alpha = 1$ applies full prioritization based on TD error magnitude.

4. Upper Confidence Bound (UCB) for Action Selection

The Upper Confidence Bound (UCB) method integrates an optimism-driven exploration bonus into Q-learning to balance exploration and exploitation [28]. The UCB value for action a in state s at step t is computed in Eq.(13):

$$\text{UCB(s,a)} = \text{Q(s,a)} + \text{c}\sqrt{\frac{\ln(t)}{Nt(a)}} \tag{13}$$

where $Q(s,a)$ is the current Q-value estimate, c is the exploration parameter (set to 2.0), t is the total number of steps, and $N_t(a)$ is the number of times action a has been selected. Action selection is performed as in Eq.(14):

$$a_t = \text{argmaxUCB}(s_t,a) \tag{14}$$

Ties are broken randomly among actions with equal maximum UCB values. The reward function combines load balancing and energy-awareness as described from Eq.(15) to Eq.(17):

$$\text{load reward} = \frac{10}{1+\frac{La}{Wa}} \tag{15}$$

$$\text{energy reward} = \frac{5}{1+energy\ consumption} \tag{16}$$

$$r_t = \text{load reward} + \text{energy reward} \tag{17}$$

The parameter c controls the exploration-exploitation tradeoff: higher values encourage exploration by favoring lesssampled actions, while lower values promote exploitation. As actions are sampled, the exploration term decreases, allowing Q-values to converge to the optimal policy under standard reinforcement learning conditions.

5. Soft Q-Learning (Maximum Entropy Q-Learning)

Soft Q-learning is a variant of Q-learning that incorporates entropy regularization into the value function, encouraging policies that are both reward-seeking and stochastic. Instead of maximizing expected return alone, soft Q-learning seeks to maximize both the expected cumulative reward and the entropy of the policy, leading to smoother, more exploratory behavior [29]. Soft Q-Learning extends standard Q-learning by adding an entropy term to encourage stochastic policies, enhancing exploration and robustness as described from Eq.(18) to Eq.(20).

Softmax Policy:

$$\pi(\text{a|s}) = \frac{\exp(Q(s,a)/\Gamma)}{\sum a\prime exp\ (\frac{Q(s,a)}{\Gamma})} \tag{18}$$

Temperature Decay:

$$\tau_t = \max(\tau_{\min}, \tau_{t-1} \times 0.995) \quad \text{with} \qquad \tau_0 = 1.0,\ \tau_{\min} = 0.1 \tag{19}$$

Policy Entropy:

$$H(\pi(\cdot|s)) = -^{\text{X}}\pi(a|s)\log\pi(a|s) \tag{20}$$

3.6. Performance Metrics

Latency: Latency measures the time taken to process a packet from its arrival at the switch to its forwarding to the destination as given in Eq.(21) [30].

$$\text{Latency} = t_{\text{end}} - t_{\text{start}} \tag{21}$$

Here, t_{start} denotes the timestamp when packet processing begins, while t_{end} represents the timestamp when packet processing completes.

Throughput: Throughput measures the number of packets processed per second. The throughput T is defined as the rate of successful packet delivery in the network which has been represented in Eq.(22):

$$\text{T} = \frac{R}{\Delta T} \tag{22}$$

Here, R represents the total number of packets received, and Δt denotes the elapsed time period.

Rewards: Rewards quantify the immediate benefit of taking an action in a given state. Here, rewards are computed based on load balancing efficiency and energy optimization which has been described in Eq.(23).

$$r_t = \alpha T_t - \beta \text{ Latency}_t + \gamma \text{ Fairness}_t - \delta \text{ Energy}_t \tag{23}$$

where T_t denotes the normalized throughput at time step t, Latency_t represents the normalized latency, Fairness_t is the normalized Jain's Fairness Index, and Energy_t denotes the normalized energy consumption. The weighting coefficients α, β, γ, and δ determine the relative importance of throughput, latency, fairness, and energy consumption, respectively, and satisfy the constraint in Eq.(24).

$$\alpha + \beta + \gamma + \delta = 1. \tag{24}$$

The average reward per epoch is given in Eq.(25):

$$\bar{R} = \frac{1}{N}\sum_{t=1}^{N}\left(\frac{10}{1+\frac{Lt}{Wa}} + \frac{5}{1+Et}\right) \tag{25}$$

Here, R denotes the average reward, N represents the total number of packet processing events, L_t indicates the controller load at time t, W_a is the controller's weight factor, and E_t represents the energy consumption at time t.

Load Distribution: Load distribution ensures the workload is evenly shared among controllers, maintaining optimal performance and system reliability. It prevents any single controller from becoming overwhelmed by reducing bottlenecks and improving resource utilization. The standard deviation of controller loads is calculated as given in Eq.(26):

$$\sigma_L = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Li - \mu L)^2} \tag{26}$$

where the mean load μ_L is given in Eq.(27):

$$\mu_L = \frac{1}{N}\sum_{i=1}^{N} L_i \tag{27}$$

Jain's Fairness Index: Jain's Fairness Index measures the fairness of load distribution across controllers. It ranges from $\frac{1}{n}$ (worst case) to 1 (best case), where n is the number of controllers as given in equation 28. A higher Jain's Index indicates a more balanced load distribution in Eq.(28).

$$\text{Jain's Index} = \frac{(\sum_{i=1}^{n} Load\ i)2}{n\ X\ \sum_{i=1}^{n} Load\ i)2} \tag{28}$$

Here, Load_i denotes the load on the i^{th} controller, and n represents the total number of controllers in the network.

3.7. Hyperparameter Tuning

Table 1 consists of 3 values of learning rate (α), exploration rate (ϵ) and reward (γ) which will be used for performing hyperparameter tuning. Best parameters will be identified using grid search.

Table 1. Hyperparameter Values

Learning Rate (α)	Exploration Rate (ϵ)	Reward (γ)
0.1	1.0	0.85
0.2	0.9	0.9
0.3	0.5	0.95
0.1	1.0	0.99

Table 1 outlines four distinct combinations of hyperparameters—learning rate (α), exploration rate (ϵ), and discount factor (γ)—that will be tested to determine the optimal configuration for a reinforcement learning model. The learning rate controls how quickly the model adapts to new information, the exploration rate balances random exploration versus exploiting known strategies, and the discount factor determines the importance of future rewards. These combinations will be evaluated based on key performance metrics.

3.8. Controller Metrics

Controller Response Time: The time taken by a controller to process a packet-in request and send back a flow-mod rule.

For each controller *i*, the response time for the *k*-th request is given in Eq.(29):

$$\mathrm{RT_i^{(k)}} = \mathrm{t^{(k)}_{processing}} + \mathrm{t^{(k)}_{network}} \tag{29}$$

where:

- $t_{\text{processing}}$ = Time taken by the controller to compute and install rules.
- t_{network} = Network propagation delay.

Average Response Time for controller *i* over *N* requests as given in Eq.(30) along with System-wide Average Response Time in Eq.(31):

$$\overline{RT_i} = \frac{1}{N}\sum_{k=1}^{N} RT_i^{(k)} \tag{30}$$

System-wide Average Response Time

$$\overline{\mathrm{RT}}_{\text{system}} = \frac{1}{M}\sum_{i=1}^{M} \overline{RT}_{\mathrm{I}} \tag{31}$$

where M = number of controllers.

Minimum Response Time (R_{min}): The minimum response time is the lowest possible latency experienced in an SDN control loop under ideal conditions, where no congestion, queuing, or additional delays are present. The minimum response time is given in Eq.(32):

$$R_{\text{min}} = T_{\text{proc min}} + T_{\text{net min}} + T_{\text{ovh_min}} \tag{32}$$

where $T_{\text{proc min}}$ is the controller's minimal processing delay (hardware-dependent), $T_{\text{net min}}$ the propagation delay under no congestion, and $T_{\text{ovh min}}$ the inherent OpenFlow protocol overhead.

Maximum Response Time (R_{max}): The maximum response time is the worst-case latency under heavy load, incorporating queuing delays and congestion effects. It extends the minimum response time with additional delays.

Under load, queuing and congestion dominate the maximum response time which has been mentioned in Eq. (33):

$$R_{max} = T_{proc,max} + \frac{\frac{\lambda}{\mu}}{\mu-\lambda} + T_{net_{max}} + T_{ovh_max} \tag{33}$$

Eq.(33) integrates the M/M/1 queuing model into the total maximum response time R_{max}, where the term $\frac{\frac{\lambda}{\mu}}{\mu-\lambda}$ captures the queuing delay, which grows sharply as the arrival rate λ approaches the service rate μ, while $T_{\text{proc_max}}$, $T_{\text{net_max}}$, and $T_{\text{ovh max}}$ account for processing, network, and overhead delays, respectively, providing a comprehensive bound for SDN control plane latency analysis and QoS assurance.

3.9. Scalability in SDN

The proposed SDN load balancer exhibits excellent scalability properties with linear complexity growth. The time and space complexity are bounded as given in Eq.(34) and Eq.(35):

$$T_{\text{select}} = \mathrm{O}(n) \tag{34}$$

$$M_{\text{Q-table}} = \mathrm{O}(m \times n) \tag{35}$$

where n is the number of controllers and m represents the state space size. The state space grows combinatorially as described in Eq.(36):

$$|\mathrm{S}| = s \times l \times d \times w \tag{36}$$

accounting for network switches (s), load levels (l), load differences (d), and weight parameters (w). The decision latency combines heuristic and UCB components as given in Eq.(37):

$$T_{\text{dec}} = P_h T_h + (1 - P_h) T_{\text{UCB}} \tag{37}$$

The system maintains O(n) operational complexity while ensuring efficient load distribution through its hybrid heuristicUCB approach, making it suitable for large-scale SDN deployments.

3.10. Congestion Scenario

SDN allows dynamic traffic management but is prone to congestion, which degrades QoS. The proposed framework addresses this by using multi-dimensional congestion metrics [31]. It begins with flow rate monitoring, where each flow's transmission rate—measured by packet count over time—indicates its aggressiveness. Higher rates suggest potential congestion contributors. Congestion is identified when the combined rate of all flows exceeds a predefined threshold. Algorithm 1 below outlines congestion metrics in multicontroller SDN.

Algorithm 1 Congestion Control

1: Input: Flow rates R, congestion threshold T_{cong}, queue occupancy Q
2: Output: Congestion detection flag, adjusted actions
3: Initialize: *congestion detected* ← *False*, *congestion window*[f] ← 10, $\forall f$
4: procedure DETECTCONGESTION

5:*total _rate* ← $^{P}{}_{f}R[f]$ ▷ Mean rate across all flows
6: if *total _rate* > T_{cong} *and not congestion_detected* then
7: *congestion_event_count* ← *congestion_event_count+1*
8: ->*congestion start time* ← now()
9: ->*congestion start throughput* ← throughput()

10: *flow contribution*[f] ← $R[f]$/*total rate,* $\forall f$
11: end if
12: congestion_detected -> (total_rate > T $_{\text{cong}}$)
13: overload <- min(1, $\frac{\max(0, total_{rate} - Tcong)}{Tcong}$)
14: Q <- max(0,min(100,Q+5 x overload)))

15: end procedure
16: procedure CONTROLACTION(a,f) ▷ Optimized action and window update
17: $\beta \leftarrow \begin{cases} 0.8 & \text{if } R[f] > T_{cong}/|R| \\ 0.9 & \text{otherwise} \end{cases}$
18: *congestion window*[f] ← max(1, *congestion window*[f] × β)
19:return (a + $\mathbf{1}_{\beta=0.8}$) mod N ▷ $\mathbf{1}$ is the indicator function
20: end procedure
21: function GETDELAY(f)
22:*delay* ← U(0.001,0.01) ▷ Uniform random delay
23:if congestion_detected then
24: delay -> delay X (1+$\frac{Q}{200}$)
25: end if 26: return *delay* 27: end function

4. Experimental Results

4.1. Multi-controller Load balancing Evaluation

Table 2. Best Hyperparameter Values

Algorithms	Learning Rate (α)	Exploration Rate (ϵ)	Reward (γ)
BE	0.1	1.0	0.85
PER	0.1	1.0	0.99
EGS	0.2	0.9	0.9
UCB	0.3	-	0.5
SOFT	0.3	-	0.95

The optimized hyperparameters from Table 2 reveal key algorithmic differences: PER's high discount factor (γ = 0.99) emphasizes long-term rewards, while EGS's balanced configuration (α = 0.2, ϵ = 0.9, γ = 0.9) explains its superior response time performance, demonstrating the critical role of hyperparameter selection in reinforcement learning

efficiency. Meanwhile, UCB and SOFT operate without an explicit exploration rate, relying on their inherent selection strategies, and favoring either immediate or future rewards depending on their γ values. These differences underscore the significant impact of hyperparameter tuning on reinforcement learning efficiency and task-specific performance.

Table 3. Performance Metrics for Network Controllers

Metric	BE	PER	EGS	UCB	SOFT
Throughput	5.5337	3.5607	7.8800	2.5669	5.5277
Average Latency	0.0447	0.0303	0.0221	0.0213	0.0115
Load Balance Ratio	0.9865	0.8465	0.9816	0.9791	0.9331
Load Distribution Fairness	0.9998	0.9741	0.9999	0.9993	0.9947

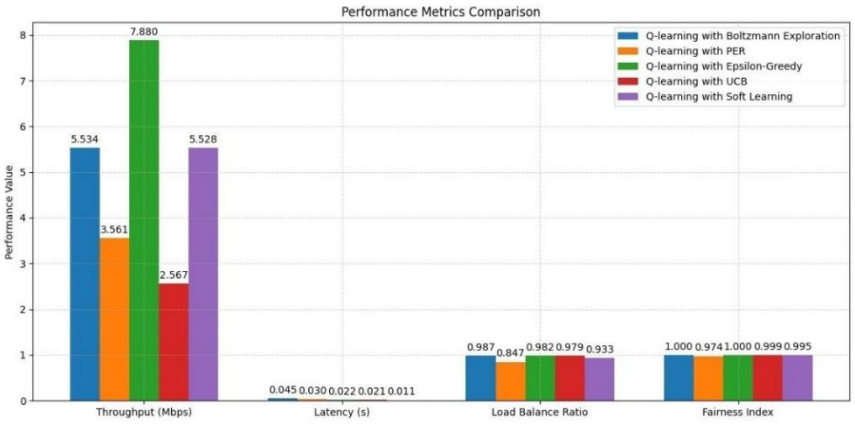


Fig. 3. Performance metrics plot

From Table 3 and Figure 3, EGS (Epsilon-Greedy Strategy) achieves the highest throughput (7.88 Mbps), lowest latency (0.022 s), and perfect fairness (1.0), reflecting efficient and fair resource use. Although BE (Boltzmann Exploration) slightly surpasses EGS in load balance ratio (0.987 vs. 0.982) while maintaining perfect fairness and competitive throughput (5.534 Mbps), it offers a balanced trade-off between throughput and stability. PER (Prioritized Experience Replay) shows acceptable latency (0.030 s) but falls behind in throughput (3.561 Mbps) and load balance (0.847), reducing its effectiveness. UCB and SOFT perform well in latency and fairness but lag in throughput compared to EGS and BE. Overall, BE suits scenarios needing balanced metrics, while EGS is optimal for maximum performance.

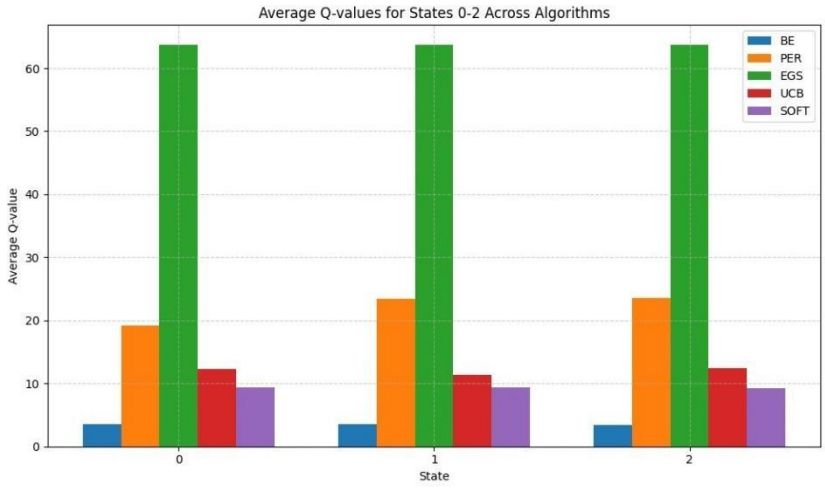


Fig. 4. Q-learning plot across controllers

Figure 4 compares Q-values across five reinforcement learning algorithms. EGS yields the highest Q-values (57–66), excelling in reward maximization. PER (18–24) balances learning with prioritized sampling, while Boltzmann (≈ 3.3–3.6) emphasizes stable, cautious exploration. UCB shows a broad range (≈ 0.9–13+), reflecting uncertainty-driven adaptability, and SOFT offers consistent, moderate values just below UCB. The best choice depends on the environment and the desired exploration–exploitation balance.

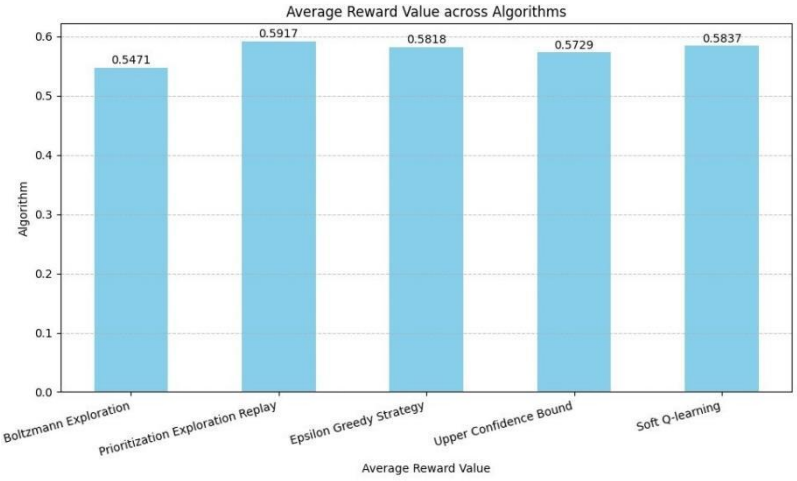


Fig. 5. Comparison of Reward value per epoch

Figure 5 shows the average reward over 20 epochs for five exploration strategies: BE, PER, EGS, UCB, and SOFT. All start with similar rewards (0.45–0.55), but by the final epoch, BE (0.5917), SOFT (0.5837), and PER (0.5818) outperform UCB (0.5729) and EGS (0.5471). BE and SOFT show more stable and consistent improvement, highlighting their effectiveness in balancing exploration and exploitation. Overall, advanced strategies (BE, PER, SOFT) yield better rewards than simpler ones (EGS, UCB).

Table 4. Average Response time for each controller

Algorithms	Controller 0	Controller 1	Controller 2
BE	0.0639	0.0659	0.0673
PER	0.0800	0.0786	0.0811
EGS	0.0343	0.0332	0.0342
UCB	0.0262	0.0263	0.0330
SOFT	0.0267	0.0302	0.0325

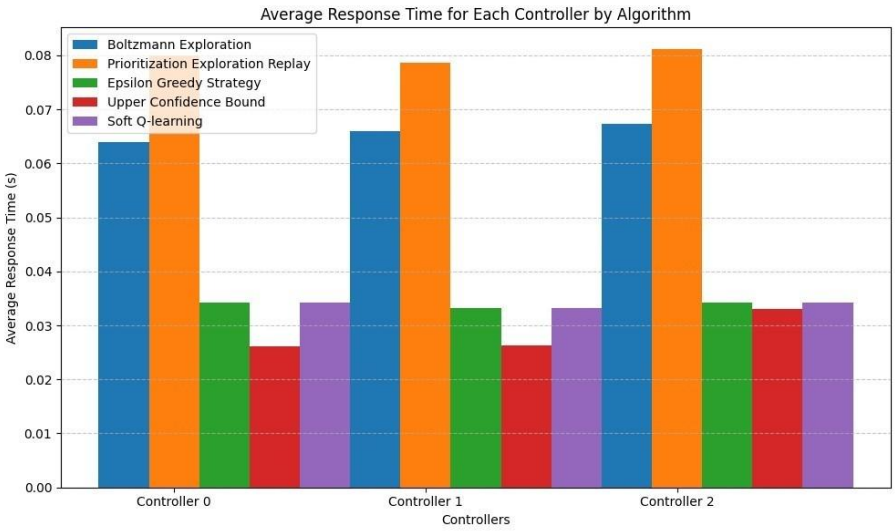


Fig. 6. Controller Average Response Time

Table 4 and Figure 6 present the average response times for Controllers 0, 1, and 2 under five learning algorithms. The UCB algorithm achieves the lowest average response times across all controllers, with values of 0.0262 s, 0.0263 s, and 0.0330 s, respectively, indicating efficient and consistent performance. The SOFT algorithm performs comparably, particularly on Controller 2 (0.0325 s). In contrast, PER and BE exhibit the highest average delays, exceeding 0.08 s in several instances, reflecting less efficient control decision latency. These results confirm UCB's suitability for latency-sensitive environments.

Table 5. Minimum Response time for each controller

Algorithms	Controller 0	Controller 1	Controller 2
BE	0.0024	0.0052	0.0047
PER	0.0027	0.0036	0.0058
EGS	0.0020	0.0028	0.0043
UCB	0.0019	0.0027	0.0042
SOFT	0.0022	0.0041	0.0054

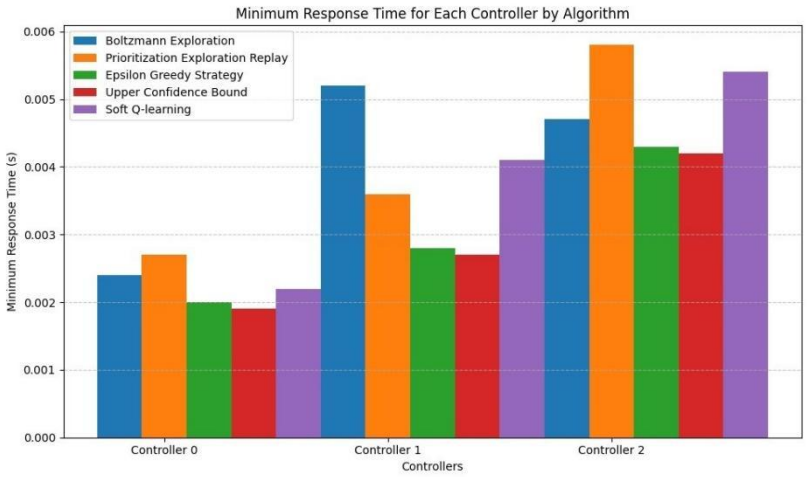


Fig. 7. Controller Minimum Response Time

Table 4 and Figure 7 report the minimum response times (in seconds) for Controllers 0, 1, and 2 under five decision-making algorithms: BE, PER, EGS, UCB, and SOFT. Among these, the UCB algorithm consistently yields the lowest response times across all controllers—0.0019 s, 0.0027 s, and 0.0042 s—demonstrating its efficiency in balancing exploration and exploitation. In contrast, PER and SOFT exhibit comparatively higher latencies, particularly on Controller 2, with 0.0058 s and 0.0054 s respectively, indicating suboptimal responsiveness under these strategies.

Table 6. Maximum Response time for each controller

Algorithms	Controller 0	Controller 1	Controller 2
BE	0.2958	0.8012	0.3481
PER	0.4276	0.4100	0.6259
EGS	0.3923	0.3740	0.3624
UCB	0.3615	0.3496	0.3332
SOFT	0.3263	0.2232	0.3619

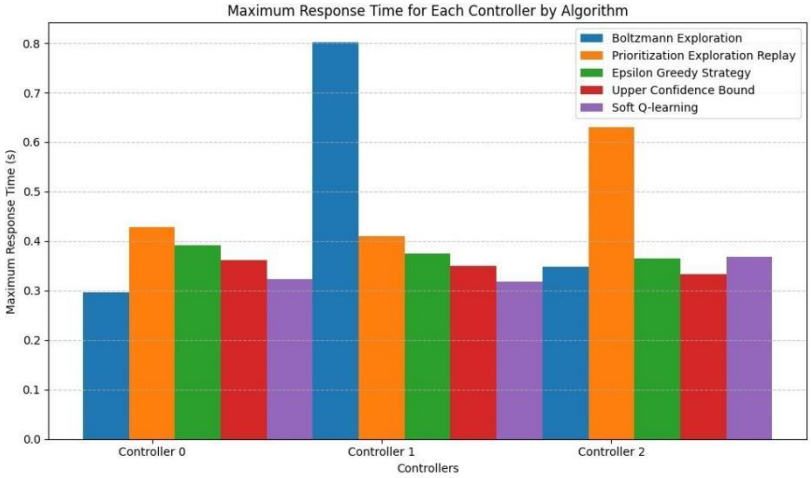


Fig. 8. Controller Maximum Response Time

Table 6 and Figure 8 present the maximum response times for each controller under five algorithms. The BE algorithm records the highest response time overall, reaching 0.8012 s on Controller 1, indicating potential instability or delayed processing. In contrast, SOFT achieves the lowest maximum response on Controller 1 (0.2232 s), while UCB maintains relatively low maximum delays across all controllers (0.3615 s, 0.3496 s, and 0.3332 s), suggesting improved consistency and better worst-case performance compared to other strategies.

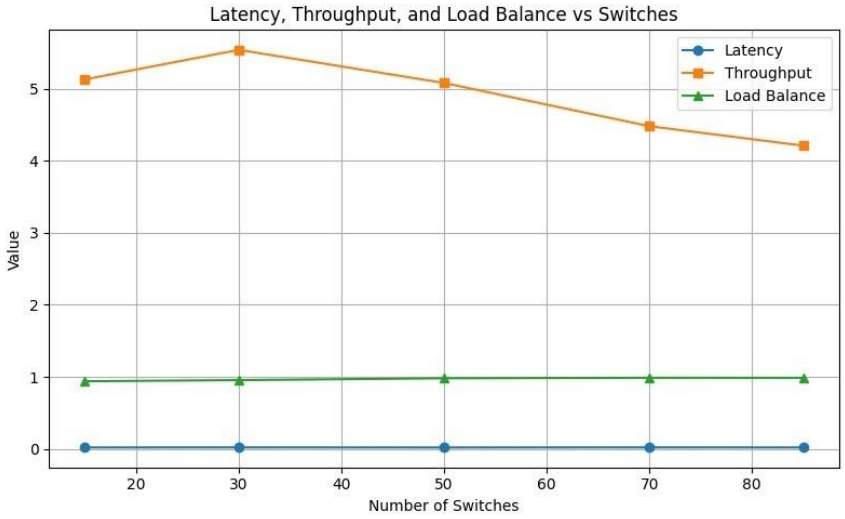


Fig. 9. Latency, Throughput and Load Balance vs Switches (Scalability)

Figure 9 depicts how switch count affects SDN performance. Latency stays constant, indicating efficient control communication. Throughput rises up to 30 switches, then declines due to overhead, while load balance steadily improves. This suggests that moderate switch counts optimize performance, balancing scalability and efficiency.

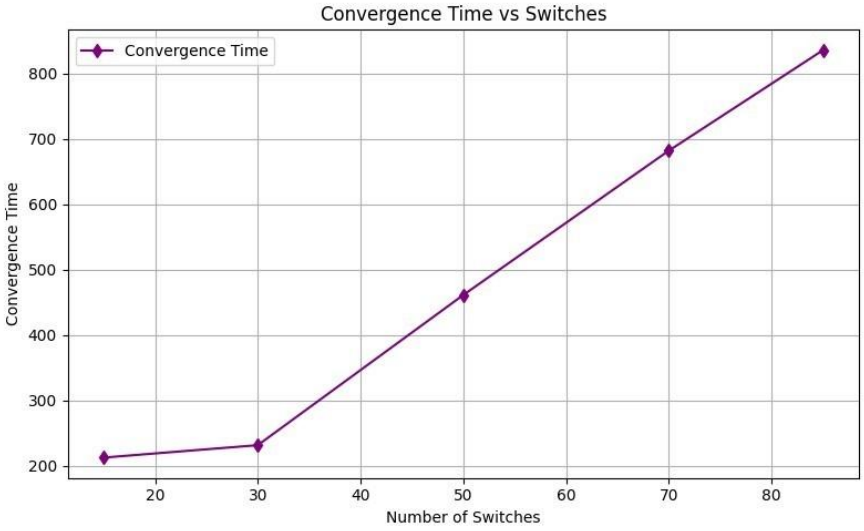


Fig. 10. Convergence Time

Figure 10 describes that SDN convergence time increases sharply with more switches, highlighting scalability limits due to rising control overhead. This trend underscores the need for optimization strategies like efficient controller placement and load balancing in large-scale networks.

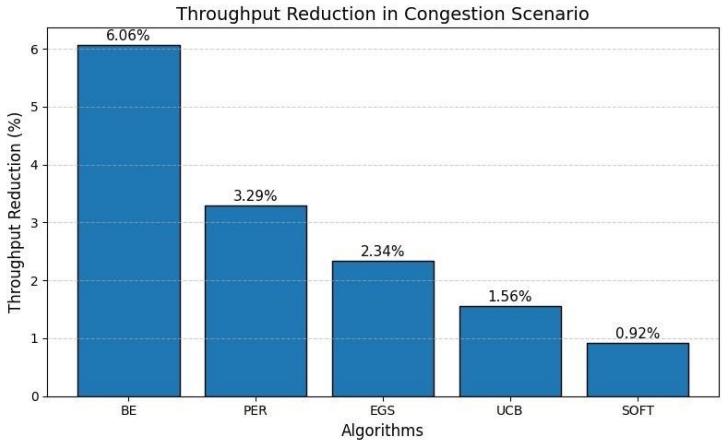


Fig. 11. Throughput Reduction

Table 7. Throughput Reduction in Congestion Scenario

Algorithms	Throughput Reduction
BE	6.06 %
PER	3.29 %
EGS	2.34 %
UCB	1.56 %
SOFT	0.92 %

In a congested SDN environment, the results presented in Fig. 11 and Table 7 highlight the differences in throughput reduction among the evaluated algorithms. The SOFT algorithm achieves the lowest throughput reduction of 0.92%, demonstrating the strongest capability to maintain network throughput under congested conditions. The UCB algorithm follows with a throughput reduction of 1.56%, while EGS records 2.34%, indicating effective throughput preservation. PER exhibits a throughput reduction of 3.29%, showing moderate performance under congestion. In contrast, the BE algorithm experiences the highest throughput reduction of 6.06%, reflecting the weakest ability to sustain network throughput during congested conditions. Overall, the results indicate that SOFT is the most effective algorithm in minimizing throughput degradation, followed by UCB, EGS, and PER, whereas BE demonstrates the poorest throughput preservation performance under congestion.

Table 8. Average Window Size in Congestion Scenario

Algorithms	Controller 0	Controller 1	Controller 2
BE	1.31	1.34	1.56
PER	1.29	1.39	1.53
EGS	1.45	1.39	1.66
UCB	1.23	1.43	1.34
SOFT	1.30	1.21	1.49

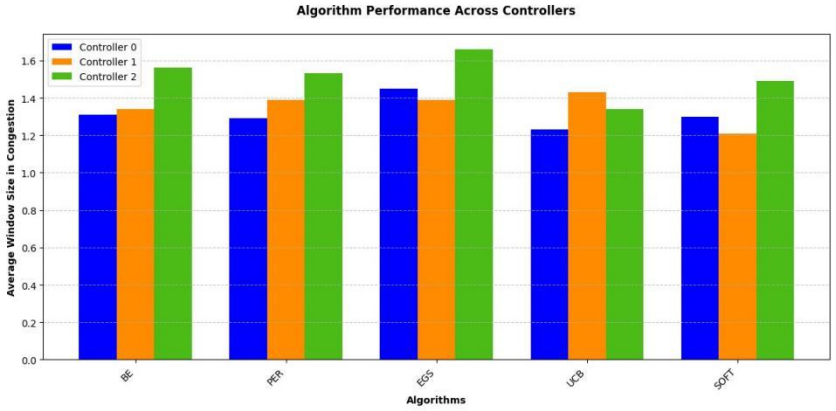


Fig. 12. Average Window Size

The bar chart in Fig. 12 and Table 8 present the average window size during congestion for five algorithms—BE, PER, EGS, UCB, and SOFT—across three controllers: Controller 0, Controller 1, and Controller 2. It is observed that Controller 2 consistently exhibits the highest average window size across all algorithms, indicating a higher level of congestion. Conversely, Controllers 0 and 1 demonstrate relatively lower and more balanced congestion levels. Among the algorithms, EGS results in the highest congestion, particularly under Controller 2, whereas SOFT under Controller 1 shows the lowest congestion, highlighting its efficiency. These findings suggest that Controller 2 may benefit from additional optimization or load balancing, while algorithms such as SOFT and UCB appear more effective in managing congestion.

4.2. Comparative Analysis

Table 9. Comprehensive Performance Comparison of Algorithms

Metric	BE	PER	EGS	UCB	SOFT
Throughput	5.53	3.56	7.88	2.57	5.53
T.Reduction (%)	6.06	3.29	2.34	1.56	0.92
Avg Window Size	1.40	1.40	1.50	1.33	1.33
Average Latency	0.0447	0.0303	0.0221	0.0213	0.0115
Avg C. Response	0.0657	0.0799	0.0339	0.0285	0.0298
Min C. Response	0.0041	0.0040	0.0030	0.0029	0.0039
Max C. Response	0.4817	0.4885	0.3762	0.3481	0.3038
Load Balance Ratio	0.9865	0.8465	0.9816	0.9791	0.9331
Load Fairness	0.9998	0.9741	0.9999	0.9993	0.9947

Table 9 presents a comprehensive comparison of five algorithms (BE, PER, EGS, UCB, and SOFT) across throughput, traffic reduction, latency, controller response, load balancing, and fairness metrics. EGS achieves the highest throughput (7.88) and the best load fairness (0.9999), demonstrating its effectiveness in maximizing network performance while maintaining equitable load distribution. BE provides the highest traffic reduction (6.06%) and the best load balance ratio (0.9865), indicating superior capability in reducing network traffic and distributing workload evenly across controllers.

SOFT attains the lowest average latency (0.0115 s) and the lowest maximum controller response time (0.3038 s), making it the most efficient in terms of response delay. UCB records the lowest average controller response time (0.0285 s) and minimum controller response time (0.0029 s), reflecting its ability to provide rapid controller responses under favorable conditions. Overall, EGS offers the best trade-off between throughput and fairness, BE excels in traffic reduction and load balancing, SOFT minimizes latency, and UCB delivers the fastest controller response, highlighting the complementary strengths of the evaluated algorithms.

Table 10. Ranking of Algorithms (1 = Best, 5 = Worst) Across Metrics

Metric	BE	PER	EGS	UCB	SOFT
Throughput	3	4	1	5	3
Average Latency	5	4	3	2	1
Load Balance Ratio	1	5	2	3	4
Load Fairness	2	5	1	3	4
Avg Response	4	5	3	1	2
Min Response	4	3	2	1	5
Max Response	3	4	2	1	1
Throughput Reduction	5	4	3	2	1
Window Size (Congestion)	3	3	1	4	4

Table 10 ranks five algorithms (BE, PER, EGS, UCB, SOFT) across key metrics, with 1 as best and 5 as worst. EGS leads in throughput, load fairness, and congestion control (window size). SOFT excels in average latency and throughput reduction. UCB consistently ranks best in controller response times, while BE shows superior load balancing. PER generally ranks lowest across most metrics. These results highlight EGS, SOFT, and UCB as top performers in different aspects, with BE strong in load distribution and PER lagging overall.

5. Conclusion and Recommendations

This study compared five Q-learning exploration strategies—Boltzmann Exploration (BE), Prioritized Experience Replay (PER), Epsilon-Greedy Strategy (EGS), Upper Confidence Bound (UCB), and Soft Q-learning (SOFT)—for load balancing in multicontroller SDN using throughput, latency, response time, fairness, scalability, and congestion resilience. EGS achieved the best overall performance with the highest throughput (7.88), fairness index (0.9999), and congestion resilience. SOFT provided the lowest latency (0.0115s), UCB achieved the fastest response time (0.0285s), and BE attained the highest load balance ratio (0.9865). These findings demonstrate that exploration strategy selection significantly affects Q-learning-based load balancing performance. Hybrid approaches such as Deep Reinforcement Learning (DRL), transformer-based models, and large-scale SDN-enabled IoT deployments can be recommended to further enhance scalability, adaptability, and efficiency.

All the Declarations and Statements

Author Contributions Statement

Binod Sapkota – Conceptualization, Literature review, Methodology, Experimental analysis, manuscript writing
Utkarsha Shukla – Methodology, Experimental analysis, manuscript writing
Babu R. Dawadi –Methodolody, manuscript review, Supervision.
Shashidhar R. Joshi –Methodolody, Manuscript review, Supervision

All authors have read and agreed to the published version of the manuscript.

Conflict of Interest Statement

The authors declare no conflicts of interest.

Funding Declaration

This research received no external funding, grant, or financial support from any organization.

Data Availability Statement

The dataset was simulated and generated from Ryu Controller in Software Defined Networking.

Ethical Declarations

The authors should declare the following: Statements of ethical approval for studies involving human subjects and/or animals.

Acknowledgments

We sincerely thank the experts for their professional evaluation and valuable recommendations, which have contributed to improving the quality of the experiment and the reliability of its results.

Declaration of Generative AI in Scholarly Writing

https://www.mecs-press.org/ijcnis/author.html#DeclarationofgenerativeAIinscholarlywriting
During the preparation of this manuscript, the authors used generative AI tools to improve language, grammar, and readability. The authors carefully reviewed, verified, and edited all AI-assisted content and take full responsibility for the accuracy, originality, and integrity of the final manuscript. No AI tool was used for data collection, experimentation, analysis, interpretation of results, or drawing scientific conclusions.

Abbreviations

None.

Appendix A/B/C…, with appendix title

None.

References

[1] A. H. Abdi., L. Audah., A. Salh., M. A. Alhartomi., H. Rasheed., S. Ahmed., & A. Tahir. (2024). Security control and data planes of SDN: A comprehensive review of traditional, AI, and MTD approaches to security solutions. *IEEE Access*. DOI: https://doi.org/10.1109/ACCESS.2024.3393548

[2] O. Tkachova., U. Chinaobi., & A. R. Yahya. (2016). A load balancing algorithm for SDN. *Scholars Journal of Engineering and Technology (SJET)*. DOI: https://doi.org/10.21276/sjet.2016.4.11.2

[3] G. Beissenova., A. Zhidebayeva., Z. Kopzhassarova., P. Kozhabekova., B. Myrzakhmetova., M. Kerimbekov., D. Ussipbekova., & N. Yeshenkozhaev. (2024). *Load balancing in DCN servers through software defined network machine learning***.** International Journal of Advanced Computer Science and Applications (IJACSA)**.** DOI: https://doi.org/10.14569/IJACSA.2024.0150254

[4] S. Raghul., T. Subashri., & K. R. Vimal. (2017). Literature survey on traffic-based server load balancing using SDN and OpenFlow. *2017 4th International Conference on Signal Processing, Communications and Networking (ICSCN)*. DOI: https://doi.org/10.1109/ICSCN.2017.8085416

[5] M. Xiang., M. Chen., D. Wang., & Z. Luo. (2022). Deep reinforcement learning-based load balancing strategy for multiple controllers in SDN. *e-Prime – Advances in Electrical Engineering, Electronics and Energy*. DOI: https://doi.org/10.1016/j.prime.2022.100038

[6] J. Ali., R. H. Jhaveri., M. Alswailim., & B.-H. Roh. (2023). Escalb: An effective slave controller allocation-based load balancing scheme for multi-domain SDN-enabled-IoT networks. *Journal of King Saud University – Computer and Information Sciences*. DOI: https://doi.org/10.1016/j.jksuci.2023.101566

[7] W. Negera., F. Schwenker., T. Debelee., H. Melaku., & Y. Ayano. (2022). Review of botnet attack detection in SDN-enabled IoT using machine learning. *Sensors*. DOI: https://doi.org/10.3390/s22249837

[8] M. Zolotukhin., S. Kumar., & T. Hämäläinen. (2020). Reinforcement learning for attack mitigation in SDN-enabled networks. *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. DOI: https://doi.org/10.1109/NetSoft48620.2020.9165383

[9] P. Sun., Z. Guo., G. Wang., J. Lan., & Y. Hu. (2020). Marvel: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning. *Computer Networks*. DOI: https://doi.org/10.1016/j.comnet.2020.107230

[10] V. Srivastava., & R. S. Pandey. (2021). Machine intelligence approach: To solve load balancing problem with high quality of service performance for multi-controller based software-defined network. Sustainable Computing: Informatics and Systems. DOI: https://doi.org/10.1016/j.suscom.2021.100511

[11] V. Tosounidis., G. Pavlidis., & I. Sakellariou. (2020). Deep Q-learning for load balancing traffic in SDN networks. *11th Hellenic Conference on Artificial Intelligence*. DOI: https://doi.org/10.1145/3411408.3411423

[12] D. Tennakoon., S. Karunarathna., & B. Udugama. (2018). Q-learning approach for load-balancing in software-defined networks. *2018 Moratuwa Engineering Research Conference (MERCon)*.

[13] J. Chen, Y. Wang, J. Ou, C. Fan, X. Lu, C. Liao, X. Huang, and H. Zhang, "Albrl: Automatic load-balancing architecture based on reinforcement learning in software-defined networking," *Wireless Communications and Mobile Computing*, vol. 2022, pp. 1–17, 2022, open Access

[14] S. Liang., W. Jiang., F. Zhao., & F. Zhao. (2020). Load balancing algorithm of controller based on SDN architecture under machine learning. *Journal of Systems Science and Information*. DOI: https://doi.org/10.21078/JSSI-2020-578-11

[15] U. Ahmed., J. C.-W. Lin., & G. Srivastava. (2022). A resource allocation deep active learning based on load balancer for network intrusion detection in SDN sensors. *Computer Communications*. DOI: https://doi.org/10.1016/j.comcom.2021.12.009

[16] S. Yeo., Y. Naing., T. Kim., & S. Oh. (2021). Achieving balanced load distribution with reinforcement learning-based switch migration in distributed SDN controllers. *Electronics*. DOI: https://doi.org/10.3390/electronics10020162

[17] M. D. Tache., O. Pascutoiu., & E. Borcoci. (2024). Optimization algorithms in SDN: Routing, load balancing, and delay optimization. *Applied Sciences*. DOI: https://doi.org/10.3390/app14145967

[18] T. Semong., T. Maupong., S. Anokye., K. Kehulakae., S. Dimakatso., G. Boipelo., & S. Sarefo. (2020). Intelligent load balancing techniques in software-defined networks: A survey. *Electronics*. DOI: https://doi.org/10.3390/electronics9071091

[19] A. Kumar., & D. Anand. (2021). Load balancing for software-defined network using machine learning. *Turkish Journal of Computer and Mathematics Education*.

[20] H. Wang., H. Xu., L. Huang., J. Wang., & X. Yang. (2018). Load-balancing routing in software-defined networks with multiple controllers. *Computer Networks*. DOI: https://doi.org/10.1016/j.comnet.2018.05.012

[21] J. Singh., P. Singh., E. M. Amhoud., & M. Hedabou. (2022). Energy-efficient and secure load balancing technique for SDN-enabled fog computing. *Sustainability*. DOI: https://doi.org/10.3390/su141912951

[22] S. M. Kerner. What is q-learning? [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/Qlearning

[23] Q-learning guide: Beginwith reinforcement learning basics. Simplilearn. [Online]. Available: https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-q-learning

[24] A. Saxena. (2024). Q-learning in machine learning: Explained by experts. *Applied AI Course*. Available: https://www.appliedaicourse.com/blog/q-learning-in-machine-learning/

[25] R. S. Sutton., & A. G. Barto. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). *MIT Press*.

[26] D. Takeshi. (2019). Understanding prioritized experience replay. *Seita's Place*. Available: https://danieltakeshi.github.io/2019/07/14/per/

[27] T. Schaul., J. Quan., I. Antonoglou., & D. Silver. (2016). Prioritized experience replay. *arXiv*. Available: https://arxiv.org/abs/1511.05952v4

[28] J.-S. Boon., & X. Zhang. (2022). Lower PAC bound on upper confidence bound-based Q-learning with examples. *University of Wisconsin–Madison*. Available: https://pages.cs.wisc.edu/~xiaominz/projs/cs761.pdf

[29] T. Haarnoja., H. Tang., P. Abbeel., & S. Levine. (2017). Reinforcement learning with deep energy-based policies. *arXiv*. Available: https://arxiv.org/abs/1702.08165

[30] M. Goodwin. (2023). What is latency? *IBM*. Available: https://www.ibm.com/think/topics/latency

[31] A. M. Abdelmoniem., B. Bensaou., & A. J. Abu. (2017). SICC: SDN-based incast congestion control for data centers. *Proceedings of the IEEE International Conference on Communications (ICC)*. DOI: https://doi.org/10.1109/ICC.2017.7996826

Authors' Profiles



Dr. Binod Sapkota earned BE in Electronics and Communication Engineering (2009), MSc in Information System Engineering (2013), and PhD in Computer Engineering (2026) from the Institute of engineering (IoE), Tribhuvan University (TU), Nepal. Since 2014, he has served as an Assistant Professor in the Department of Electronics and Computer Engineering, Thapathali Campus, IoE, TU. He teaches electronics and computer engineering courses and supervises bachelor's and graduate projects and theses. His research interests include next-generation networking, software-defined networking, AI/ML, cybersecurity, and digital forensics. He has many research papers in the international indexed journals and published books of computer Science and Engineering.



Utkarsha Shukla received his B.Tech in Computer Science and Engineering from KL University (KL Deemed to be University) in 2019. He received his MSc. in Informatics and Intelligent Systems Engineering from Institute of Engineering, Thapathali Campus, Tribhuvan University in 2025. He teaches computer science related courses at bachelors' level, His research interest areas include Software Defined Networking, Natural Language Processing, Machine Learning, Cyber Security, and Digital Forensics.



Dr. Babu R. Dawadi received his B.E. in Computer Engineering (2003), M.Sc. in Information and Communication Engineering (2008), and Ph.D. in Computer Engineering (2021) from the Institute of Engineering (IOE), Tribhuvan University, Nepal. He also earned a master's degree in public administration from Tribhuvan University in 2013. He is currently Co-Chair of the Laboratory for ICT Research and Development (LICT), and Director of the Network, Cyber Security and Digital Forensic Lab. His research interests include computer networks, IPv6, cybersecurity, digital forensics, AI, and ICT policy. He has served in several academic and administrative leadership roles at IOE and previously worked at the Nepal Telecommunications Authority and as an ICT consultant for national and international organizations. He is a recipient of the NAST Best Ph.D. Fellow Award. He has many research papers in the international indexed journals and also published many books of computer Science.



Prof. Dr. Shashidhar R. Joshi received BE in Electrical Engineering degree (Hons.) from Sardar Vallabhbhai Regional College of Engineering, South Gujarat University, Surat, India, in 1984, MSc in electrical engineering from the University of Calgary, Canada, in 1992, and the PhD degree from the Institute of engineering (IoE), Tribhuvan University (TU), Nepal, in 2007. He was a Research Fellow with Osaka Sangyo University, Japan, from 1998 to 1999. He joined the Institute of Engineering in 1985. He is currently a Full Professor in the Department of Electronics and Computer Engineering, IoE, Pulchowk Campus, and was a Visiting Professor with South Asian University, New Delhi, India. He is the former Dean of the IoE, TU. His research interests include image processing, networking, and artificial intelligence.

How to cite this paper

Binod Sapkota, Utkarsha Shukla, Babu R. Dawadi, Shashidhar R. Joshi, "Comparative Study of Exploration Strategies in Q- Learning for Multi-Controller SDN Load Balancing", International Journal of Wireless and Microwave Technologies(IJWMT), Vol.16, No.5, pp. 1-19, 2026. DOI:10.5815/ijwmt.2026.05.01