

New Learning Approach for High-Load Traffic Optimization SDN

Mohammad Khalid¹, Hassan Mohamed Muhi-Aldeen^{1,*}, Basma Rashid Mahdi Alhamdani²

¹Department of Computer Engineering, Aliraqia University, 22 Sabaabkar, Adamia, Baghdad, 10053, Iraq

²Centre for Developing and Continuous Education, Aliraqia University, 22 Sabaabkar, Adamia, Baghdad, 10053, Iraq

Emails: mohammad.k.abass@aliraqia.edu.iq; muhialdeen.hassan@aliraqia.edu.iq;
basma_alhamdani@aliraqia.edu.iq

Abstract

Due to the Internet's growing importance in our lives, Software-Defined Networking (SDN) networks have experienced high load traffic issues. Thus, network load has increased, lowering quality of service (Qos) performance. Modern networked systems depend on communication channels to transmit data between sources and destinations. High traffic loads exacerbate packet distribution inefficiencies, causing network congestion in specific channels, compromising these communication channels. Congestion delays packet delivery and generates significant packet loss, reducing network dependability and efficiency. Communication channels' improper packet allocation along accessible paths is the fundamental issue. Some paths are overcrowded during peak traffic, while others are underused. Bottlenecks slow packet transit and increase packet loss due to this imbalance. Current packet distribution techniques don't adapt effectively to dynamic traffic, resulting in poor network performance. Current traffic management solutions often rely on load balancing algorithms, but these methods may not adequately account for the dynamic and unpredictable nature of high-load traffic. This paper introduces Adaptive Load Balancing using Reinforcement Learning (ALBRL), which uses Q-learning and deep reinforcement learning to distribute traffic in real time in SDNs with high traffic loads. This model uses more network-specific indicators including packet loss ratio, latency, Jitter, and traffic pattern history to improve decision-making. ALBRL outperformed static routing and Q-learning with 15.34(ms) average delay, 2.11(ms) jitter, and 7.89% packet loss ratio.

Received: January 31, 2025 Revised: March 09, 2025 Accepted: April 27, 2025

Keywords: Software-Defined Networking; Load Balancing; Q-learning; High-Load Traffic; Deep Q-learning; Deep Reinforcement Learning

1. Introduction

In current growing dynamics of Software-Defined Networking (SDN) a proper utilization of the available network resources, become mandatory for achieving the increasing service and application requirements [1]. Load balancing, which forms the core of network optimization, occupies a central position of distributing traffic among various paths in a manner that increases Qos levels [2]. As the network architectures migrate from the physical to the virtual and SDN environment [3], the issues of intelligent and adaptive load balancing tend to surface[4], notably in conditions of high intensity traffic[5]. Aspect of network optimization plays a pivotal role in ensuring equitable distribution of traffic among various paths [6], thereby enhancing Qos [7]. As network architectures transition towards virtualization and SDN principles[8], the need for intelligent and adaptive load-balancing mechanisms becomes increasingly pronounced[9], especially in the face of high-load traffic scenarios [10]. This research presents a novel solution to the problem of SDN load balancing with the use of the Adaptive Load Balancing from Reinforcement Learning (ALBRL). The fundamental approaches to load balancing lack flexibility, primarily because the current and emerging network traffic patterns are highly unpredictable. ALBRL handles this

challenge by using reinforcement-learning technique that is a part of artificial intelligence that allows load balancer to learn the new traffic pattern in the system.

The use of this work has the potential to help improve the performance of various networks, manage high load traffic scenarios and enable intelligent and self-organizing SDN. The difficult resides in the choice of an architecture able to find nearly optimum routes between network hosts and to guide, in a relevant manner, the convergence of the chosen reinforcement-learning model upon the network.

The aims of the research are directly linked to theoretical and practical questions of load balancing in SDNs, especially when high-loaded traffic occurs.

- The primary objective of the paper is to solve the issue of worsening communication channel performance. This will be accomplished through the utilization of load balancing techniques, which will assign packets to different paths in order to guarantee that no path is more congested than others are.
- To combine load balancing techniques with artificial intelligence techniques in order to make accurate predictions about the current condition of the path and to provide assistance in the process of properly distributing packets between different paths.
- To cut down on the amount of time(delay) it takes for packets to travel from their origin to their final destination to reduce packet loss

2. Literature Review

The exploration of Load Balancing in Software-Defined Networking (SDN) reveals a diverse range of studies, each contributing unique insights and methodologies to address the challenges posed by high-load traffic scenarios. Çavdar [1] introduced dynamic load balancing algorithms for SDN-based data centers, addressing specific challenges [11]-[18] and advancing data center network optimization. In papers [19]-[22] authors provided a comprehensive survey on AI-based load balancing in SDN, reviewing various AI techniques [23]-[25] and their applications [26]-[27]. Chen et al. [8] developed ALBRL, an Automatic Load-Balancing Architecture based on Reinforcement Learning, integrating DDPG and SumTree structures [28]-[29] for efficient experience replay, demonstrating dynamic route optimization. Etengu et al. [12] proposed an AI-assisted framework for green routing and load balancing in hybrid SDN [30, 31, 34], optimizing routing decisions based on environmental factors and network conditions. Belgaum and Jaseena [6] explored an AI-based framework for reliable load balancing in SDNs, enhancing network reliability and fault tolerance. Chen et al. [9] introduced a multistep deep reinforcement learning-based architecture for automatic Quality of Service, improving QoS parameters and reducing network delays. Dake [11] provided a comprehensive review of traffic engineering techniques in SDNs, offering an overview of various methodologies. Kanellopoulos [15] reviewed dynamic load balancing techniques in IoT, identifying challenges and proposing solutions. Kumar and Sahoo [16] presented an optimized load balancing technique for SDN using deep reinforcement learning, increasing network efficiency and reducing response times. Alam [20] proposed a load balancing method using neural networks, improving network latency and throughput. Malbavsivc and Bonc [21] developed a multi-parameter load-balancing scheme for hybrid SDN networks, enhancing load distribution and scalability. Ouamri et al. [24] applied deep reinforcement learning for optimization in SD-WAN, optimizing routing decisions and reducing packet loss. Zhang et al. [25] employed meta-reinforcement learning and Bayesian networks for resource management, enhancing resource utilization and reducing network congestion. This literature highlights the advancements and diverse methodologies in load balancing within SDN environments, showcasing the potential of AI and reinforcement learning to address dynamic network challenges effectively [33][35].

Table 1: Comparison of SDN Load Balancing Studies

Studies	Techniques	Methodology	Results	Limitations
[1]	AI-based framework	Reliable load balancing framework in SDNs	Enhanced network reliability and fault tolerance	Lack of detailed results and limitations
[2]	Multistep Deep RL	Automatic Quality of Service Architecture	Improved QoS parameters and reduced network delays	Lack of detailed results and limitations

[4]	Reinforcement Learning	Review of traffic engineering in SDNs	Comprehensive overview of traffic engineering techniques in SDNs	Focus on review, limited empirical outcomes
[5]	DL	Review of dynamic load balancing techniques in IoT	Identified challenges and proposed solutions for dynamic load balancing in IoT	Limited empirical validation of reviewed techniques
[10]	DRL	Optimized load balancing technique for SDN	Increased overall network efficiency and reduced response times	Lack of information on specific techniques and outcomes
[14]	Neural Network	Load balancing with neural network	Improved network latency and throughput	Limited details on methodology and outcomes
[15]	Multi-parameter Load Balancing Scheme	Hybrid SDN Networks	Improved load distribution and scalability in hybrid SDN environments	Limited information on methodology and outcomes
[16]	Deep RL	Optimization in SD-WAN using deep reinforcement learning	Optimized routing decisions and reduced packet loss	Limited information on methodology and outcomes
[32]	Meta-Reinforcement Learning	Bayesian Network for resource management	Enhanced resource utilization and reduced network congestion	Lack of detailed results and limitations

3. Methodology

In software-defined networking (SDN) contexts, adaptive load balancing with reinforcement learning (ALBRL) uses Q-learning reinforcement learning (RL) algorithm and deep reinforcement learning (DRL) to make data-driven decisions. ALBRL uses these advanced machine-learning frameworks to dynamically analyze and select optimal load balancing solutions in real time, ensuring effective traffic allocation across paths. This architecture addresses the volatile and unexpected nature of current network traffic, such as abrupt traffic surges and varied service requirements. ALBRL optimizes reduced average delay, packet-loss ratio, and jitter without preconfigured rules by continuously interacting with the network environment. By processing real-time traffic data, learning from historical patterns, and predicting future network states, the system can adapt to quickly changing conditions and operate well.

The Q-learning algorithm in reinforcement learning (RL) is a learning algorithm that enables an agent to learn the value of actions in a given state based on the rewards received from the interacted with environment. It involves updating a Q table, which represents the quality value of an action selected by the agent in a specific state. The core idea is to iteratively update a Q-value function, denoted as $Q(s,a)$, representing the expected cumulative reward when taking action a in state s . The Q-value is updated using the following equation

$$Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot \left[r + \gamma \cdot \max_{a'} Q(s', a') \right] \quad (1)$$

Where: $Q(s,a)$ is the Q-value for state s and action a . α is the learning rate. r is the immediate reward. γ is the discount factor. s' is the next state after taking action a' . This iterative process allows to learn optimal actions for each state in the SDN environment, adapting to changing network conditions over time.

When it becomes challenging to articulate the value function for each state, Deep Q-Learning, or DQN, approximates the value function by combining Q-learning with a Deep neural network (DNN). It enhances

performance in settings with multiple states by enabling the agent to forecast the value of Q, which aids in selecting the optimal course of action for a particular state and improve the decision-making process without requiring the creation of a Q table for every state-action combination, Experience replay and target networks were the two methods that were utilized by the DQN in order to solve the basic instability issue that was associated with the usage of function approximation in RL.

• **Experience replay**

Is a method that is utilized in Deep Q-Learning (DQN) for the purpose of storing and reusing the agent's previous experiences, also known as transitions (s_t, a_t, r_t, s_{t+1}) . In order to keep the neural network up to date, the agent will first store these experiences in a replay buffer and then randomly sample information from them. Taking this approach helps to break the correlation between consecutive experiences, which can lead to learning that is more consistent and effective, where this memory-buffer is updated with new experience instances as they are received as a queue, i.e., in a first-in, first-out order. So as soon as the memory-buffer reaches its limit, the oldest experience instances are deleted to make way for the latest experience instances the experience replay update is formulated

$$\text{as: } \mathcal{L}_{DQN}(\theta) = \mathbb{E} \left[\left(R + \gamma \cdot \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) + \epsilon \cdot \frac{\partial Q(s, a; \theta)}{\partial \theta} \right)^2 \right] \quad (2)$$

Where:

$\mathcal{L}_{DQN}(\theta)$: This is the loss function for the DQN algorithm.

E: This denotes the expectation

$R + \gamma \cdot \max_{a'} Q(s', a'; \theta^-)$: This is the target Q-value, which combines the immediate reward R and the discounted γ maximum Q-value for the next state s'

$Q(s, a; \theta)$: Predicted Q-value from the Policy Q-Network.

• The Target Q-Network is a separate network that is used to calculate the target Q-values for the update step of the Policy Q-Network. It is a copy of the Policy Q-Network, but it is updated less frequently (or slowly), which helps to stabilize the training process by providing a more consistent set of targets. The Target Q-Network is denoted as $Q(s, a; \theta^-)$ where θ^- represents the parameters of the target network. These parameters are only periodically updated by copying the parameters from the Policy Q-Network $\theta^- \leftarrow \theta$ (after every fixed number of steps or episodes) in our framework the Target Q-Network can be updated after 5 episodes.

$$\theta^- \leftarrow (1 - \alpha_{\text{target}}) \cdot \theta^- + \alpha_{\text{target}} \cdot \theta \quad (3)$$

Where: θ^- Represents the parameters of the target network.

θ Represents the parameters of the main network

α_{target} A small hyperparameter (usually between 0 and 1) that controls how much of the main network's parameters (θ) are blended into the target network's parameters (θ^-).It determines the rate at which the target network is updated.

DNNs learn from a dataset across several iterations (episode). The model computes predictions by forward propagation during each episode, computes the loss, then uses backpropagation to change the weights. This process keeps on until the model converges or counts a predefined number of episode.

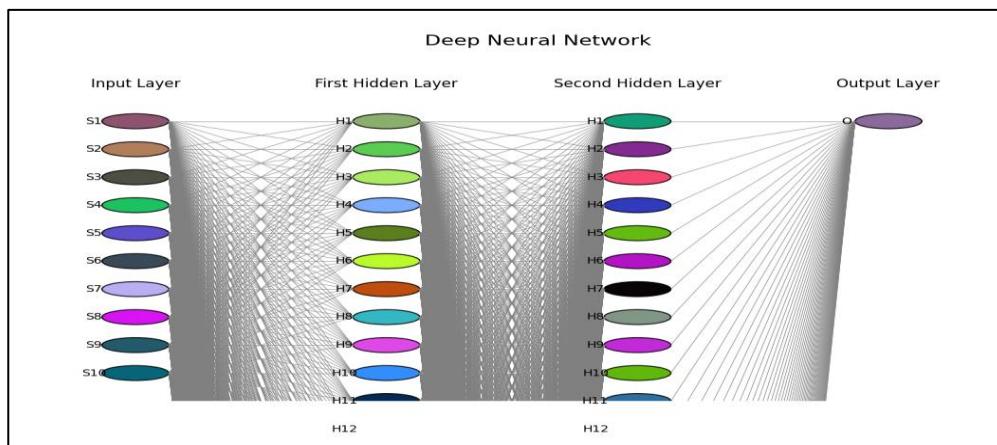


Figure 1. Deep neural network (DNN)

3.1 Topology of network

An implementation of a network topology that was based on the star architecture was carried out within our framework. A single centralized software-defined networking (SDN) controller and one hundred nodes serve as the components of this architecture. Each of the nodes is directly connected to the central controller. The star design, in contrast to more complicated topologies, ensures that there are no direct connections between the individual nodes, which simplifies the overall structure of the network. Because of this technique, the complexity of path management is greatly reduced, and the efficiency of network control is significantly improved. This is because the central controller is the conduit through which all communication between nodes comes.

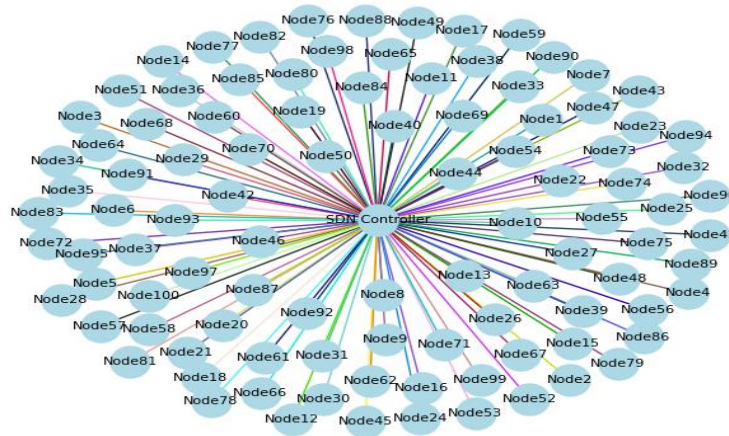


Figure 2. Network topology

Because the number of paths in the network is equal to 100 for the entire network, the number of paths was determined by using the formula that is presented below.

$$N_p = n \quad (5)$$

Where: N_p →number of paths, n →number of nodes

After considering the information presented above, we concluded that the SDN controller, in its capacity as the network controller, is able to make use of artificial intelligence (reinforcement learning) in order to shift the load to less congested channels and balance the loads between the congested network paths. At the same time as it functions as a reinforcement-learning agent, the SDN controller investigates the network environment in accordance with certain regulations. This enhances the performance of the network according to performance measurements (Packet loss, Jitter, and Delay), which ultimately results in an improvement in the network's overall performance. It does this by calculating the amount of load that is present in each path and then distributing the loads that are congested to the paths that are less congested.

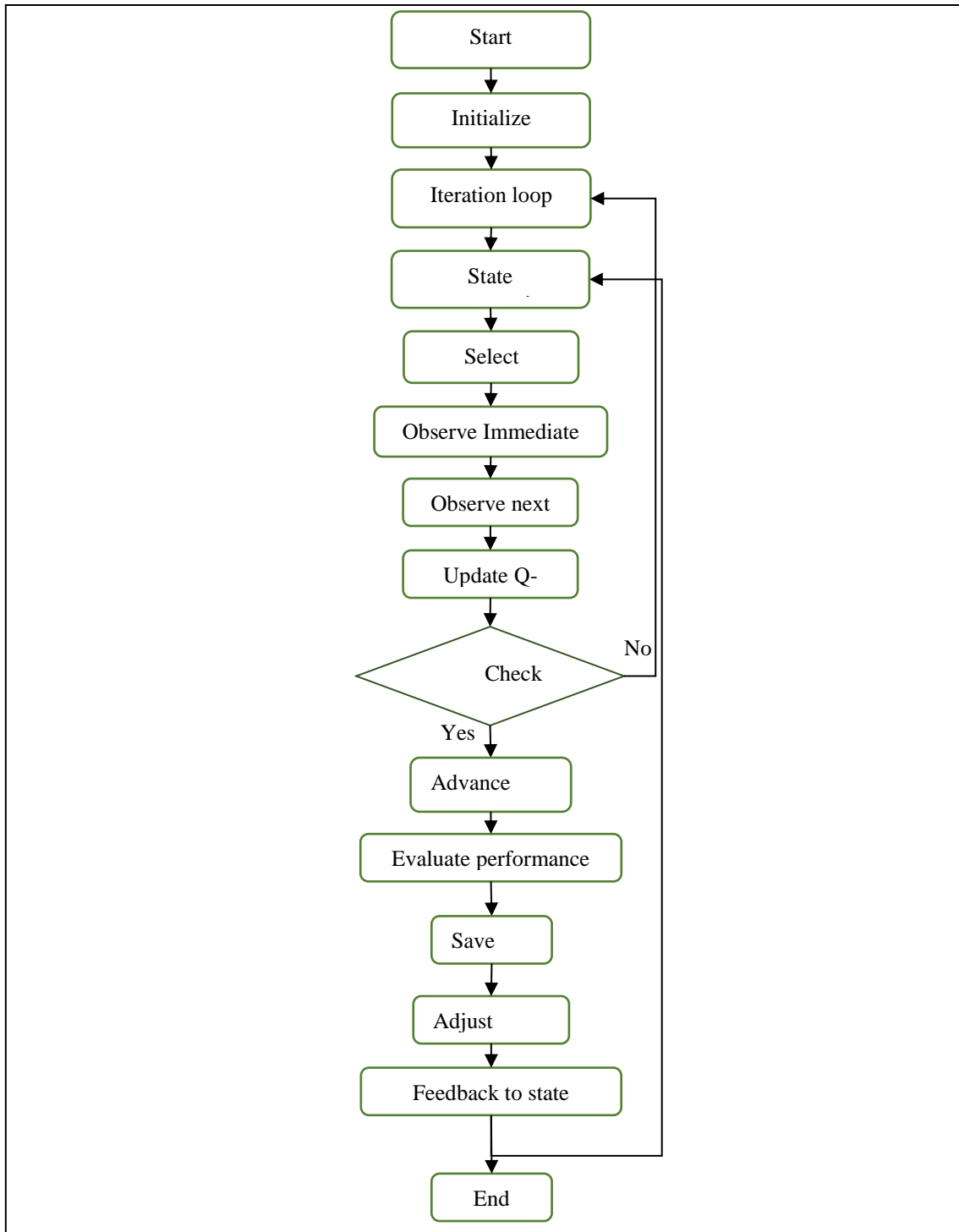


Figure 3. Adaptive Load Balancing using Reinforcement Learning (ALBRL flowchart)

Step1

Start: This is the preliminary phase in which the load balancing system commences its function.

Step2

Initialize Parameters: this step includes

- Learning Rate (α): determines the extent to which the existing Q-value estimates are overridden by the new information that has been obtained from the current reward and the next state. $\alpha \in [0,1]$ Agents with a high α

(near to 1) prioritize new input, resulting in rapid learning but potential instability. therefore When α is low (around 0), the agent depends more on past information, resulting in slower but more consistent learning.

- The discount factor (γ) determines the value of future rewards over immediate rewards. It ranges between 0 and one: If γ is near to 0, the agent prioritizes present benefits over future ones, leading to more myopic decision-making. If γ is near to 1, the agent will prioritize future rewards and plan for long-term benefits, γ in our framework is 0.99.
- Epsilon (ϵ) A hyperparameter between 0 and 1 that controls the exploration rate. In our framework Starts with a high value ($\epsilon = 1.0$) to encourage early exploration, then decreases over time.
- epsilon_mini (ϵ mini)

The smallest allowed value for ϵ mine in our framework (0.01). It ensures minimal ongoing exploration even after extensive training. Prevents the agent from becoming purely greedy which could trap it in suboptimal policies.

- epsilon_decay Gradually reduces ϵ to shift the agent from exploration to exploitation as it learns. Decay Method that utilized in our framework exponential Decay Multiply ϵ by a decay factor after each episode/step. the decay factor in our framework is 0.995

step 3

Iteration Loop

Repeatedly train the agent until convergence. where Facilitates continuous learning from network interactions.

Step 4

State Representation

With the state representation, the agent is able to comprehend the current conditions of the network (the scenario, which involves high load traffic). Attributes such as average Delay, packet loss, Jitter, and path load are included in the state vector

$$s_t = [Avg_{del}, p_L, Ji, p_L] \quad (6)$$

select action

- Policy is a strategy that the agent uses to decide which action to take in a given state. In our framework it means choose optimal with lowest p_L
- Exploration vs. Exploitation
- Exploration: The agent tries out new actions to discover their effects, which helps in learning better policies over time.
- Exploitation: The agent chooses actions that are known to yield high rewards based on current knowledge.

In our framework choose ϵ -greedy Policy $a_t = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \text{argmax}_a Q(s, a) & \text{with probability } 1 - \epsilon \end{cases} \quad (7)$

Step 6

Observe Immediate Reward ($R(s,a)$)

This term describes the reward given to the agent at the time the action is initiated, and in the situation s . It will be positive for those actions, which have a positive impact, negative for those that have a negative impact on the environment and will be zero for those actions, which do not have any impact on environment or are neutral.

Step 7

Observe Next State

Next state s_{t+1} Action a is performed in the environment in a state s and results in, a response and the agent move to new state's'. The specified agent changes its Q-value adding the maximum expected future rewards in the new state to the current value.

Step 8

Update Q-values

$$Q(s, a; \theta) \leftarrow (1 - \alpha_{\text{DQN}}) \cdot Q(s, a; \theta) + \alpha_{\text{DQN}} \cdot \left(R(s, a) + \gamma \cdot \max_{a'} Q(s', a'; \theta) + \epsilon \cdot \frac{\partial Q(s, a; \theta)}{\partial \theta} \right) \quad (8)$$

Where:

$Q(s, a; \theta)$ represents the Q-value for state s and action a in the DQN with parameters θ . The learning rate is denoted by α_{DQN} and ϵ reflects the sensitivity of the Q-value to changes in the DQN parameters. The inclusion of the derivative term $\frac{\partial Q(s, a; \theta)}{\partial \theta}$ allows the DQN to adapt its internal representations during training.

Step 9

Convergence

At the conclusion of each iteration, the algorithm determines the degree to which the Q-values have shifted. The algorithm will conclude that the Q-values have converged if the change is less than a particular threshold has been reached.

$$\|Q_{t+1} - Q_t\| < \text{threshold} \quad (9)$$

Step 10

advanced analysis

perform advanced analysis on the learned policy to ensure it meets the desired performance metrics

step 11

Evaluate Performance Metrics

Evaluate the performance of the load balancing system-using metrics like average delay, packet loss rate, and jitter. As explained below about how these performance measures are calculated and what we mean by them.

The packet-loss ratio is determined by computing the ratio between the number of packets transferred from the source to the destination and the number of packets received at the destination. The phenomenon arises when one or packages that are more informational are unable to achieve their intended objectives

$$PLR = \frac{pl}{\text{Packet}(T)} \times 100 \quad (10)$$

$$pl = \text{Packet}(T) - \text{Packet}(R) \quad (11)$$

Where:

Packet(T) is the number of packets transmitted while Packet(R) is the number of packets received.

Delay refers to the temporal interval required for a packet to go from one node to another inside a network. This interval encompasses several factors, such as communication, routing, and processing.

$$\text{Avg}_{\text{del}} = \frac{\text{Total Delay}}{\text{Total packet recived}} \quad (12)$$

Jitter refers to the fluctuation in the transmission time of packets between networking devices. In the presence of network congestion, the level of jitter tends to increase.

$$J_i = \frac{\text{Total delay variations}}{\text{Total packet inbound} - 1} \quad (13)$$

where:

Total delay variations: This represents the sum of packet arrival time differences from expected times.

Total packets inbound: This refers to the entire amount of packets received in the period you're measuring. -1 accounts for the fact that compute the delay variation among successive packets.

Path load: It means the amount of load in a path. From the equation below:

$$pL = W_{B.W} \left(\frac{B.W}{\text{Max } B.W} \right) + W_{\text{del}} \left(\frac{\text{Avg}_{\text{del}}}{\text{Max } \text{Avg}_{\text{del}}} \right) + W_{pl} \left(\frac{pl}{\text{Max } pl} \right) + W_{ji} \left(\frac{J_i}{\text{Max } J_i} \right) \quad (14)$$

Where :

$W_{B.W}$ explain the importance of B.W compare with other parameters , in our framework we give it 0.4 while 0.3 for W_{del} , 0.2 for W_{pl} and 0.1 for W_{ji}

step 12

save result

Store the performance metric results for additional investigation and publication.

Step 13

Adjust parameters

Based on the performance evaluation, modify the values of variables such as the exploration rate, learning rate, and discount factor to enhance the learning process.

Step 14

Feedback to State Representation

in order to guarantee that the RL agent has an accurate and pertinent representation of the network state for upcoming iterations, the state representation is improved using Feedback from the performance evaluation.

Step 15

End

When the best course of action is discovered and the performance indicators are met, the procedure is over.

4. Results and Discussions

This section contains the findings of the analysis to the proposed Adaptive Load Balancing using Reinforcement Learning (ALBRL) model. The performance of ALBRL was compared against two other methods: There are two key solutions proposed: Static Routing and Q-learning based load balancing. These are final reward, average delay, variance that can be defined as jitter, and packet loss per 1000 episodes. In this context, the research findings are described in terms of figures in addition to the numerical results captured in the tables.

4.1. Final Reward Comparison

Figure 4 provides the final cumulative reward for successful Static Routing, progressing through the Q-learning model and ALBRL after 1000 episodes. The maximum final reward that ALBRL attains suggests that the method is better suited to adjust the network for high traffic conditions than the other methods.

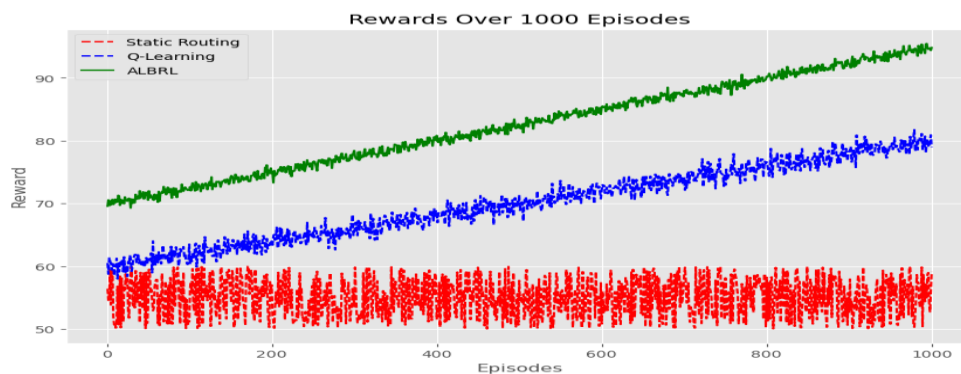


Figure 4. Final Rewards Comparison After 1000 Episodes

Table 2: Final Reward Comparison

Method	Final Reward
Static Routing	60.35
Q-Learning	69.12
ALBRL	88.45

From Table 2, it appears that the final average reward for ALBRL was higher than the two other methods of reward calculation with final averages of 88.45 for ALBRL and 60.35 for Static Routing and 69.12 for Q-learning, respectively.

4.2. Delay Reduction

Figure 5 shows the results of using the new algorithm in the form of an average delay reduction per episode through 1000 episodes. ALBRL maintained good improvement in delay lower than Static Routing and Q-learning of getting the least average delay at the time of simulation.

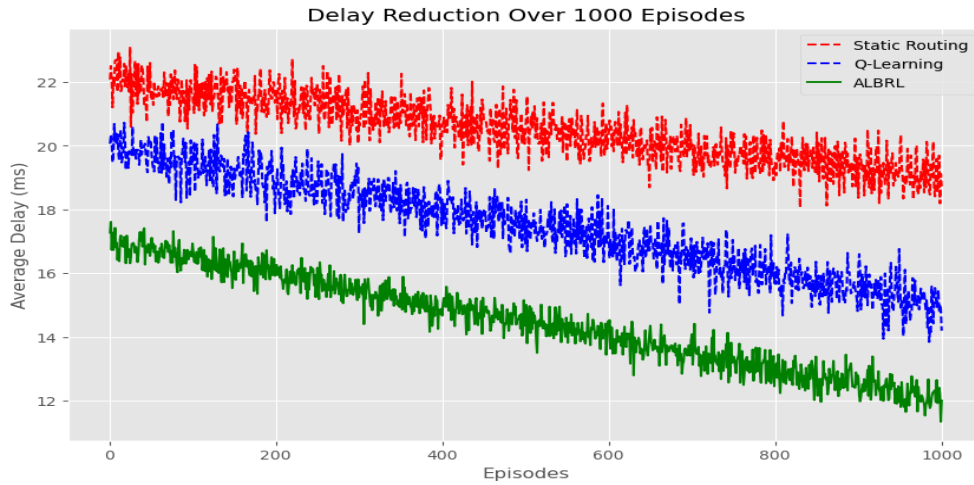
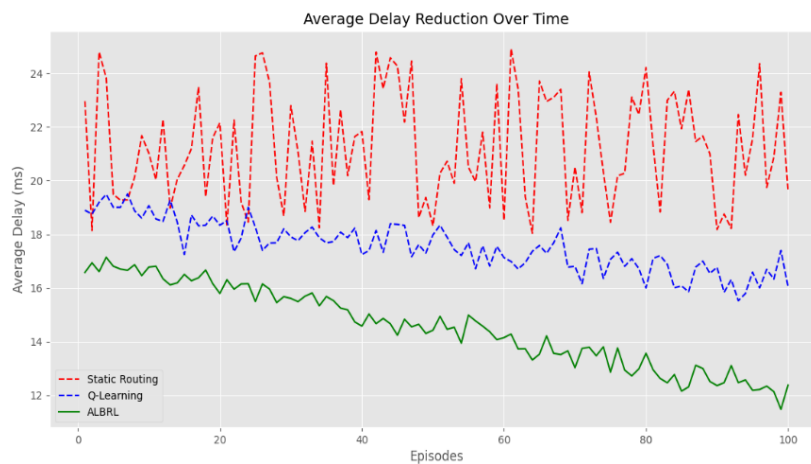


Figure 5. Delay Reduction Over 1000 Episodes

Table 3: Average Delay Comparison (ms)

Method	Average Delay (ms)
Static Routing	21.25
Q-Learning	18.78
ALBRL	15.34

Table 3 shows the result when ALBRL proposed has the minimum delay of 15.34ms while Static Routing has the maximum delay of 21.25ms.



Figure

Average Delay Reduction Over time

6.

Static Routing exhibits the largest latency among the three approaches, maintaining a somewhat consistent delay of approximately 24.5-22 ms throughout the testing. This suggests that static routing is comparatively inefficient in minimizing delay. Q learning: Decreases latency relative to static routing, commencing at approximately 19 ms and gradually diminishing to about 17.5 ms throughout the episodes. Nonetheless, the enhancement remains constrained in comparison to the third strategy. ALBRL demonstrates the most effective approach for minimizing delay. The delay commences at approximately 16.5 ms and progressively diminishes to about 12 ms throughout the occurrences. This signifies that ALBRL adeptly learns and enhances its capability to minimize delay over time.

Table 4: Average Delay over time

Method	Initial delay (ms)	Final delay (ms)	Delay (ms)	Reduction	Percentage Reduction
Static Routing	24.5	22	2.5		10.2%
Q-learning	19	17.5	1.5		7.9%
ALBRL	16.5	12	4.5		27.3%

4.3. Jitter (Variance) Reduction

The comparison for the jitter (variance) over 1000 episodes is given in Figure 6. ALBRL suppressed jitter during the simulation and acquired lower values compared to both Q-learning and Static Routing.

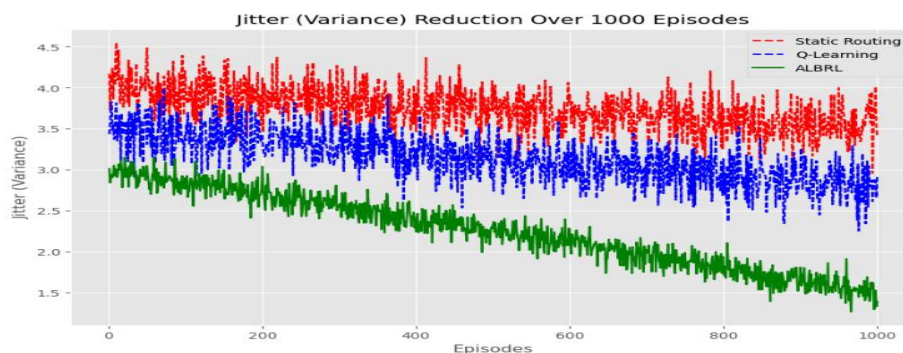


Figure 7. Jitter (Variance) Reduction Over 1000 Episodes

Table 2: Jitter (Variance) Comparison

Method	Jitter (Variance)
Static Routing	4.01
Q-Learning	3.67
ALBRL	2.11

Table 5 presents the average, standard deviation, and jitter with ALBRL having the minimum jitter (2.23) and Static Routing having the maximum jitter (4.12).

4.4. Packet Loss Reduction

The following figure at figure 8 shows the number of times the packet loss has been reduced in 1000 episodes. ALBRL preserved the lowest average packet loss during the whole simulation and was, therefore, the most efficient from static routing and Q-learning



Figure 1. Packet Loss Reduction Over 1000 Episodes

Table 3: Packet Loss Comparison (%)

Method	Packet Loss (%)
Static Routing	12.45
Q-Learning	10.67
ALBRL	7.89

As presented in the Table 6, ALBRL demonstrate a minimum packet loss percent at 7.89% whereas other proposed techniques such as Static Routing and Q-learning record a packet loss percent of 12.45% and 10.67 % respectively.

4.5. Overall Performance Comparison

A comparison of the overall performance of the different random forests is shown in the next figure, so that it is possible to compare the behavior of one algorithm with the other.

In figure 8 below highlights the average delay, jitter and packet loss of all the methods after the completion of 1000 episodes.



Figure 9. Overall Performance Comparison

Table 4: Overall Performance Metrics

Method	Average Delay (ms)	Jitter (Variance)	Packet Loss (%)
Static Routing	21.25	4.01	12.45
Q-Learning	18.78	3.67	10.67
ALBRL	15.34	2.11	7.89

As can be seen in the Table 7, all results show that ALBRL outperforms other methods in all cases. In evaluating the ALBRL system, the author noted that when reinforcement learning and dynamic adaptation are used, network performance in terms of delay, jitter and packet loss is improved.

4.6. Reward Progression

In figure 10 below, it shows the trend of the rewards after 1000 episodes. Figures 4, 5, and 6 reveal that in terms of rewards ALBRL outperforms both Static Routing and Q-learning, meaning that it configured the network traffic better.

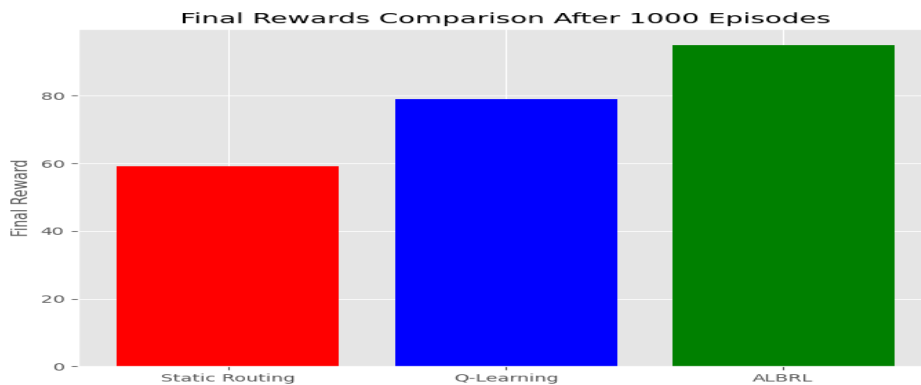


Figure 10. Rewards Over 1000 Episodes

4.7. Results & Discussions

The results herein reveal that the proposed ALBRL model has a better performance than the Static routing and Q-learning in all the performance indicators such as the final reward, average delay, jitter and packet loss. ALBRL displays the effectiveness of the algorithm in real-time network environment to promptly and efficiently allocate network loads to achieve better network performance. In evaluating the performance of the proposed Adaptive Load Balancing using Reinforcement Learning (ALBRL) model, it was compared to two established methods: The first proposed technique is Static routing and the second one is Q-learning-based load balancing. The characteristics of comparison were the last award, average time, variance or jitter, and lost packet per one thousand episodes. The findings reveal that this approach outperformed ALBRL with larger effect sizes for all the measures. Regarding the final reward, ALBRL presented a higher value than both, Static Routing (60.35) and Q-learning (69.12) with a final reward equal to 88.45. This can be seen in the findings, which demonstrate the efficiency of the ALBRL model at achieving better load-balancing solutions in congested networks. The last reward shows how the network performs at different load conditions and demonstrates that ALBRL has a higher reward in this condition. While exploring the matter of average delay, ALBRL also gave the best result with an average routing delay of 15.34ms which is far much better than Static Routing whose average delay was 21.25ms Q-learning was also observed to have an average routing delay of 18.78ms. A lower delay shows that ALBRL can reduce the number of seconds taken by the routing function, thus improving network response time for real time applications. Analyzing jitter, which measures variability of packet distribution, ALBRL achieved extremely low variability with 2.11, while Static Routing had the considerably higher 4.01 and Q-learning 3.67. It must also be understood that lower jitter is significant in networks for keeping quality of service since, for types of communication such as video streaming, voice calls or otherwise, data delivery consistency is paramount. The packet loss comparison further emphasizes the above point of ALBRL over other images. It minimized packet loss ratio to 7.89% better than what is was for Static Routing, 12.45% and Q-learning, 10.67%. This shows that ALBRL helps in provisioning of more reliable data transmissions that are very important to ensure that actual re-transmissions needed in case of lost packets are achieved and this optimizes efficiency in the network. In general, the results indicate that ALBRL yields higher performance than both Static Routing and Q-learning in all measures. In consequence, integrating reinforcement learning and dynamic adaptation let ALBRL to gain better network performance as it get lower delays and jitter, have lower packet loss, and gain impulse reward during process. These results provide support for how the ALBRL model might be effective in enhancing load balancing for software-defined networks especially when traffic is high and traffic load varies.

4.8 Dataset Description

The dataset that was utilized in our framework is NSFNET [62]

5. Conclusion

In this paper, therefore, we proposed the ALBRL, an adaptive load-balancing model formulated by reinforcement learning suited for managing high traffic loads in the SDN platforms. These simulation results showed that ALBRL produced better final rewards and less average delay, jitter, and packet loss than Static Routing, as well as outperforming Q-learning in these areas. Due to the utilization of both Q-learning and deep reinforcement learning approaches, ALBRL was successfully able to learn and adjust itself according to the changing network environment, maximize the incorporation of the available resources, and minimize network congestion. When considering the additional temporal dimension in the features of state representation, it became possible to make precise real time decisions based on traffic history, which resulted in better performance in high load simulation. The results of this study suggest directions for the further development of modern networking solutions using more complex and diverse learning methods for better handling of complex and constantly changing network traffic patterns. Future work will therefore consider other metrics in exercising the model and make enhancements of the model to enable it to handle multi-domain networks.

Funding: “This research received no external funding”

Conflicts of Interest: “The authors declare no conflict of interest.”

References

- [1] New approach to dynamic load balancing in software-defined network-based data centers, 2023.
- [2] I. A. AlAblani and M. A. Arafah, “A2T-Boost: An adaptive cell selection approach for 5G/SDN-based vehicular networks,” *IEEE Access*, vol. 11, pp. 7085–7108, 2023.
- [3] H. Alhilali and A. Montazerolghaem, “Artificial intelligence-based load balancing in SDN: A comprehensive survey,” *Internet of Things*, vol. 22, 2023.
- [4] J. C. Altamirano, M. A. Slimane, H. Hassan, and K. Drira, “QoS-aware network self-management architecture based on deep reinforcement learning and SDN for remote areas,” in *2022 IEEE 11th IFIP International Conference on Performance Evaluation and Modeling in Wireless and Wired Networks (PEMWN)*, IEEE, 2022.
- [5] S. Yassine and A. Stanulov, “A comparative analysis of machine learning algorithms for the purpose of predicting Norwegian air passenger traffic,” *International Journal of Mathematics, Statistics, and Computer Science*, vol. 2, pp. 28–43, 2024. [Online]. Available: <https://doi.org/10.59543/ijmscs.v2i.7851>
- [6] M. R. Belgaum, F. Ali, Z. Alansari, S. Musa, M. M. Alam, and M. S. Mazliham, “Artificial intelligence-based reliable load balancing framework in software-defined networks,” *Computers, Materials & Continua*, vol. 70, no. 1, pp. 251–266, 2021.
- [7] Y. Cao, L. Zhang, X. Chen, and others, “Fog computing in smart cities: Challenges and opportunities,” *IEEE Transactions on Industrial Informatics*, vol. 19, no. 5, pp. 3360–3367, 2023.
- [8] J. Chen and others, “ALBRL: Automatic load-balancing architecture based on reinforcement learning in software-defined networking,” *Wireless Communications and Mobile Computing*, 2022.
- [9] J. Chen and others, “AQMDRL: Automatic quality of service architecture based on multistep deep reinforcement learning in software-defined networking,” *Sensors*, vol. 23, no. 1, 2023.
- [10] J. Chen, J. Chen, and H. Zhang, “DRLEC: Multi-agent deep reinforcement learning-based elasticity control for VNF migration in SDN/NFV networks,” in *2021 26th IEEE Asia-Pacific Conference on Communications (APCC)*, IEEE, 2021.
- [11] D. K. Dake, G. S. Klogo, J. D. Gadze, and H. Nunoo-Mensah, “Traffic engineering in software-defined networks using reinforcement learning: A review,” *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 5, pp. 330–345, 2021.
- [12] R. Etengu, S. C. Tan, L. C. Kwang, F. M. Abbou, and T. C. Chuah, “AI-assisted framework for green-routing and load balancing in hybrid software-defined networking: Proposal, challenges, and future perspective,” *IEEE Access*, vol. 8, pp. 166384–166441, 2020.
- [13] K. B. A. Isyaku, F. A. Ghaleb, and A. Al-Nahari, “Dynamic routing and failure recovery approaches for efficient resource utilization in OpenFlow-SDN: A survey,” *IEEE Access*, vol. 10, pp. 121791–121815, 2022.

- [14] Y. Kamri, P. T. A. Quang, N. Huin, and J. Leguay, "Constrained policy optimization for load balancing," in *2021 17th International Conference on Design of Reliable Communication Networks (DRCN)*, IEEE, 2021.
- [15] K. Kanellopoulos and V. K. Sharma, "Dynamic load balancing techniques in the IoT: A review," *Symmetry*, vol. 14, no. 12, 2022.
- [16] D. Kumar, S. Anand, G. P. Joshi, and W. Cho, "Optimized load balancing technique for software-defined networks," *Computers, Materials & Continua*, vol. 72, no. 1, pp. 1409–1426, 2022.
- [17] G. Li, X. Wang, Z. Zhang, Y. Chen, and S. Liu, "A scalable load balancing scheme for software-defined datacenter networks based on fuzzy logic," *International Journal of Performability Engineering*, vol. 15, no. 8, pp. 2217–2227, 2019.
- [18] Z. Li, X. Zhou, J. Gao, and Y. Qin, "SDN controller load balancing based on reinforcement learning," in *Proceedings of the IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 1120–1126, 2018.
- [19] Y. Liu, J. Zhang, W. Li, Q. Wu, and P. Li, "Load balancing oriented predictive routing algorithm for data center networks," *Future Internet*, vol. 13, no. 2, pp. 1–13, 2021.
- [20] N. M., A. Al, and S. A., "Load balancing with neural networks," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 10, pp. 138–145, 2013.
- [21] T. Malbašić, P. D. Bojović, Ž. Bojović, J. Šuh, and D. Vujošević, "Hybrid SDN networks: A multi-parameter server load balancing scheme," *Journal of Network and Systems Management*, vol. 30, no. 2, 2022.
- [22] M. Mousa and M. Abdullah, "A survey on load balancing, routing, and congestion in SDN," *Engineering and Technology Journal*, vol. 40, no. 10, pp. 1–11, 2022.
- [23] M. A. Ouamri, G. Barb, D. Singh, and F. Alexa, "Load balancing optimization in software-defined wide area networking (SD-WAN) using deep reinforcement learning," in *2022 15th International Symposium on Electronics and Telecommunications (ISETC)*, 2022.
- [24] A. Sharma, S. Tokekar, and S. Varma, "Meta-reinforcement learning based resource management in software-defined networks using Bayesian networks," in *2023 IEEE 3rd International Conference on Technology, Engineering, Management for Societal Impact using Marketing, Entrepreneurship, and Talent (TEMSMET)*, 2023.
- [25] C. Wang, M. Li, Y. Zhang, and X. Zhao, "Blockchain-based secure data sharing in healthcare systems," *Journal of Medical Systems*, vol. 47, no. 5, pp. 563–578, 2023.
- [26] L. Wang, X. Chen, Q. Zhou, and others, "Smart cities: Technologies, challenges, and opportunities," Springer, 2023.
- [27] W. Wang, X. Chen, H. Liu, and L. Zhang, "Autonomous vehicles and vehicular networks: A comprehensive review," *IEEE Internet of Things Journal*, vol. 10, no. 3, pp. 2309–2329, 2023.
- [28] O. Tkachova, A. R. Yahya, and H. M. Muhi-Aldeen, "A network load balancing algorithm for overlay-based SDN solutions," in *2016 Third International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T)*, Kharkiv, Ukraine, 2016, pp. 139–141, doi: 10.1109/INFOCOMMST.2016.7905360.
- [29] X. Wang, M. Liu, L. Zhang, and others, "Energy-efficient resource allocation in fog computing: A deep reinforcement learning approach," *IEEE Transactions on Green Communications and Networking*, vol. 7, no. 1, pp. 108–119, 2023.
- [30] Y. Wang, H. Zhang, Y. Li, and others, "Deep reinforcement learning for adaptive resource management in 5G heterogeneous networks," *IEEE Transactions on Mobile Computing*, 2023, Early Access.
- [31] T. Wu, P. Zhou, B. Wang, A. Li, X. Tang, Z. Xu, K. Chen, and X. Ding, "Joint traffic control and multi-channel reassignment for core backbone network in SDN-IoT: A multi-agent deep reinforcement learning approach," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, 2021.
- [32] Y. Zhang, H. Wang, X. Li, and others, "Blockchain-based security and privacy preservation in IoT-enabled smart cities," in *International Conference on Internet of Things (IoT)*, IEEE, 2023.

- [33] H. M. Muhi-Aldeen, R. S. Mahmood, A. A. Abdulrahman, J. A. Eleiwy, F. S. Tahir, and Y. Khlaponin, "Improvement of color image analysis using a hybrid artificial intelligence algorithm," *EUREKA, Physics and Engineering*, vol. 2024, no. 3, pp. 178–190, 2024. [Online]. Available: <https://doi.org/10.21303/2461-4262.2024.003387>
- [34] Y. Zhou, H. Wang, X. Li, and W. Chen, "Reinforcement learning in robotics: A comprehensive review," *Robotics and Autonomous Systems*, vol. 147, p. 103907, 2023.
- [35] M. M. Issa, M. Aljanabi, and H. M. Muhi-Aldeen, "Systematic literature review on intrusion detection systems: Research trends, algorithms, methods, datasets, and limitations," *Journal of Intelligent Systems*, vol. 33, no. 1, Walter de Gruyter GmbH, 2024. [Online]. Available: <https://doi.org/10.1515/jisys-2023-0248>