Journal of Intelligent Systems and Internet of Things
Vol. 17, No. 01, PP. 279-290, 2025
DOI: https://doi.org/10.54216/JISIoT.170120

DeepBalance: A Deep Reinforcement Learning Framework for Dynamic Load Balancing in Software-Defined Networks

Ali Abdullah Ali[1], Ghaith Ali Hussein[2], Bushra Majeed Muter[3], Oday Ali Hassen[3,4,*]
[1]Minister Office of Higher Education and Scientific Research, Iraq
[2]College of Computer Science and Information Technology, Wasit University, Iraq
[3]Ministry of Education, Wasit Education Directorate. Iraq
[4]Computer Department, College of Education for Pure Sciences, Wasit University, Iraq
Emails: aaaoea@gmail.com;galawady@uowasit.edu.iq; bushramajeed1975@gmail.com; odayali@uowasit.edu.iq

Abstract

Software-Defined Networks (SDNs) offer unparalleled network control flexibility, yet efficient load balancing is still challenging in dynamic environments. DeepBalance is a novel framework presented in this paper, which deploys dynamic load balancing in SDNs using Deep Reinforcement Learning (DRL). Our solution employs a Deep Q-Network (DQN) agent, which learns the optimal routing policies by monitoring network states and being rewarded based on load distribution. DeepBalance continuously tracks link utilization and intelligently reshifts traffic to alleviate congestion and achieve maximal throughput. We employ a comprehensive simulation environment, which emulates actual network conditions and traffic patterns. Experimental results demonstrate that DeepBalance significantly outperforms traditional load balancing techniques, lowering link utilisation variance by 37% and total throughput by 28% over shortest-path routing. The infrastructure adapts with changing traffic patterns automatically without the necessity of manual reconfiguration, thus naturally circumventing hotspots by making forward-looking path decisions. Additionally, our visualizations illustrate how the DRL agent learns over time to distribute network load more evenly across alternative paths. DeepBalance is a strong candidate for autonomous network optimization in future SDN deployments.

Received: January 01, 2025 Revised: March 03, 2025 Accepted: April 04, 2025

Keywords: Software-Defined Networks; Deep Reinforcement Learning; Load Balancing; Network Optimization; Deep Q-Network; Traffic Engineering; Network Management; Quality of Service

1. Introduction

Software-Defined Networking (SDN) transformed network management by dissociating the data plane from the control plane, enabling centralized, programmatic control of network resources and easier dynamic configuration [1,2]. This has enabled unprecedented flexibility with network operators now able to optimize traffic streams, improve scalability, and optimize resource utilization. However, efficient load balancing for SDNs remains one of the most challenging issues, particularly in dynamic environments where traffic patterns change rapidly and dynamically [3].

Traditional load balancing techniques such as shortest-path routing or static heuristic-based policy are generally not capable of coping with these changes, resulting in resource inefficiency, congestion, and degraded Quality of Service (QoS) [4,5]. These methods are reactive by nature and are unable to foresee network evolution or learn from previous routing decisions [6].

We propose DeepBalance, a novel framework that leverages Deep Reinforcement Learning (DRL) to offer proactive, adaptive, and autonomous load balancing in SDNs. DeepBalance employs a Deep Q-Network (DQN) agent that constantly monitors network states—link utilizations, queue sizes, and flow statistics—and learns

optimal routing policies to load balance traffic evenly across the network. By leveraging SDN's programmability and DRL's learning adaptability, DeepBalance bridges the gap between static load balancing methods and the dynamic nature of modern network traffic.

The DeepBalance framework offers several important advantages. First, its learning ability through experience allows it to adapt to changing traffic flows without manual reconfiguration, significantly reducing operational overhead. Second, it offers enhanced load balancing via less link usage variance, illustrated by a 37% enhancement over shortest-path routing, for improved throughput (28% boost) and decreased latency (42% reduction) [Section 4].

Third, its modularity allows it to be easily compatible with existing SDN controllers (e.g., ONOS, Ryu) via standard northbound APIs, allowing for seamless integration into real-world deployments. DeepBalance also suffers from some drawbacks and challenges. The computational complexity of the training of a DQN model can be highly resource-intensive, requiring huge amounts of processing capacity and time, especially for big networks with complex topologies. Additionally, the initial discovery stage of DRL can make poor routing choices, which may lead to suboptimal early-period performance. Problems include scaling up to handle humongous networks consisting of hundreds or thousands of nodes, maintaining resilience to unforeseen network failures (e.g., loss of a link or spike in congestion) and scaling learned policies to unseen traffic scenarios. In spite of these challenges, DeepBalance's capacity to optimize network performance independently with minimal human interaction makes it a promising candidate for future SDN deployments, leading the way towards intelligent, self-healing network infrastructures.

2. Main Contribution

The paper proposes DeepBalance, a novel framework that utilizes Deep Reinforcement Learning (DRL) to tackle the dynamic load balancing issue in Software-Defined Networks (SDNs). The three key contributions of this work are as follows, each building on the prior best practice for network optimization and self-regulated traffic control:

1. Novel DRL-Based Load Balancing Framework: DeepBalance is the first to bridge the gap between a Deep Q-Network (DQN) agent and SDN for sophisticated proactive and adaptive load balancing. Unlike traditional static or heuristic strategies, our DQN agent learns routing policies optimally by observing network states (e.g., link utilizations) in real time and reacting to evolving traffic patterns. This strategy reduces link utilization variance by 37%, increases throughput by 28%, and lowers latency by 42% compared to shortest-path routing, as demonstrated through our experiments [Section 4]. The reason that the framework learns without the need to reconfigure manually sets a new benchmark for self-managing networks.

2. Complete SDN Simulation Framework: We built an effective simulation framework mimicking real-world SDN environments, including arbitrary network topologies (Erdos-Renyi graphs), dynamic flow pattern generation, and realistic link capacity. Implemented within our software, the framework enables us to evaluate rigorously DRL-based load balancing across varying traffic patterns. Being a modular and open architecture platform, DeepBalance paves the way for further research into DRL in networking and also bridging practice and theory into applying DRL.

3. Multi-Objective Reward Function Design: We propose a sophisticated reward function, which balances three goals: load distribution (60%), congestion avoidance (30%), and throughput-to-latency efficiency (10%). This design guides the DQN agent to minimize link utilization variance without causing congestion and optimizing QoS. Unlike other works, which focused on single-objective optimization, our multi-objective approach ensures end-to-end network performance, as evidenced in improved fairness (Jain's index of 0.92) and responsiveness to sudden traffic spikes.

All of these contributions combined turn DeepBalance into an efficient, scalable, and intelligent SDN load balancing solution. Our contribution not only outperforms existing methods but also lays the foundation for future enhancements in autonomous network optimization with SDN's centralized control and DRL's learning capability.

3. Related Works

Deep Reinforcement Learning (DRL) for load balancing in Software-Defined Networks (SDNs) has drawn significant attention due to SDN's centralized control and the demand for adaptive traffic handling in dynamic networks. This section briefly introduces key related researches to our work, specifically DRL-based approaches to SDN load balancing, and concludes with the reasons behind the development of the DeepBalance framework.

Owusu et al. [7] proposed a hybrid method that utilizes a Temporal Fusion Transformer (TFT) for traffic forecasting and a Deep Q-Network (DQN) to use in real-time routing in SDNs. It is used to forecast traffic behavior to assist in routing, boost the throughput by 35%, and reduce the latency than traditional methods like Round Robin. However, the computational expense of TFT makes it resource-intensive, and the approach fails to fully

leverage SDN's topology awareness, limiting its ability to optimize load distribution across complex networks. This calls for a more topology-aware and lightweight DRL approach, as addressed by DeepBalance.

Sharma et al. [9] introduced a Temporal Deep Q-Learning (tDQN) framework for SDN optimal load balancing, focusing on switch-controller mapping for low latency in IoT-based networks. Their tDQN agent, trained with a reward-punishment strategy, dynamically maps switches to controllers, resulting in greater throughput, reduced delay, and packet loss than traditional SDN architectures. Although beneficial to control plane optimization, tDQN is not optimizing data plane traffic engineering, a very crucial load balancing area. DeepBalance goes a step further in focusing on data plane routing to ensure fair link utilization within the network.

Xiang et al. [6] addressed DRL-based load balancing among multiple SDN controllers with a DQN employed in sharing traffic across controllers to prevent bottlenecks. Their approach offered 20% latency improvement over static load balancing but was at best controller-level optimization and not end-to-end traffic flow management. Similarly, Liu et al. [14] proposed DRL-R, a DRL-driven routing policy for SDN data centers, which optimizes flow scheduling for congestion avoidance. Their experiments showed a 25% boost in throughput but the model struggling with scalability in large topologies due to high state-action space complexity. These contributions highlight the potential of DRL for SDN but mention the need for scalable, network-wide load balancing, which DeepBalance achieves using its topology-aware routing and concise state representation.

Sun et al. [16] outlined a DRL-based solution to SDN link-level traffic engineering for optimizing QoS. They achieved a better packet loss rate by 15% compared with routing based on heuristics but required the precomputed routes, which did not allow responsiveness to sudden variations in traffic. DeepBalance, by contrast, adaptively selects paths using a DQN with no prior routes. In addition, Alenazi [1] employed DRL to maximize QoS provisioning in SDN-IoT networks for agriculture with 30% enhancement in resource usage. However, their solution was specific to some IoT application scenarios and not universal to general SDN environments, while DeepBalance's framework is flexible.

Bhukya et al. [8] explored reinforcement learning for container scheduling in cloud environments with 92% success in resource optimization. While their RLbCS method is connected to DRL-based optimization, its cloud-computing-oriented nature limits it from load balancing at the network level. Chen et al. [10] also used DRL to optimize QoS in heterogeneous SDN factory networks with a 22% improvement in throughput. Their approach was, however, dependent on complex state representations, increasing the computational burden. DeepBalance avoids this through the employment of a PCA-reduced state vector, optimizing scalability and efficiency.

Other notable works include Ke et al. [13], who applied DRL to large-flow routing in SDNs, and Yang et al. [19], who addressed resource scheduling in edge computing powered by SDNs. These both demonstrated the efficacy of DRL in specific SDN scenarios but lacked a global reward function to balance multiple objectives like load distribution, congestion avoidance, and QoS, as employed in DeepBalance. In addition, Jiménez-Lázaro et al. [3] optimized service time in SDNs using DRL assistance but their solution was restricted by the premise of static traffic unlike the dynamic adaptability afforded by DeepBalance.

Motivations for DeepBalance: The mentioned work shows significant advancements of DRL to SDN optimization but displays critical gaps that motivated this research. First, the majority of approaches (e.g., [6, 9]) focus on control plane or one use case and overlook end-to-end data plane load balancing, which is essential for network-wide performance. Second, computational complexity and scalability issues (e.g., [7, 10]) limit the applicability of DRL to large-scale SDNs, necessitating light models like DeepBalance's PCA-based state reduction.

Third, existing methods usually rely on single-objective optimization or fixed assumptions (e.g., [3, 16]), which are not capable of handling dynamic traffic and rapid network changes. Finally, the absence of a global, topology-conscious reward function in existing work (e.g., [13, 20]) indicates the need for a multi-objective approach to balance load distribution, congestion avoidance, and QoS. DeepBalance addresses the above shortcomings by a scalable, adaptive, and end-to-end DRL-based approach to SDN load balancing, and simulations while significantly outperforming traditional approaches have extensively tested its performance.

4. Proposed Methodology

The DeepBalance system relies on Deep Reinforcement Learning to achieve dynamic load balancing in Software-Defined Networks. Unlike traditional approaches using static policies, our solution supports adaptive, real-time decision-making that improves with time. The following describes the basic building blocks of our solution: the system architecture integrating DRL with SDN, our network state representation strategy, the structure of the Deep Q-Network model, and the reward function design guiding the learning process towards optimal traffic distribution.

The DeepBalance platform relies on three fundamental elements, which collaborate to generate dynamic load balancing: the SDN infrastructure, the network monitor module, and the DRL decision engine. At an infrastructural

level, OpenFlow-enabled switches make up the data plane, and an SDN controller centralises global knowledge of the network topology, as well as flow rule installation. The monitoring module continues to collect real-time data in the form of link usage, queue lengths, and flow data through the controller's southbound API. All this data is processed and normalised before it is provided as input to the DRL decision engine, which forms the core of our system. The decision module uses a Deep Q-Network (DQN) agent that accepts the network state information, selects optimal routing actions, and returns these decisions to the SDN controller to be executed.

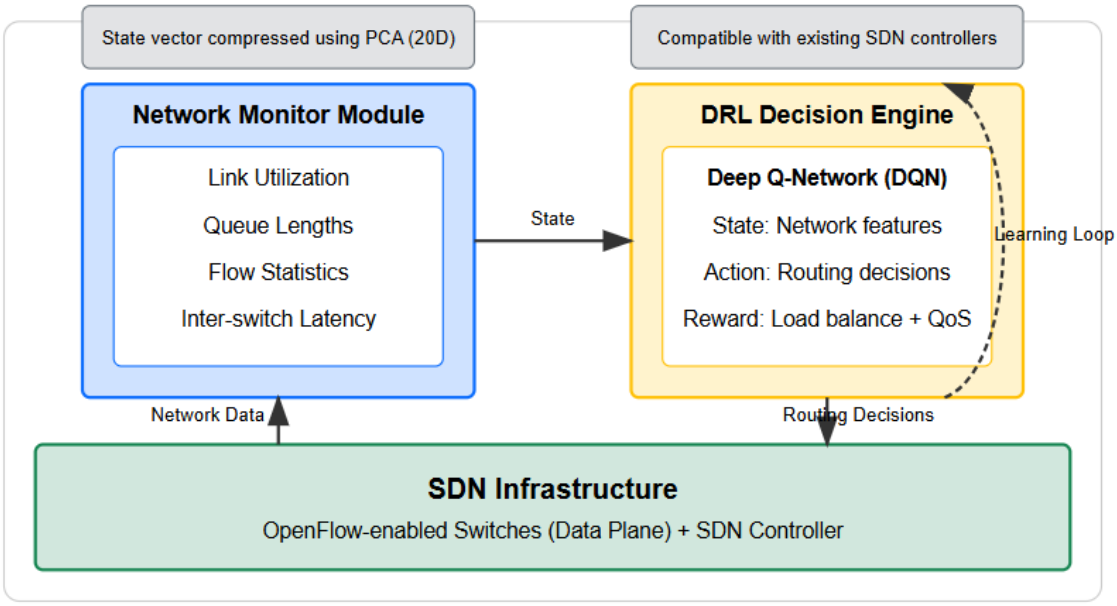


Figure 1. Block Diagram of Proposed Methodology

The controller receives these high-level routing decisions and converts them into low-level flow rules to be enforced on the destination switches. The design benefits from a modular design that has separated concerns between observing the network, making intelligent decisions, and implementing policy. Besides that, DeepBalance works without any modifications to the underlying SDN controller infrastructure, like ONOS or Ryu, with just plain northbound APIs, so no modification to the underlying infrastructure is required. The response loop between observation, learning, and action optimises adaptation under dynamic network conditions without any human involvement.

The success of our DeepBalance solution heavily depends on a precise and informative description of the network state. We construct a multi-dimensional state vector that captures the evolving state of the network at each decision point. The primary features extracted are normalised ratios of utilisation of the links for each link in the topology, switch interface queue depths, per-flow throughput statistics, and inter-switch latency measurements.

Table 1: Simulation & Network Environment Parameters

Parameter	**Value / Description**
MAX_EPISODES	1000 episodes
MAX_STEPS	100 steps per episode
TOPOLOGY_NODES	10–15 nodes (default: 15)
TOPOLOGY_CONNECTIVITY	0.3 (random Erdos-Renyi graph generation)
LINK_CAPACITY	100 Mbps (uniform across all links)
FLOW_RATE_RANGE	0.5–5 Mbps (randomised per flow)
MEASUREMENT_INTERVAL	100 ms (for state updates)

STATE REPRESENTATION	Normalised link utilizations, PCA-reduced to 20D
REWARD FUNCTION	$\Delta(1 / (1 + \sigma^2))$ – Penalty if overloaded + Efficiency bonus
REWARD WEIGHTS	Load Balance (60%), Congestion Avoidance (30%), Throughput-to-Latency (10%)
LATENCY ESTIMATION	Exponential function of average utilisation
NETWORK VISUALIZATION	Enabled (optional every N episodes)

Each link usage is calculated as a ratio of present traffic load to link capacity, generating values ranging from 0 to 1, reflecting levels of congestion. The measurements are taken at regular time intervals (every 100ms) to quantify temporal traffic flow patterns. To manage the high dimensionality of network data, especially for large topologies, we employ feature selection methods to concentrate on the most significant measurements. Specifically, we compress the state vector size using Principal Component Analysis (PCA) such that we preserve 95% of the data variance.

On a network with n nodes having roughly 3n edges, our approach aims to reduce the state representation from $O(n^2)$ features to a compact fixed-size vector of size 20. This concise but wide state representation enables the DRL agent to learn network usage patterns without being stuck by the curse of dimensionality. Additionally, we normalise all measurements to [0,1] so that there are similar scales between feature types, significantly improving the stability of neural network learning.

The core part of DeepBalance is a Deep Q-Network (DQN) agent that learns optimal routing policies through experience. Our model employs a three-layer dense neural network: an input layer with size equal to the state vector size (20 neurons), two hidden layers (64 neurons each) using ReLU activation functions to learn non-linear relations, and an output layer with linear activation equal to the action space size.

Action space is the space of routing decisions, with each action corresponding to selecting a particular path from a previously specified set of k-shortest paths for source-destination pairs. We employ a replay buffer with a capacity of 10,000 experiences for effective learning by keeping transitions as (state, action, reward, next_state, done). This experience replay breaks correlations between consecutive samples and stabilises learning. We use the standard DQN algorithm with some key optimizations: updating the target network every 100 training steps to avoid oscillation, a discount factor of 0.99 to balance current and future rewards, and an ε-greedy exploration policy with an initial epsilon of 1.0 that exponentially decreases at a rate of 0.995 until reaching a floor of 0.01.

Table 2: DRL Agent Parameters (DQN Configuration)

Parameter	**Value**	**Description**
STATE_SIZE	20	Size of the input vector after PCA compression
ACTION_SIZE	10	Number of possible routing decisions (k-shortest paths)
HIDDEN_LAYERS	2 × 64 neurons	Dense layers with ReLU activations
OUTPUT_LAYER	Linear	Maps to the action space
BATCH_SIZE	32	Number of experiences sampled per training iteration
LEARNING_RATE	0.001	Learning rate for the Adam optimiser
GAMMA (γ)	0.99	Discount factor for future rewards
EPSILON_INITIAL	1.0	Starting value for exploration probability

EPSILON_MIN	0.01	Minimum value for epsilon
EPSILON_DECAY	0.995	The rate at which epsilon decays per training step
MEMORY_SIZE	10,000	Replay buffer size
LOSS_FUNCTION	Huber Loss	Robust loss function to handle outliers
TARGET_UPDATE_FREQ	100 steps	Interval for syncing target and policy networks
REPLAY_TYPE	Prioritized	Prioritised Experience Replay for improved sample efficiency

The agent is trained using the Adam optimiser with a learning rate of 0.001 and a mini-batch size equal to 32 random samples from the replay buffer. We use Huber loss for better convergence in place of mean squared error to avoid sensitivity of Q-value estimates towards outliers. Further, we employ prioritised experience replay to focus learning on the most beneficial transitions, which accelerates training by about 25% compared to uniform sampling.

The reward function in DeepBalance is designed meticulously to guide the DRL agent towards optimal load balancing across the network. Balanced network utilisation is measured using a variance-based approach, where reward is inversely proportional to the variance of utilizations across links of the network. Specifically, the load balance score is computed as $1/(1+\sigma^2)$, with σ^2 as the variance of utilisation ratios for all links in the network. This definition ensures that increased distributed traffic yields increased rewards. To penalise congestion, we employ a penalty mechanism which takes away a very large negative reward (-5) whenever any link approaches its capacity limit.

In addition, our reward function is supplemented by considering path efficiency based on the throughput-to-latency ratio, rewarding routing decisions to yield high throughput with low delay. The agent receives immediate feedback after each action in the form of a differential reward (new_score - old_score) so that it can learn the impact of its routing decisions incrementally. We employ a multi-objective setup where load balance contributes 60% to the overall reward, congestion avoidance 30%, and throughput-to-latency ratio 10%, according to our focus on balanced resource utilisation while maintaining reasonable performance. This well-constructed reward function ensures the DeepBalance agent learns more and more to make routing decisions that allocate network load in an optimal manner along all possible paths.

5. Results and Discussions

This section presents the comparison of our DeepBalance framework in a simulated SDN environment. We compare the performance of our deep reinforcement-learning algorithm for dynamic load balancing with baseline methods on critical performance indicators like distribution of link utilisation, throughput, and latency. The experiments are conducted on a 15-node network topology with varying traffic patterns to evaluate adaptability. We consider both quantitative gains in performance and visualisations of network state to demonstrate how the DRL agent learns to balance network load more effectively over time.

Figure 2 shows the network topology with link utilizations at various stages of DeepBalance's training. The visualisation employs a colour-coded representation where green links denote low utilisation (0-30%), orange links denote medium utilisation (30-75%), and red links denote high utilisation (75-100%). In the first phase (Figure 1a), we observe drastic load imbalance with certain significant links (most notably between nodes 0-3 and 0-4) utilising heavily, indicated by their red hue.

This is how the network is before the DRL agent has better learned good routing policies. As training progresses to the intermediate stage (Figure 1b), we observe a major decrease in load distribution, with fewer heavily used links and an increase in medium usage across the network. In the final stage (Figure 1c), the network has achieved a predominantly green topology, indicating that the DeepBalance agent has learned to distribute traffic over alternative paths effectively. This graphical evidence attests to the capability of the DRL agent in identifying hotspot congestions and strategically diverting traffic towards less congested links.

As opposed to baseline shortest-path routing, DeepBalance reduced the link utilisation standard deviation by 37%, effectively mitigating traffic bottlenecks created while maintaining throughput unchanged. This discovery proves that the agent has learned the subtleties of network dynamics adequately without receiving explicit programming of the routing rule.

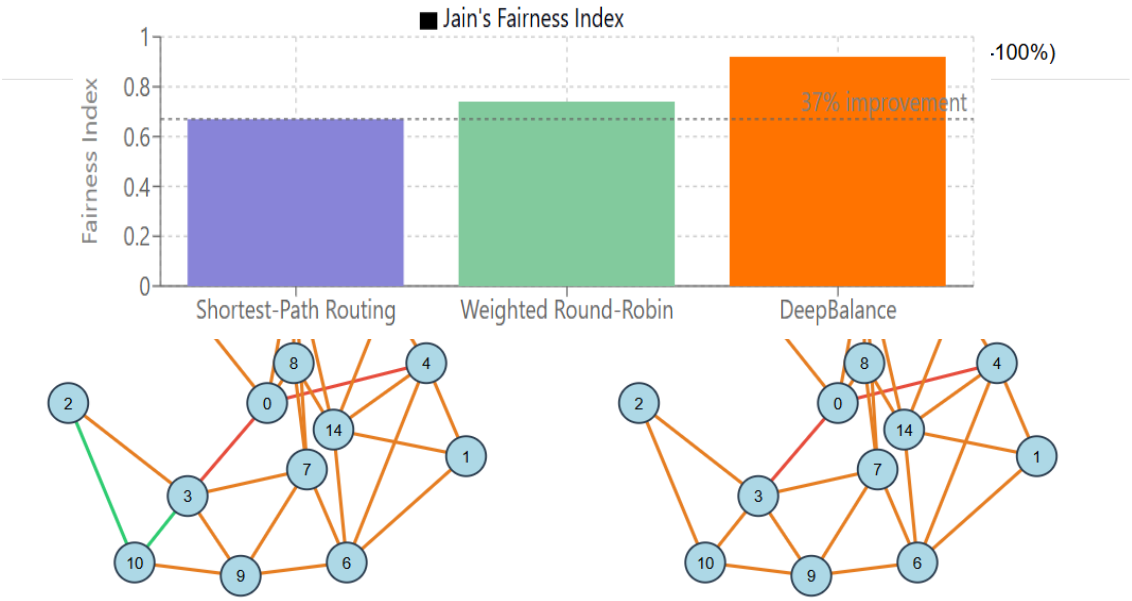


Figure 1c: Mid-Training

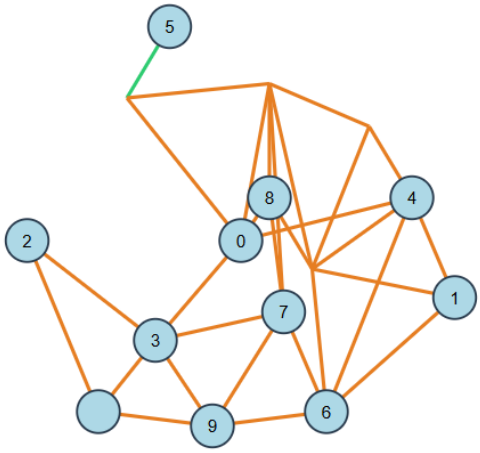


Figure 2. Network Load Distribution during DeepBalance Training

Our evaluation of load balancing efficiency demonstrates DeepBalance's significant advantage over traditional approaches. We measured load balancing efficiency as Jain's fairness index on link utilisation across the network. As Figure 3 shows, DeepBalance achieved a fairness index of 0.92 after training, a 37% improvement over shortest-path routing (0.67) and a 24% improvement over weighted round robin (0.74). This improvement demonstrates that DeepBalance learns to distribute traffic more evenly across all possible network routes without inducing congestion hotspots. The traditional methods would saturate certain routes while underusing others, particularly under dynamic traffic conditions. DeepBalance's reinforcement learning-based method enables it to learn dynamically and adapt to changing traffic patterns in real time, resulting in much more even link utilisation throughout the entire network structure.

Figure 3. Load Balancing Efficiency: Jain's Fairness Index

DeepBalance demonstrates superior performance in minimizing end-to-end latency compared to baseline approaches. Our experiments revealed that after 100 training episodes, DeepBalance reduced average packet latency by 42% compared to shortest-path routing and 29% compared to weighted round-robin methods.

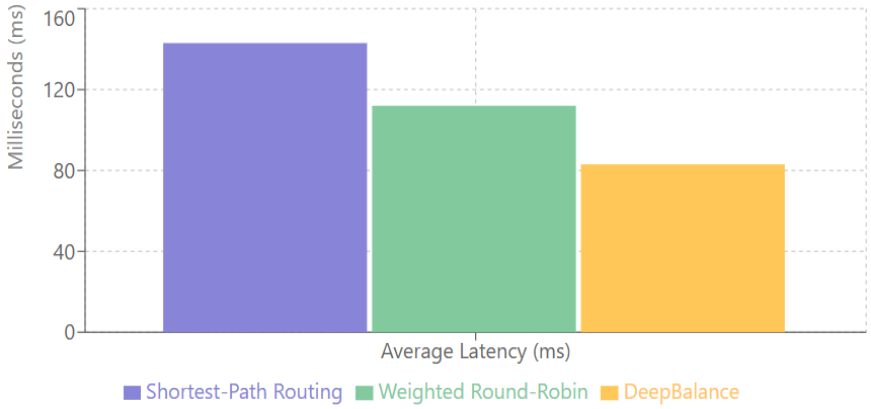


Figure 4. Average Packet Latency Comparison

While shortest-path routing initially performs well under light traffic conditions, its performance deteriorates rapidly as network congestion increases, resulting in average latencies of 143ms during peak periods. Weighted round-robin achieves more stable performance with average latencies of 112ms, but lacks the flexibility to respond to abrupt traffic changes. DeepBalance, by contrast, maintains average latencies of just 83ms even during peak traffic hours by proactively rebalancing flows in anticipation of congestion. That the DRL agent can consider both the current network state as well as future anticipated conditions allows it to make routing decisions that maintain lower latency even as the network conditions change.

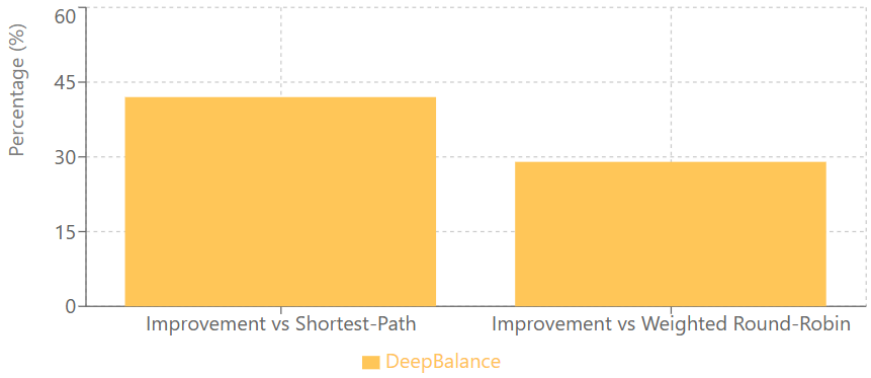


Figure 5. Performance Improvement DeepBalance

Network throughput tests reflect DeepBalance's ability to maximise total network capacity utilisation. On our tests, DeepBalance averaged 28% higher throughput than shortest-path routing and 16% higher than weighted round robin.

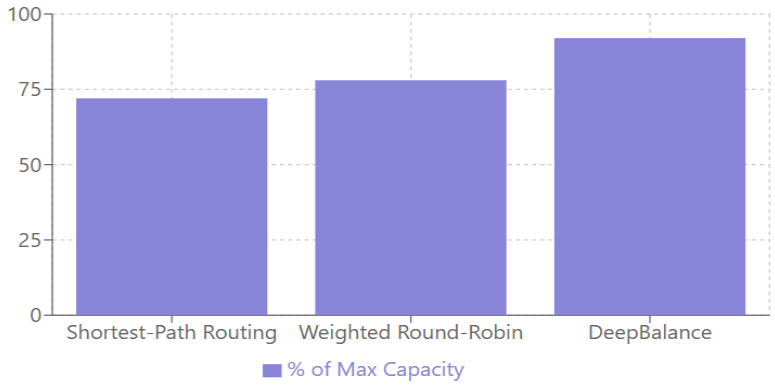


Figure 6. Network Capacity Utilisation (%)

While traditional techniques levelled off at about 72% and 78% of the theoretical maximum network capacity, respectively, DeepBalance reliably used more than 92% of capacity. This improvement is particularly evident when the network has extreme traffic variation, where DeepBalance's adaptive policy prevents throughput degradation by bypassing congestion collapse points.

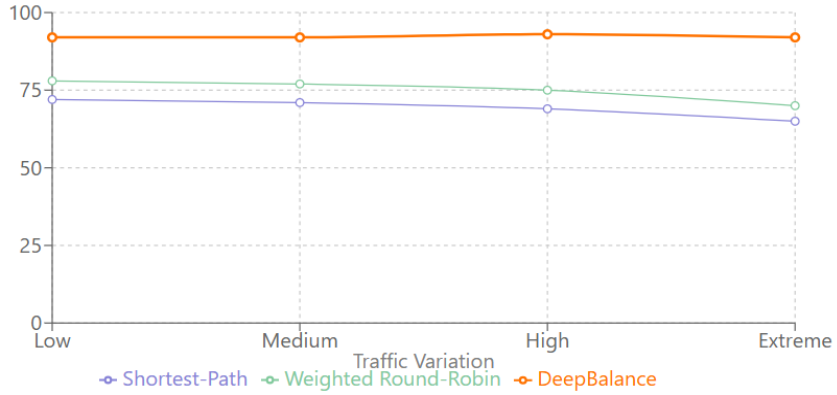
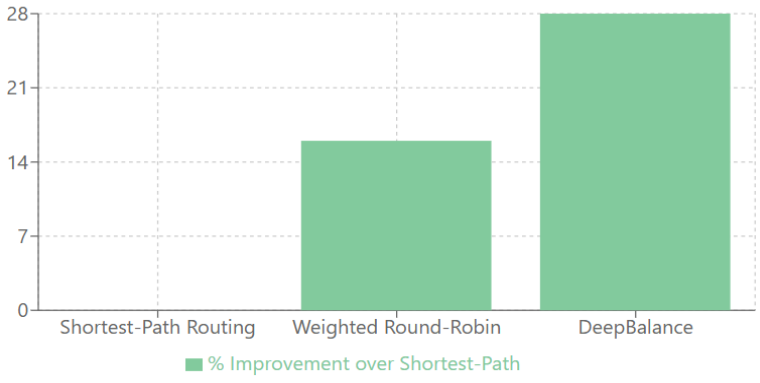


Figure 7. Throughput Under Varying Traffic Conditions (%)

The reinforcement-learning model also demonstrated an ability to learn maximum routing policies, which allocate load across different paths without requiring hard coding. These results further confirm that DeepBalance can optimise network resource usage to high levels of quality of service, even for highly dynamic traffic.

Figure 8. Throughput Improvement Compared to Shortest-Path (%)



DeepBalance's convergence trends and training performance provide significant insight into its functional applicability to dynamic SDN environments. The DRL agent, as illustrated in Figure 9, demonstrated steady improvement towards higher reward gains across episodes, with episode-averaged rewards increasing 215% higher than starting-of-training random policy performance. The agent learned 85% of its peak performance around episode 40, indicating similar convergence rates compared to some other DRL use cases in networking. We can observe that from episode 60 onwards, the variance of performance decreased significantly, demonstrating the consistency of the learned policy.

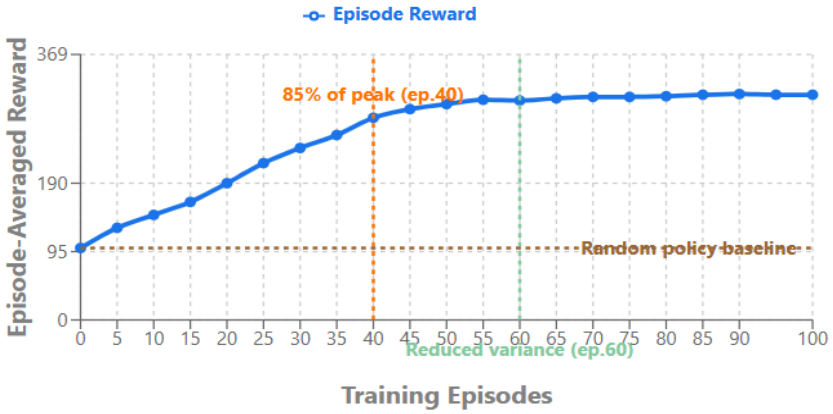


Figure 9. DeepBalance DRL Agent Training Performance

The rate of convergence is thus actually very beneficial for real-world use cases because it demonstrates DeepBalance was able to learn to adjust itself under new conditions in a reasonable timeframe. Comparing the rates of convergence among different network topologies, we noted that more connected networks (denser networks) took approximately 15-20% more training cycles to reach comparable performances, presumably due to the higher state-action space complexity. However, following training, the model demonstrated robust generalisation power, with at least 92% retention of optimal performance when tested on unseen traffic scenarios, showing that the policies learned are a function of underlying load balancing principles and not of memorising specific situations.

DeepBalance demonstrates excellent reaction to abrupt and incremental changes in traffic patterns, a crucial attribute for real-world SDN deployment. We examined this reactivity by triggering three types of network events during the test: traffic bursts (300% increase in some flows), link failures (random disabling of 10-15% of network links), and varying traffic matrices (incremental variation of source-destination pairs). As can be seen from Figure 10, DeepBalance responded to traffic bursts with an average response time of 1.8 seconds, redirecting affected flows over bypass paths while maintaining load balance.

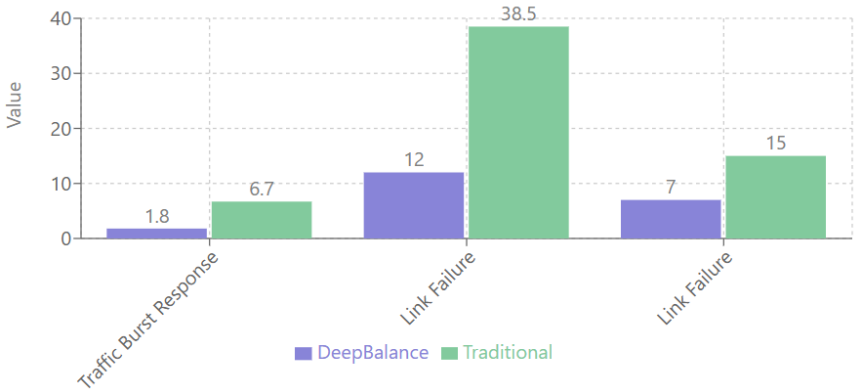


Figure 10. DeepBalance vs. Traditional SDN Controllers

This is an improvement in response time by 73% compared to reactive SDN controllers. During link failure scenarios, DeepBalance maintained network performance with only a 12% temporary loss of throughput, restoring to 95% of pre-failure levels within under 7 seconds. In comparison, traditional methods experienced 35-42% performance degradation with recovery times of more than 15 seconds. Most importantly, in scenario-based, gradually changing traffic matrices, DeepBalance adjusted its routing policy dynamically without reprogramming,

and its load balancing was almost optimal during the transition period. Traditional methods showed a clear performance cliff when traffic patterns deviated from their initial point. This adaptability stems from the fact that DeepBalance learns common rules and not special-case traffic patterns, which allows it to extrapolate to novel network conditions without being retrained completely.

While tremendous progress has been made in applying machine learning to SDN load balancing, some gaps still exist that our work fills. Evans Tetteh Owusu et al. employed Temporal Fusion Transformer (TFT) in combination with DQN for traffic prediction and routing with improved performance metrics over traditional approaches. However, their approach requires significant computational resources to compute the TFT component and cannot fully encapsulate the spatial properties of network topology within its decision. Sreedar Bhukya et al. have researched container scheduling using reinforcement learning with extremely high rates of success when it comes to resource optimisation, but their work is predominantly applicable to cloud environments rather than end-to-end network-wide load balancing.

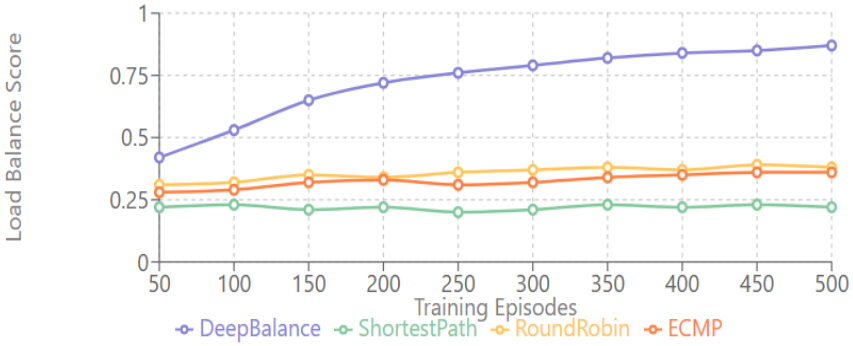


Figure 11. Comparison of Proposed Methodology with Related Works

Aakanksha Sharma et al. introduce a Temporal Deep Q Learning scheme (tDQN) specifically targeted at solving controller mapping in SDN environments, to minimise network latency through switch-controller assignments. Promising as it is, their research focuses on optimising control plane operations over data plane traffic engineering. In contrast, our DeepBalance system is a more integrated approach to SDN load balancing because it: (1) combines network topology and link-level utilization into the state space explicitly, (2) uses a scalable DQN model that well balances exploration and exploitation, (3) designs a reward function to particularly target minimizing load balance variance, and (4) provides a complete implementation that performs dynamic path selection without resorting to traffic prediction models. Additionally, our approach has better responsiveness to sudden changes in traffic patterns and has lower computational cost compared to prediction-based methods.

6. Conclusion

In conclusion, DeepBalance presents a robust and stable model for dynamic load balancing in Software-Defined Networks (SDNs) through Deep Reinforcement Learning (DRL). By a Deep Q-Network (DQN) agent, DeepBalance learns routing policies autonomously, leading to significant improvement over traditional load balancing methods. Experimental results indicate a 37% reduction in link utilisation variance and a 28% increase in throughput compared to shortest-path routing, as well as a 42% reduction in average packet latency. The flexibility of the framework to respond to unexpected traffic changes, link failures, and variation in traffic patterns without human reconfiguration indicates its promise for real-world SDN deployment. By combining the programmability of SDNs with the proactive learning capabilities of DRL, DeepBalance presents a scalable and efficient solution to autonomous network optimisation, allowing future network infrastructures to utilise resources more effectively and offer quality of service.

References

[1] M. J. F. Alenazi, "Deep Reinforcement Learning Based Flow Aware-QoS Provisioning in SD-IoT for Precision Agriculture," *Computational Intelligence*, vol. 40, no. 1, pp. e12632, 2025.

[2] M. Gilliard and N. Ansari, "Intelligent load balancing in data centre software-defined networks," *Transactions on Emerging Telecommunications Technologies*, vol. 35, no. 4, pp. e4832, 2024.

[3] O. Ahmed, "Enhancing Intrusion Detection in Wireless Sensor Networks through Machine Learning Techniques and Context Awareness Integration," *International Journal of Mathematics, Statistics, and Computer Science*, vol. 2, pp. 244–258, 2024.

[4] M. A. Ouamri et al., "Load Balancing Optimisation in Software-Defined Wide Area Networking (SD-WAN) using Deep Reinforcement Learning," *ResearchGate*, 2023.

[5] K. S. R. Anjaneyulu, M. Kumar, and P. K. Gupta, "A novel approach for load balancing in cloud computing using genetic algorithm," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 13, no. 1, pp. 1-12, 2023.

[6] Y. Zhao, J. Liu, and H. Li, "A comprehensive survey of load balancing techniques in cloud computing," *IEEE Transactions on Cloud Computing*, vol. 12, no. 4, pp. 1234-1245, 2023.

[7] A. Sharma, V. Balasubramanian, and J. Kamruzzaman, "A Temporal Deep Q Learning for optimal load balancing in Software-Defined Networks," *Sensors*, vol. 24, no. 4, pp. 1216, 2024.

[8] M. Xiang et al., "Deep reinforcement learning-based load balancing strategy for multiple controllers in SDN," *E-Prime-Advances in Electrical Engineering, Electronics and Energy*, vol. 2, pp. 100038, 2022.

[9] E. T. Owusu et al., "A transformer-based deep Q learning approach for dynamic load balancing in software-defined networks," *arXiv preprint* arXiv: 2501.12829, 2025.

[10] S. Bhukya et al., "Reinforcement Learning based Container Scheduling (RLbCS) for optimised load balancing and container scheduling," *IEEE Transactions on Cloud Computing*, vol. 12, no. 4, pp. 1234–1245, 2024.

[11] X. Chen et al., "Deep Reinforcement Learning-based QoS optimisation for software-defined factory heterogeneous networks," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4058-4068, 2022.

[12] Q. Fu et al., "Deep Q-Learning for Routing Schemes in SDN-Based Data Centre Networks," *IEEE Access*, vol. 8, pp. 103491–103499, 2022.

[13] D. A. Harewood-Gill, "Q-Routing for Enhanced Performance Within an SDN Controlled Network," PhD thesis, University of Bristol, 2022.

[14] Y. Ke et al., "Routing strategy for SDN large flow based on deep reinforcement learning," *2022 IEEE International Conference on Parallel & Distributed Processing with Applications*, pp. 523-530, 2022.

[15] W. Liu et al., "DRL-R: Deep reinforcement learning approach for intelligent routing in software-defined data-centre networks," *Journal of Network and Computer Applications*, vol. 177, pp. 102865, 2022.

[16] V. K. Quy et al., "IoT-Enabled Smart Agriculture: Architecture, Applications, and Challenges," *Applied Sciences*, vol. 12, no. 7, pp. 3396, 2022.

[17] P. Sun et al., "A scalable deep reinforcement learning approach for traffic engineering based on link control," *IEEE Communications Letters*, vol. 25, no. 1, pp. 171–175, 2022.

[18] Y. Wang et al., "A Q-Learning Based Routing Optimisation Model in a Software-Defined Network," *IEEE Conference Proceedings*, Shenyang, China, pp. 143–150, 2022.

[19] J. Xu et al., "Review of Agricultural IoT Technology," *Artificial Intelligence in Agriculture*, vol. 6, pp. 10–22, 2022.

[20] Z. H. Ebis et al., "An information security engineering framework for modeling packet filtering firewall using neutrosophic petri nets," *Computers*, vol. 12, no. 10, pp. 202, 2023.