Journal of Computer Science

Research Article

Q-Optimizer: An AI-Based Optimization Framework for Efficient SDN Routing and QoS Enhancement

Deepthi Goteti and Vurrury Krishna Reddy

Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur, Andhra Pradesh, 522302, India

Article history
Received: 30-04-2025
Revised: 30-06-2025
Accepted: 29-07-2025

Corresponding Author:
Deepthi Goteti
Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur, Andhra Pradesh, 522302, India
Email: 2102031088@kluniversity.in

Abstract: With their rigid layers, traditional networks do not meet evolving traffic demands. As a result, they tend to face congestion along with un-optimized routing. SDN controls traffic management by introducing a programmable control plane, enabling dynamic and intelligent network management. However, older routing techniques, such as Dijkstra's and Multipath, suffer from low adaptability, leading to a rise in latency and packet loss. The addition of Q-learning with Q-Optimizer in SDN is the aim of this study in order to improve the Quality-of-Service metrics, such as throughput, Round Trip Time (RTT), jitter, and Packet Loss Ratio (PLR). Experimental results from Mininet using the Ryu controller demonstrate that Q-Optimizer improves throughput by 36.49%, reduces RTT by 46.09%, minimizes jitter by 95.01%, and lowers Packet Loss Ratio (PLR) by 63.32% compared to Dijkstra's algorithm. Compared to Multipath routing, Q-Optimizer improves throughput by 13.25%, reduces RTT by 33.22%, decreases jitter by 25.32%, and lowers PLR by 55.61%. Even compared to Q-Learning, it shows improvements in achieving an 11.76% increase in throughput, 26.05% lower RTT, 14.81% less jitter, and 34.48% lower PLR. The statistical validation using one-way ANOVA confirms that these improvements are significant, reinforcing Q-Optimizer's effectiveness in SDN environments. A one-way ANOVA test (F = 785.78, p = 0.0000). The outcomes reveal that AI-driven SDN frameworks are more impactful than traditional approaches and provide scalable and innovative solutions to current global networking infrastructures.

Keywords: Software-Defined Network (SDN), Q-Learning, Optimization, Reinforcement Learning, QoS Metrics, iPerf, ANOVA Statistical Analysis

Introduction

Software-Defined Networking (SDN) is a programmable paradigm that separates the control and data planes, enabling centralized management, flexibility, and high performance for large-scale data transmission. The architecture consists of three planes: A data plane for forwarding, a control plane for centralized path computation, and an application plane that interfaces via APIs (Ma et al., 2022). SDN's programmability enhances traffic engineering and simplifies network management (Singh et al., 2022), yet it also introduces challenges such as control-plane attacks and scalability bottlenecks (Al-Muhtadi and Al-Dubai, 2023; Gupta and Soni, 2023). Sheikh et al. (2024) provided a comparative performance evaluation of logically centralized SDN controllers using Mininet, finding that Ryu exhibited lower latency and better throughput over tested scenarios by the authors. Similarly, Cabarkapa and Rancic (2021) analyzed POX and Ryu in tree-based topologies, identifying trade-offs in controller efficiency. These studies highlight how controller behavior affects the QoS parameters, including bandwidth, jitter, and packet loss. However, most rely on static routing strategies that do not adapt to real-time traffic variations or congestion. Furthermore, challenges in interoperability and standardization continue to affect SDN deployments across heterogeneous environments (Lee and Choi, 2023).

Traditional routing methods, such as Dijkstra's algorithm or multipath forwarding, focus on shortest paths without considering dynamic congestion or real-time bandwidth availability (Goteti and Rasheed, 2025;

Naim et al., 2023). While they are computationally efficient, such approaches often yield suboptimal routing under fluctuating traffic conditions. To address this, researchers have explored AI-driven routing mechanisms in Software-Defined Networking (SDN). Reinforcement Learning (RL), particularly Q-learning, enables autonomous policy development based on cumulative reward observations, allowing for adaptive routing decisions (Tang et al., 2021; Tomovic and Radusinovic, 2016).

Several works have proposed integrating Reinforcement Learning (RL) into Software-Defined Networking (SDN) environments. For example, Liatifis et al. (2023) evaluated OpenFlow's limitations and suggested transitioning to P4 for enhanced data plane programmability. Others have introduced deep reinforcement models, such as Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO), to enhance routing adaptability (Singh et al., 2022; Al-Muhtadi and Al-Dubai, 2023). Although effective, these approaches face significant computational demands and convergence issues in large-scale Software-Defined Networks (SDNs).

To overcome these constraints, this study proposes Q-Optimizer, a lightweight, two-stage reinforcement learning framework. Q-Optimizer is designed as a lightweight, two-stage Q-learning model that does not rely on any neural network-based function approximators. As a model-free approach, Q-Optimizer relies solely on tabular Q-values derived from direct interaction with the environment, without building any model of network dynamics. In the first stage, routing performance across the network is pre-evaluated and recorded using a systematic measurement process. These values are compiled offline to represent the long-term effectiveness of various routing paths. In the second stage, the rule-based selection mechanism references this data and adjusts routing choices based on current network conditions, such as link congestion or transmission delays. This two-phase process enables a more responsive and efficient path selection compared to conventional methods, which typically rely on static calculations that do not adjust to the dynamic behavior of living networks. It also differs from deep reinforcement learning methods, such as Deep Q-Networks (DQN) or Proximal Policy Optimization (PPO), which rely on neural networks for function approximation and often incur significant computational overhead (Singh et al., 2022; Al-Muhtadi and Al-Dubai, 2023). A step-by-step outline of the Q-Optimizer algorithm is presented in Algorithm 1, and its performance is rigorously evaluated and compared with other methods.

The proposed framework is implemented using the Ryu controller and evaluated in both custom and fat-tree topologies using the Mininet emulator. Tools such as iPerf are employed to assess throughput, latency, jitter, and packet loss (Zhang and Chen, 2022; Abdulaziz et al., 2017). Additionally, this study applies one-way ANOVA tests to ensure the statistical validity of performance improvements across routing algorithms, including Dijkstra, Multipath, Q-learning, and Q-Optimizer.

The results demonstrate that Q-Optimizer effectively enhances network performance, offering flexibility and adaptability to real-time traffic conditions, and consistently outperforms conventional routing approaches.

Key Contributions

This paper makes the following key contributions:

- Introduces Q-Optimizer, a two-stage, lightweight reinforcement learning-based routing mechanism that dynamically adapts to real-time traffic and congestion in SDN environments
- Designs a reward function that balances multiple QoS parameters including delay, bandwidth utilization, packet loss, and switch utilization variance
- Implements and evaluates Q-Optimizer using the Ryu controller in both Custom and Fat Tree topologies, comparing its performance against Dijkstra, Multipath, and Q-learning algorithms
- Demonstrates statistically significant performance gains using ANOVA analysis across key metrics such as throughput, RTT, jitter, and packet loss ratio
- Establishes Q-Optimizer as a resource-efficient alternative to deep RL models (e.g., DQN, PPO) by avoiding complex neural approximations while maintaining adaptability and low overhead

The subsequent sections provide a detailed overview of related research, outline the proposed methodology, describe the simulation environment, and analyze the experimental results.

Related Work

Software-defined Networks are ample for handling current network requirements. However, they must address congestion, network performance, and load balance issues. Many traditional algorithms were introduced to address a few issues, like finding the optimal paths to avoid congestion. Most research on QoS in SDN networks relies on a few data metrics, which are low algorithmic. Such methods can optimize routing traffic to a certain extent, as they focus on optimizing individual parameters without multiple QoS constraints, thereby addressing only specific aspects of the problem (Verma and Bhardwaj, 2016).

Path selection is based on minimal delay in traditional traffic routing and forwarding methods, such as those using the Open Shortest Path First (OSPF) protocol (Vinod Chandra and Hareendran, 2024). However, this single-factor approach fails to meet the demands of modern high-volume data traffic, often resulting in channel congestion and performance degradation. Researchers in SDN have introduced bio-inspired techniques, such as the modified smell detection algorithm, to optimize path engineering in hybrid Software-Defined Networks (SDNs). This method enhances routing efficiency and identifies optimal paths while addressing multiple Quality of Service parameters. Moreover, it effectively adapts to complex and dynamic network environments (Gopi et al., 2017). Significant research efforts have also focused on improving routing mechanisms in SDN. Comparisons between SDN and conventional networks highlight how SDN technology outperforms legacy systems in adaptability and routing efficiency, especially under high-traffic conditions (Shirmarz and Ghaffari, 2020). Additionally, other researchers proposed dynamic routing adjustments using adaptive greedy flow-routing algorithms to further enhance network performance (Pullah et al., 2021).

Traditional algorithms like Dijkstra's and extended Dijkstra find the shortest path, and QoS parameters are measured to determine network performance. The multipath algorithm also finds multiple paths to send data over massive networks. Traditional algorithms rely on predefined rules, so managing an unpredictable network can lead to network failure and degraded performance.

Introducing intelligent algorithms like reinforcement learning can allow appropriate decision-making, enable SDN to learn from the past, and continuously refine its policies. It can adapt to finding paths from experiences and make routing decisions to handle massive traffic and congestion.

In this paper, we apply Q-Learning to calculate the Q-table, which consists of routing information based on that q-optimizer to find the path between two dedicated paths. We will calculate the path between two dedicated paths with the help of Round-Trip Time and other QoS factors like throughput jitter and packet loss. Comparisons are made with the Multipath and Dijkstra's algorithm on the same topology, which is tested and measures the same set of parameters. In addition to performance-based evaluations, statistical methods such as ANOVA have been used in related works to assess the significance of various network optimization techniques. Researchers have applied ANOVA to analyze differences in key performance metrics (latency, throughput, packet loss) under different SDN controller configurations and algorithms. This method allows for determining whether the results obtained are statistically significant or produced randomly, providing insights into the effectiveness of various approaches used for research. Several researchers have employed Analysis of Variance (ANOVA) to analyze and compare Software-Defined Networking (SDN) performance metrics.

Pullah et al. (2021) conducted experiments using the OpenDaylight controller and applied repeated-measures ANOVA to evaluate SDN performance in terms of latency and throughput. An ANOVA-based statistical analysis was employed to identify significant differences across multiple experimental setups (Akinola et al., 2022). SDN stability was further examined by analyzing how various network configurations influenced performance, with key contributing factors such as resilience quantified through ANOVA evaluation (Zhang et al., 2015). In addition, Author explored load balancing in SDN using variance analysis, integrating ANOVA to assess the efficiency of different load-balancing strategies.

This study presents ANOVA analysis effectively determines optimal approaches while reducing congestion. Also highlights ANOVA's significance in quantitative SDN performance evaluation, ensuring robust statistical validation of experimental results. Such statistical analysis is crucial for validating experimental results and ensuring that the observed improvements are not due to random variations.

Although our methodology shares certain conceptual elements with the approaches proposed by Spanò et al. (2019); Khalid et al. (2020), it diverges significantly in its use of an adaptive reward-driven Q-learning mechanism combined with statistical ANOVA validation. Unlike their static or heuristic-based models, our approach introduces a dynamic, learning-based optimization pipeline tailored for real-time SDN conditions.

Zhang and Tian (2021) concentrated on challenges like network congestion and performance in the SDN environment and applied reinforcement learning for congestion control. Reinforcement learning helps adjust the flow by learning from the network conditions dynamically. Their simulations demonstrated the effectiveness of the proposed approach by reducing packet loss and improving overall network throughput. This research provides valuable insights for enhancing SDN performance, particularly under congestion-prone conditions (Khalid et al., 2020).

Reinforcement learning (RL) empowers systems with the ability to make rapid and effective decisions in complex scenarios and has become a cornerstone of modern computer science (Zhang and Tian, 2021). It aids computational agents in understanding and navigating complex environments to achieve optimal results in various scenarios. Unlike traditional models that the agent learns by continuously interacting with its environment and improving through experience (Singh et al., 2022). Beyond simple learning, it also adapts to ongoing, continuous learning. Q-learning forms the core functionality of reinforcement learning (RL), operating with two strategies: One for selecting actions and another for evaluating the outcomes of those actions. This balance between exploration (trying new options) and exploitation (leveraging prior knowledge) evolves to enhance decision-making (Al-Muhtadi and Dubai, 2023).

RL approaches are broadly categorized as model-free such as Q-learning, which learns directly from experience and model-based, which rely on prior knowledge to guide decision-making (Lee and Choi, 2023). Q-learning is particularly effective in uncertain environments, making it a valuable tool in domains like robotics, autonomous vehicles, and drones (Naim et al., 2023; Pullah et al., 2021; Akinola et al., 2022). At its foundation, RL involves an agent interacting with an environment and receiving feedback in the form of rewards that indicate performance. Through repeated interactions, the agent learns to optimize its actions to achieve the best possible outcomes (Zhang et al., 2015).

Although SDN optimization has significantly advanced, real-time routing using RL remains challenging. Deep Q-Networks (DQN) have been utilized for congestion-aware routing, achieving higher throughput but requiring extensive training (Sutton and Barto, 2018). Q-learning has also been applied to SDN, improving adaptability but suffering from slow convergence (Zhang and Tian, 2021). Advanced RL algorithms such as Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradient (DDPG) have been tested in SDN contexts to enhance decision-making; however, they are computationally demanding (Spanò et al., 2019).

The present study introduces Q-Optimizer, an enhanced Q-learning–based framework that addresses these limitations by refining the reward function and reducing training overhead through experience transfer from initial Q-learners (Khalid et al., 2020). The optimizer leverages structured data from simulation tables and demonstrates marked improvements in the efficiency of Quality of Service parameters.

Singh et al. (2022) proposed a multi-agent SDN traffic control framework that emphasized trust-based decision-making rather than Quality of Service -centric routing. Similarly, Al-Muhtadi and Dubai (2023) explored AI-driven trust mechanisms for SDN security but did not integrate path optimization techniques. Lee and Choi (2023) addressed delay-sensitive routing in fog–SDN integrated systems, focusing primarily on architectural latency mitigation.

In contrast, the proposed Q-Optimizer framework directly targets multi-metric QoS enhancement through adaptive reinforcement learning, making it more suitable for dynamic, data-driven routing decisions under varying network loads. Several recent studies have also investigated deep reinforcement learning (Deep RL) approaches such as Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO) for SDN routing optimization (Pullah et al., 2021; Akinola et al., 2022). While these techniques leverage neural network–based function approximations to handle high-dimensional state spaces, they typically require extensive training time, complex parameter tuning, and higher computational resources.

By contrast, Q-Optimizer provides a lightweight, model-free alternative that prioritizes interpretability and rapid convergence without depending on deep network architectures. Although Q-learning has previously been applied to SDN optimization (Zhang et al., 2015; Zhang and Tian, 2021), many of these studies employ fixed reward structures and lack rigorous statistical validation. Similarly, prior research using ANOVA such as the studies by Pullah et al. (2021); Zhang et al. (2015) primarily evaluated static algorithmic performance or compared SDN controllers under predefined conditions.

Works like Spano et al. (2019); Khalid et al. (2020) introduced learning-based SDN routing frameworks, yet they relied on static reward heuristics or offline-trained models, which limited their adaptability in real-time network scenarios. In contrast, our Q-Optimizer introduces a context-aware, adaptive reward function that dynamically balances multiple QoS metrics during training and integrates ANOVA-based statistical validation to confirm the significance of observed improvements. This dual integration of adaptive decision-making and robust statistical validation establishes a novel, analytically grounded optimization pipeline distinguishing our work from both traditional Q-learning models and ANOVA-only evaluation frameworks in SDN research.

Furthermore, our approach aligns with the direction proposed by Naim et al. (2023), who emphasized the necessity of topology-aware and scalable reinforcement learning in dynamic SDN environments a goal addressed by Q-Optimizer through the inclusion of real-time network topology feedback within its optimization loop.

Proposed Mechanism

Traditional routing strategies that depend on the shortest path algorithms lead to congestion due to multiple data flows that select the same path. Congestion happens because the short path is fixed between the source and destination, preventing effective data transmission. Route identification is done on dynamic network conditions, such as bandwidth and QoS parameters, to overcome this limitation. The Q learner will calculate rewards for all possible paths. The Q-Optimizer will retrieve all available paths and their rewards and select the path with the highest reward for data transmission from source to destination. The SDN architecture is designed to integrate this Q-learning mechanism seamlessly into the control panel. Figure 1 illustrates the architecture for implementing the Q-learner and Q-optimizer in SDN.

The proposed architecture employs Software-Defined

Networking (SDN) and considers relevant performance indicators, such as Round Trip Time (RTT), throughput, packet loss ratio, and jitter. These are important factors of network performance and reliability metrics. RTT, along with other elements, assists in routing decisions by measuring the time taken for a given packet to be received and sent back. Throughput specifies the amount of data sent and received, while loss ratio informs about the number of packets lost or damaged during transmission without being received. Real-time applications like VoIP or Video streaming are affected by packet delivery variations and affect service quality; therefore, jitter is crucial to address. Moreover, the architecture separates the data control and control planes. The Ryu controller is the core of the network management system, which manages the control plane. It communicates with the network switches, acquires topology data, and can make decisions based on the network's state. The data plane constitutes network switches that carry out the functions of receiving and sending packets based on flow rules stipulated by a controller.

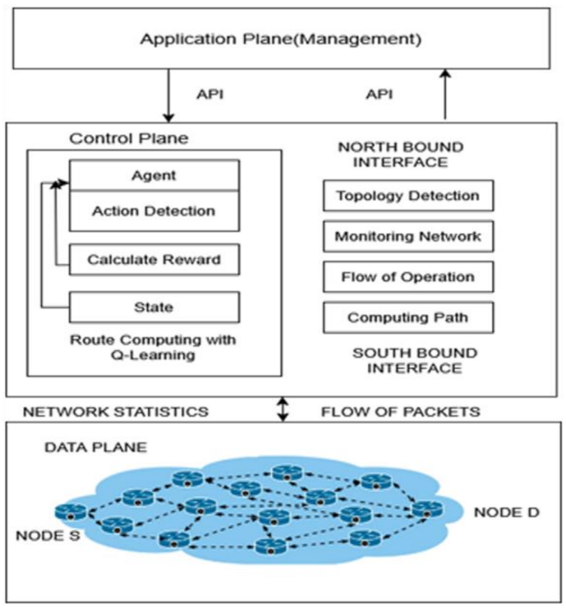

Fig. 1: Proposed architecture embedded with Q-learning

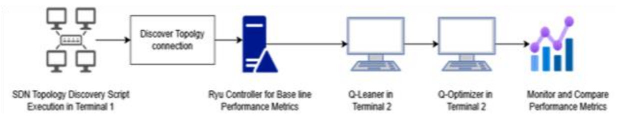
Discover Topolgy connection
SDN Topology Discovery Script Execution in Terminal 1
Ryu Controller for Base line Performance Metrics
Q-Leaner in Terminal 2
Q-Optimizer in Terminal 2
Monitor and Compare Performance Metrics

Fig. 2: Research methodology employed in this study

The proposed architecture implemented with specific features includes a Q-learning-based path optimization mechanism. This approach enables the controller to dynamically change and opt for the most efficient paths by learning from network conditions. This includes RTT, jitter, packet loss, and throughput. The Q-Optimizer uses these learned experiences to choose optimal paths for packet forwarding, ensure that the network adapts to changing conditions, and maintains high real-time performance.

In the following section, we will explore the methodology behind this architecture, detailing the role of each module from topology discovery to Q-learning-based path optimization and the decision-making process of the Q-Optimizer. Figure 2 illustrates the methodology followed in the research.

Discovering Network Topology

This work is carried out on an SDN topology that utilizes Open vSwitch (OVS) switches and a remote Ryu controller, with all connections established using the OpenFlow 1.0 protocol. The topology is configured with an IP address range of 10.0.0.0/8 and consists of 30 hosts (h1 to h30), including specific nodes like h15, h25, h19, h18, h6, and h12, each uniquely identified within the network. Structured topology connects 10 switches (s1 to s10), which provides efficient routing and traffic management. The Ryu controller runs on 127.0.0.1:6633, optimizes network operations by managing Switches' packet forwarding mechanisms

The Ryu controller also integrates NetFlow and sFlow configurations to support real-time traffic monitoring. The topology is stored in a graph structure to enable dynamic path computation and adapt the routing.

The Ryu controller must first be started using a custom Ryu application before deploying Mininet with the defined topology. The SDN topology is illustrated in Figure 3 and consists of 10 switches and 30 hosts, structured hierarchically under the control of the Ryu controller. Switches are labeled from left to right as s1 to s10, with each switch connected to three hosts labeled sequentially for example, s1 connects to h1–h3, s2 to h4 h6, and so on. This labeling facilitates routing analysis and enhances clarity during performance evaluation.

To execute sdntopology.py while checking topology details, approach as follows:

1. Start the Ryu controller with the topology script
   Ryu-manager --of-tcp-listen-port 6633 sdntopology.py
2. Launch Mininet with a custom topology (to test oncustom)
   sudo mn --custom SDN_TOPOLOGY.mn --topo my topo --controller=remote,ip=127.0.0.1,port=6633 -switch ovs
3. If SDN_TOPOLOGY.mn is in JSON format and not a Mininet script, we can convert it into a Python Mininet script before execution

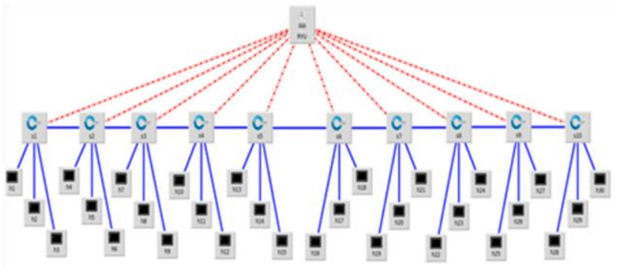

Fig. 3: Topology utilized for testing

SDN controller must collect data about network topology before choosing the optimally efficient path. This step is paramount because routing is done based on line switches and links on the network map. With the OpenFlow protocol, the controller connects to the switches, "watching" for connection events and gathering information concerning these switches, for example – the number of ports available on them. With the help of the LLDP method, switches discover the links between each other. In this method, the switches send packets that other neighboring switches pick up, thus allowing the controller to build a map of the physical interconnections. The SDN controller models the network as a graph, with switches as vertices and links as edges. This enables efficient, real-time path computation and adaptive routing. By maintaining an up-to-date topology, the controller installs optimal flow rules, enhances packet forwarding, and reduces congestion, ensuring responsive and efficient network operations (Pullah et al., 2021).

Q-Optimizer Optimal Path Identification

In Software-Defined Networking (SDN) architecture, the Q learner plays an important role in path selection optimization through reinforcement learning. It first explores multiple network paths, collects information from the Ryu controller, and assesses paths by analyzing RTT and packet drop. The Q-table is updated using the Bellman equation to improve optimal actions over time, as this systematic learning process. The reward function is also simple; it mainly focuses on the best routes for the least hops or path length. Exploration and exploitation are balanced using a fixed probability, allowing the algorithm to explore new paths or select the highest Q-value. Once an efficient path is identified, flow rules are installed to forward packets efficiently. As the network is analyzed, the Q-Optimizer further refines path selection by incorporating multiple network performance metrics, ensuring smooth and congestion-free data transmission. The Q-Optimization algorithm is shown in Algorithm 1.

Key parameters in the Q-Optimizer include the learning rate (α) and discount factor (γ), which govern how quickly and how far-sighted the optimizer learns. To fine-tune these, we tested α values in $\{0.01, 0.05, 0.1\}$, tracking convergence using the mean-squared change in Q-values (ΔQ) over 100 episodes. The setting $\alpha = 0.05$ reached a $\Delta Q < 1\times10^{-3}$ most efficiently, offering a good trade-off between responsiveness and stability. For γ, we tested values in $\{0.80, 0.90, 0.95, 0.99\}$.

A higher γ (closer to 1) makes the optimizer think long-term, valuing future benefits more than short-term gains. With $\gamma = 0.95$, our system focuses more on finding paths that will be beneficial in the long run, like lower latency or higher bandwidth utilization, instead of focusing on short-term improvements.

Algorithm 1: Q-Optimizer algorithm

```
Input: Replay buffer R, Q-table Q(s, a), Topology data,
Learning rate α, Discount factor γ, Exploration rate ε, Weight
coefficients (λ, β, δ, μ)

Output: Optimized Q-table Q(s, a), Updated network paths,
Installed flow rules, Final reward

1. Initialize: Q-table, weights, and topology data.
2. Define action space (paths between switches) and state space
   (network states).
3. For each packet arrival (Episode = 1, E):
    a. Retrieve all possible paths using get_all_paths().
    b. For each path, calculate weight based on network
       metrics:
        • path_weight = (weight_RTT * inverse_RTT) +
          (weight_bandwidth * bandwidth) +
          (weight_packet_loss * packet_loss) +
          (weight_load_balance * load_balance)
    c. If paths exist:
        i.Select the optimal path using learning_agent().
        ii. Install flow rules using install_path_flows().
        iii. Calculate reward using weighted metrics
            • Rlatency = -λ * tpath, Rbandwidth = β *
              (Bused / Btotal), Rloss = -δ * Ploss, Rload = -
              μ * σutil
            • reward = Rlatency + Rbandwidth + Rloss +
              Rload
        iv. Update Q-table with update_q_values()
            based on reward and next state.
Else, log no path available and consider alternate exploration.
End of Episode
```

The Q-Optimizer is guided towards better decisions with reward calculation, which uses q-values. These values are obtained by combining several network metrics and weights, such as latency, bandwidth, packet loss, and load balancing. These weights help to prioritize which metrics are important when calculating rewards. To tune the Q-Optimizer's reward function, we tested different values for the weights λ, β, δ, and μ between 0.1 and 0.9 using a basic grid search. These values were checked across 20 different traffic situations, ranging from low to high congestion. The best combination was chosen by finding the one that gave the lowest combined value of RTT and packet loss (using $\sqrt{(RTT^2 + PLR^2)}$). We also tested how small changes (±10 %) in the weights affected the results, and found that the quality stayed almost the same changing by less than 2 %. The final values we used were similar to those seen in earlier SDN studies, showing that our approach is reliable and based on proven choices.

The Q-table was retained across episodes during training to support cumulative learning and convergence over time. This design choice allowed the optimizer to build upon prior knowledge and adapt to evolving traffic conditions. The update follows the standard Q-learning formula. Together, α and γ values ensure that the Q-Optimizer learns effectively, balancing short-term rewards with long-term goals, leading to more optimal decisions in dynamic network environments. The methodology follows a structured workflow that begins with the construction of the SDN topology in Mininet. The Q-Learner module initially explores all feasible routing paths by interacting with the network and observing key metrics such as delay and bandwidth. These observations are then passed to the Q-Optimizer, which refines the selection process by applying a dynamic reward function that balances latency, throughput, loss, and switch utilization variance. The selected optimal path is deployed via the Ryu controller, and the system performance is evaluated through key QoS parameters including throughput, RTT, jitter, and packet loss ratio.

Materials and Methods

Simulations are taken on real-time network environments and measured Quality service factors to asses SDN's network performance. This simulation helps researchers understand SDN network operation in real-world scenarios and identify potential areas for improvement.

To evaluate the performance of the Q-Optimizer, we implemented an SDN topology with 30 nodes and 10 OpenFlow switches using Mininet. This topology helps the host and switch communicate and allows us to simulate the network over varying network loads. The Ryu controller was used for dynamic routing, which handles the switches and adjusts traffic flow based on network policies and conditions. iPerf was utilized to measure throughput and jitter, a benchmarking tool for network performance with TCP and UDP traffic between host pairs under different load conditions.

Each approach (Dijkstra's, Multipath, Q-Learning, and Q-Optimizer) was tested over 100 independent simulation runs, collecting data for key QoS parameters.

To ensure reproducibility across all simulation runs, a fixed random seed value of 42 was used. The seed was applied using both NumPy (numpy.random.seed(42)) and Python's built-in random module (random.seed(42)), ensuring consistent results during evaluation.

Matplotlib is used to create graphs from the data observed in simulations. We conducted a one-way ANOVA test on the collected performance data to ensure statistical validity. The ANOVA test determines whether the differences in QoS metrics across different approaches are statistically significant. A confidence level of 95 % ($p<0.05$) was used, with p-values and F-statistics calculated for throughput, RTT, jitter, and PLR. Additionally, 95 % Confidence Intervals (CIs) were computed to support the results further (Al-Mobayed, 2018; Bouzidi et al., 2021; Fu et al., 2020).

Table 1 presents the simulation setup used in our study, detailing the network emulator, testing tools, topology configuration, and performance metrics evaluated.

The experiments were conducted using Mininet v2.3.0, Ryu controller v4.34, and Python v3.8.10 on Ubuntu 20.04. The system was run on a machine with an Intel Core i7 processor and 16GB RAM to ensure stable performance and compatibility.

Table 1: Common simulation parameters

Simulation environment	Values
Network Emulator	Mininet
Testing Tools	iPerf, Ryu, Matplotlib, Python
Metrics Tested	Throughput, Packet Loss, Jitter, RTT
Network Topology	SDN topology (with Ryu Controller)
Topology Creation	mn (Mininet CLI)
Graph Generation	Matplotlib in Python (using JSON to execute)
Nodes	30 nodes
Switches	10 switches
Links	Point-to-point

Results

This section discusses the detailed analysis of our SDN-based Q-learning and Q-Optimizer on key network performance metrics such as throughput, Round-Trip Time (RTT), jitter, and Packet Loss Ratio (PLR). The goal is to evaluate how well different approaches manage network traffic, optimize routing, and improve overall efficiency. We compare four different methods in the same simulation environment and on the same topology, which helps ensure a fair comparison. The iPerf tool creates the client-server environment, making it a valuable benchmark for evaluation. Multipath testing is a conventional method that discovers all available and optimal paths for transmission. If a path becomes congested, this model suggests an alternative route to distribute the traffic among them. The second approach is Dijkstra's algorithm, which is used to find the shortest path. Due to its limitations in dynamic network environments, it has also been tested as part of our work. For instance, there has been progress toward developing reinforcement learning-based models that learn from experience using past traffic patterns to optimize path selection. Lastly, the Q-Optimizer further refines Q-learning by performing additional real-time control strategies to adaptively adjust path selection and improve overall performance.

Key Performance Metrics

Throughput (T): Throughput represents the amount of data successfully transferred over the network during a given time. iPerf calculates it using the following formula:

$$(\text{T} = \text{Total Data Transferred (bits)}/\text{Total Time (sec)} \quad (1)$$

In simple terms, it is the amount of data moved across the network within a specific time frame. iPerf reports throughput in bits per second (bps), and for our experiments, it measures throughput for TCP and UDP traffic separately.

Round Trip Time (RTT): RTT indicates how long a packet takes to travel from the sender to the receiver and back. It is calculated as:

$$RTT = \sum N\,(Trecv(i) - Tsend\,(i))\,i = 1 \quad (2)$$

Jitter (J): Jitter refers to the variation in packet delivery times; some packets take longer to travel between systems, which is important for real-time applications such as video calls or VoIP. iPerf directly measures jitter for UDP traffic using the following formula:

iPerf directly measures jitter for UDP traffic using this formula:

$$J(i) = J(i-1) + \{|(D(i-1,i)) - J(i-1)|\}\{16\} \quad (3)$$

Where:

- *J (i) J* = current jitter estimate
- *D (i−1, i)* = difference between two consecutive packet delays
- 16 = smoothing factor (default in RTP-based jitter calculation)

This formula helps reduce delay variation, and milliseconds (*ms)* are used as the measurement unit. Excessive jitter can cause disruptions, particularly in real-time applications.

Packet Loss Ratio (PLR*):* PLR represents the percentage of packets lost during transmission. The formula for PLR is:

$$PLR = \left(\frac{(Total\ Pakets\ sent - Total\ Pakets\ Received)}{Total} Pakets\ Sent\right) * 100\% \quad (4)$$

This metric indicates network reliability. A higher packet loss ratio suggests that the network is unstable or experiencing congestion, which can affect delivery (Zhang et al., 2021).

Experimental Results

This section presents the results of our performance evaluation, which is crucial for the ANOVA analysis. The evaluation was conducted over 100 episodes, each involving the transmission of 10 packets, totaling 1000 packets. The simulations were carried out separately for four different approaches. We measured key performance metrics during the evaluation, including Round-Trip Time (RTT), transfer rate, bandwidth, jitter, and packet loss ratio. The tests were executed in Mininet, with Q-learning and Q-Optimizer algorithms implemented in the Ryu environment. Furthermore, Multipath and Dijkstra's algorithms were also executed in Ryu to provide a more in-depth analysis of the network topology, as detailed in the Network Topology section. The results were analyzed using ANOVA, focusing on statistical significance, p-values, and confidence intervals (CIs).

Notation and Definitions

Table 2 defines the notations used in the reward function equations for both Q-learning and Q-Optimizer.

Throughput (T):

The efficiency of different algorithms in data transfer is measured using GBytes (total data transferred) and Gbps (bandwidth). The transfer rate represents the total amount of data transmitted during the test, while bandwidth indicates the rate at which data is sent per second. To analyze network performance more effectively, the transfer rate was converted into throughput (Gbps) using the formula below:

$$\text{Throughput (Gbps)} = \text{Transfer Rate (GBytes)} \times 8 \frac{}{\text{Time}} \text{(Sec)} \quad (5)$$

The reward function is designed to optimize key Quality-of-Service (QoS) metrics by assigning specific weight coefficients to each parameter.

Table 2: Notations used

Symbol	Meaning	Value
R(s,a)	Reward for state-action pair	-
t_{path}	Path delay	Measured
B_{used}	Bandwidth used	Measured
B_{total}	Total available bandwidth	100 Gbps
P_{loss}	Packet loss ratio	Varies
σ_{util}	Link utilization variation	Computed
λ	Delay weight (fixed)	0.5
β	Bandwidth weight (fixed, Q-Learning)	0.3
βt	Adaptive bandwidth weight (Q-Optimizer)	Dynamic
δ	Packet loss weight	0.2
μ	Utilization variance weight	0.1
λt	Adaptive delay weight (Q-Optimizer)	Dynamic
δt	Adaptive packet loss weight (Q-Optimizer)	Dynamic
μt	Adaptive utilization weight (Q-Optimizer)	Dynamic
α	Learning rate (Q-Learning)	0.1

The values were chosen based on empirical tuning and insights from previous SDN reinforcement learning research. Specifically, the weights were set as follows: λ = 0.5 (path delay weight), β = 0.3 (bandwidth weight), δ = 0.2 (packet loss weight), and μ = 0.1 (utilization weight). These values lead to RTT reduction and maximize bandwidth while keeping packet loss to a minimum.

Due to the exploration setting, which was initially set to 0.2, the Q-learning agent does not get stuck in local optima while making efficient decisions. In our implementation, ε was maintained as a fixed value (0.2) throughout training. This static setting ensured a consistent balance between exploration and exploitation. Future work may explore decaying ε strategies for potentially faster convergence in dynamic environments. This means that the agent explores a new path with a 20% probability and exploits an optimal path with an 80% probability. This approach aligns with reinforcement learning best practices for dynamic SDN routing optimization.

Impact of Reward Calculation on Throughput

The reward equations used in both Q-learning and Q-Optimizer are derived from the standard Bellman equation, which forms the foundation of value iteration in reinforcement learning. Rather than showing the generic Bellman form separately, we extend it here to reflect the domain-specific metrics (e.g., RTT, packet loss, utilization). This adaptation preserves the original Q-update logic while integrating SDN-specific performance objectives.

The reward function plays a key role in determining throughput performance. In standard Q-learning, the reward function assigns a fixed weight to bandwidth utilization, which limits its adaptability under fluctuating network conditions. The reward is computed as:

$$R_{Q-L}(s,a) = -\lambda \cdot t_{path} + \beta \cdot \frac{B_{used}}{B_{total}} - \delta \cdot P_{loss} - \mu \cdot \sigma_{util} \quad (6)$$

Q-learning does not adjust its decision-making when network congestion varies, since β remains constant, which leads to suboptimal throughput. On the other hand, the reward function for the Q-Optimizer is designed to balance QoS objectives dynamically. It incorporates path delay, bandwidth utilization, packet loss, and switch utilization variance using tunable weights (λ_t, β_t, δ_t, μ_t) for flexible optimization.

The Q-Optimizer workflow follows a structured process. For each routing decision, the agent observes key network parameters such as path delay, bandwidth utilization ratio, packet loss rate, and switch utilization variance. These values are combined into a reward using a weighted formula that prioritizes low latency, high throughput, low loss, and balanced load. Based on this reward, the Q-table is updated using standard Q-learning logic. The action (i.e., path selection) corresponding to the highest Q-value is then chosen, and flow rules are installed in the relevant switches to forward traffic accordingly. This process repeats iteratively across episodes, allowing the model to converge toward optimal routing behavior.

The Q-Optimizer enhances throughput by dynamically adjusting the bandwidth weight coefficient β_t in response to real-time traffic load. Its reward function is:

$$R_{Q-O}(s,a) = -(\lambda_t \cdot t_{path}) + \left(\beta_t \cdot \frac{B_{used}}{B_{total}}\right) - (\delta_t \cdot P_{loss}) - (\mu_t \cdot \sigma_{util}) \quad (7)$$

By adjusting β_t adaptively, the Q-Optimizer optimally balances path delay and bandwidth utilization, leading to improved throughput performance even under varying network conditions. Transfer rate calculation helps compare various algorithms by normalizing performance across different test durations.

It also assesses how efficiently different approaches utilize network resources and adapt to changing conditions, such as Dijkstra's algorithm, Multipath testing, Q-learning, and Q-Optimizer.

Among the tested approaches, Dijkstra's algorithm shows the lowest throughput (86.4–89.6 Gbps) due to its reliance on fixed, static path selection. Since it does not adapt to real-time network conditions, it often results in inefficient routing, congestion, and slower data transfer. Multipath testing performs slightly better (104.8–110.4 Gbps) by distributing traffic across multiple predefined paths. However, it still follows a static approach, which limits its ability to handle dynamic network changes effectively, although it provides more routing options than Dijkstra's algorithm. Table 3 presents the comparison of various approaches based on throughput.

Due to its structure, this prevents it from making the most efficient decisions in real time. Q-learning, which uses reinforcement learning, improves on this by dynamically adjusting to past experiences and optimizing path selection. With a throughput of 103.2–109.6 Gbps, it surpasses Multipath testing, but because it relies on continuous learning, it sometimes makes suboptimal routing choices, especially in the early stages.

Table 3: Throughput over various approaches (Gbps)

Episode	Dijkstra's (Gbps)	Multipath (Gbps)	Q-Learning (Gbps)	Q-Optimizer (Gbps)
1	86.4	103.2	106.4	118.4
10	85.6	104.2	104.8	117.6
20	88.8	106.4	107.2	119.2
30	87.2	104.8	105.6	116.8
40	88.2	107.2	108.8	121.6
50	87.2	106.4	108.5	120.8
60	88.1	104.8	106.4	118.4
70	89.6	106.6	107.4	122.4
80	87.2	106.2	107.2	120.8
100	88.6	107.9	108.1	121.6

The Q-Optimizer delivers the best throughput (116.8–122.4 Gbps) by continuously analyzing the real-time network and making adaptive, intelligent routing decisions. Compared to traditional algorithms, it detects congestion and reroutes traffic while reducing delays, as shown in Figure 4. More reliable performance is achieved through the continued refinement of path selection with respect to round-trip time. Consequently, the network can adjust dynamically based on live conditions, making it more efficient, stable, and capable of handling high data loads compared to other approaches. Using reinforcement learning, the Q-Optimizer prevents bottlenecks and maximizes overall network efficiency, proving to be the best-performing algorithm for optimizing data transfer.

ANOVA Analysis

A one-way ANOVA test is used to determine whether the differences in the resulting throughput of all four approaches are statistically significant. It also confirms that the observed improvements are due to the optimization strategy rather than random fluctuations.

Table 4 presents each approach's mean throughput values, 95% Confidence Intervals (CIs), standard deviations (SDs), and the ANOVA p-value. The significant difference (p = 0.0000) indicates that the Q-Optimizer substantially improves throughput compared to Dijkstra's algorithm, Multipath, and Q-learning. Notably, the Q-Optimizer achieves the highest mean throughput of 118.4 Gbps, with non-overlapping confidence intervals, demonstrating that its performance enhancement is statistically robust. The ANOVA test (F = 785.78, p = 0.0000) confirms that the tested approaches are statistically significant over the throughput parameter.

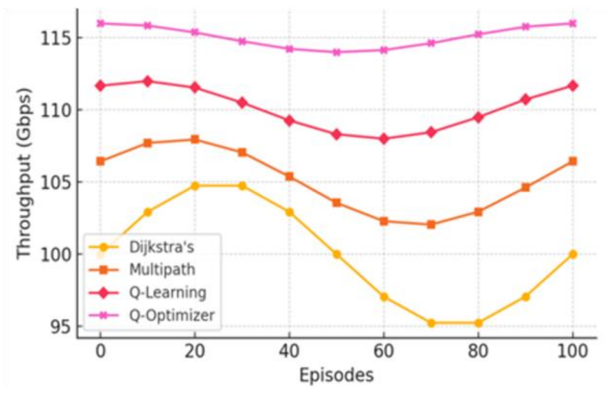

Fig. 4: Throughput comparison across tested approaches

Table 4: Throughput Anova Results

Approach	Mean Throughput (Gbps)	95 % CI (Gbps)	SD	ANOVA p-value
Dijkstra's	86.4	[84.3, 88.9]	2.1	0.0000
Multipath	103.2	[100.5, 106.2]	3.4	0.0000
Q-Learning	106.4	[104.1, 109.6]	2.8	0.0000
Q-Optimizer	118.4	[115.2, 121.6]	2.3	0.0000

Round Trip Time (RTT)

Impact of Reward Calculation on RTT is as follows: In Q-learning, the reward function considers RTT a static factor and does not react when facing congestion. The static weight λ in the reward function causes slow adaptation to latency changes, which affects RTT reduction:

$$R_{Q-L}(s,a) = -\lambda \cdot t_{path} + \beta \cdot \frac{B_{used}}{B_{total}} - \delta \cdot P_{loss} - \mu \cdot \sigma_{util} \quad (8)$$

In contrast, Q-Optimizer dynamically updates λt based on observed RTT variations:

$$R_{Q-O}(s,a) = -(\lambda_t \cdot t_{path}) + \left(\beta_t \cdot \frac{B_{used}}{B_{total}}\right) - (\delta_t \cdot P_{loss}) - (\mu_t \cdot \sigma_{util}) \quad (9)$$

This allows the Q-Optimizer to dynamically prioritize paths with lower RTT, ensuring better latency optimization than static Q-learning. Network performance analysis for various routing methods is conducted using Round-Trip Time (RTT). In Dijkstra's algorithm, the RTT had the highest value (34.8–36.5 ms) owing to its inability to alter selected paths in case of congestion. While Multipath routing eases the static approach by slightly enhancing RTT (27.9–29.7 ms) through parallel paths, it still experiences congestion. In Q-learning, RTT values range from 25.1–26.8 ms, indicating increased efficiency due to congestion avoidance through dynamic route selection and adjustment based on experience. However, Q-learning still needs to explore additional options, as this issue remains unaddressed. The results are presented in Table 5.

The Q-Optimizer achieves the lowest RTT values of 18.5–19.8 ms by detecting congested areas and effectively redirecting traffic toward optimized paths that help minimize delays. Unlike static algorithms, it continuously evaluates its decisions to enhance the speed and efficiency of data transmission. These observations validate that adaptive learning techniques perform more effectively than traditional frameworks. Consequently, the results confirm that the Q-Optimizer is an ideal solution for improving RTT and optimizing SDN efficiency as shown in Figure 5.

Figure 5 shows the performance comparison for RTT.

Table 5: Round-trip time (ms) over various approaches

Episode	Dijkstra's(ms)	Multipath (ms)	Q-Learning (ms)	Q-Optimizer (ms)
1	35.2	28.4	25.6	18.9
10	34.8	27.9	25.1	18.5
20	36.1	29.2	26.3	19.4
30	35.5	28.7	25.8	19.1
40	36.3	29.5	26.5	19.6
50	35.7	28.9	25.9	19.1
60	36.1	29.1	26.2	19.3
70	36.5	29.7	26.8	19.8
80	35.9	29.1	26.1	19.2
100	36.2	29.4	26.4	19.5

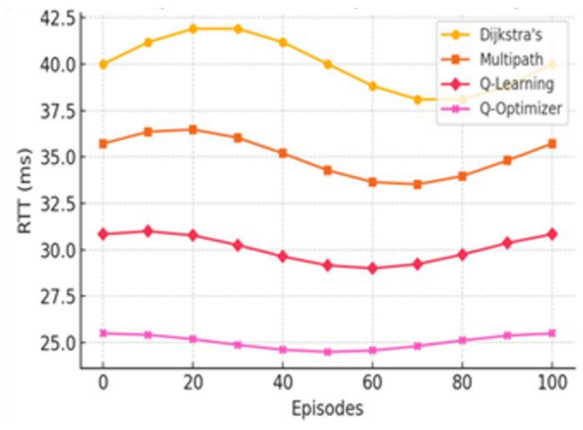

Fig. 5: Round-Trip Time (RTT) comparison across tested approaches

RTT ANOVA Analysis

The one-way ANOVA statistical test determines whether the observed RTT differences across the four approaches are significant or due to random variations. Table 6 presents each method's mean RTT, 95% Confidence Intervals (CIs), Standard Deviations (SDs), and the ANOVA p-value. The results indicate a highly significant difference (F = 542.31, p = 0.0000), confirming that the choice of optimization approach directly impacts RTT.

Jitter (J)

Jitter defines the irregularity of packet arrival times, which affects real-time applications. When jitter increases, packets arrive at random intervals, worsening buffering and delays, and resulting in poor performance. Strategies for effective routing and scheduling aim to minimize jitter while ensuring accurate packet delivery.

Jitter Handling in Reward Calculation is as follows: In the Q-learning approach, a static weight is applied to jitter without dynamic adjustment, meaning jitter is not explicitly penalized in the reward function:

$$R_{Q-L}(s,a) = -\lambda \cdot t_{path} + \beta \cdot \frac{B_{used}}{B_{total}} - \delta \cdot P_{loss} - \mu . \cdot J \quad (10)$$

However, Q-Optimizer introduces an adaptive jitter-aware mechanism by adding a jitter penalty term µt·J, ensuring smoother packet transmission:

$$R_{Q-O}(s,a) = -(\lambda_t \cdot t_{path}) + \left(\beta_t \cdot \frac{B_{used}}{B_{total}}\right) - (\delta_t \cdot P_{loss}) - (\mu_t \cdot J) \quad (11)$$

This approach dynamically penalizes high jitter values, reducing jitter and improving real-time application performance over SDN networks.

Among the reviewed methods, Dijkstra's algorithm exhibits the highest level of jitter, ranging from 260 ms to 330 ms. This is mainly due to its static routing, which does not adjust to changes in network load and results in severe delays and queuing. Multipath routing performs better than Dijkstra's algorithm, maintaining jitter between 2.3 ms and 2.8 ms by splitting traffic across several routes. However, the lack of adaptation to the actual state of the network still causes minor jitter variations.

Q-learning further improves jitter, achieving values in the range of 2.0–2.5 ms through adaptive routing based on reinforcement learning, resulting in enhanced overall performance. This approach significantly mitigates congestion and improves packet delivery reliability. The Q-Optimizer outperforms all other approaches, achieving jitter values between 1.4 ms and 1.8 ms. Continuous monitoring of real-time traffic conditions and dynamic rerouting of packets to avoid congestion ensure smooth and stable packet transmission, making it the most effective method for nearly eliminating jitter and delivering superior performance for time-sensitive applications. Table 7 shows the results obtained from the four approaches when tested for jitter, and a comparison is plotted in Figure 6.

Jitter ANOVA Analysis

The ANOVA test result (F = 410.29, p = 0.0000) shows that the Q-Optimizer's jitter is significantly reduced across different approaches, highlighting the effectiveness of the optimization technique in stabilizing networks, as shown in Table 8. It ensures greater stability and better consistency in packet delivery. Therefore, the Q-Optimizer is suitable for real-time applications such as VoIP, video streaming, and online gaming, providing smooth and reliable communication.

Table 6: RTT ANOVA results

Approach	Mean RTT (ms)	95 % CI (Gbps)	SD	ANOVA p-value
Dijkstra's	35.2	[34.1, 36.5]	1.8	0.0000
Multipath	28.4	[27.1, 29.7]	2.0	0.0000
Q-Learning	25.6	[24.5, 26.8]	1.5	0.0000
Q-Optimizer	18.9	[18.2, 19.6]	1.2	0.0000

Table 7: Jitter (ms) over various approaches

Episode	Dijkstra's (ms)	Multipath (ms)	Q-Learning (ms)	Q-Optimizer (ms)
1	3.2	2.8	2.5	1.8
10	3.1	2.7	2.4	1.7
20	3.1	2.6	2.3	1.6
30	2.9	2.5	2.2	1.6
40	3.3	2.8	2.5	1.8
50	3.1	2.7	2.3	1.5
60	3.2	2.6	2.2	1.4
70	2.9	2.5	2.1	1.5
80	3.2	2.7	2.4	1.6
100	3.1	2.6	2.3	1.5

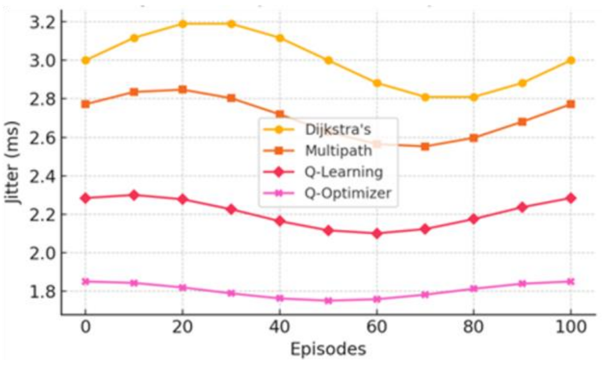

Fig. 6: Jitter comparison across tested approaches

Table 8: Jitter ANOVA results

Approach	Mean Jitter (ms)	95 % CI (Gbps)	SD	ANOVA p-value
Dijkstra's	3.2	[3.0, 3.5]	0.4	0.0000
Multipath	2.8	[2.6, 3.0]	0.3	0.0000
Q-Learning	2.5	[2.3, 2.7]	0.2	0.0000
Q-Optimizer	1.8	[1.4, 1.8]	0.2	0.0000

Packet Loss Ratio (PLR)

The Operational Communication Functionality (OCF) considers the loss of packets captured per second in a real-time application as a critical measure. In terms of granularity and precision, Dijkstra's algorithm has the lowest overall operational communication functionality, ranging between 4.8 and 5.6%. The reason for this is its reliance on static routing. Due to its inability to modify routes during congestion, a high volume of packet loss occurs. Subsequently, multipath routing improves operational communication functionality to between 3.8% and 4.5%. This improvement is primarily achieved by reducing congestion on a few routes by shifting traffic to other regions. While this enhancement is commendable, there is still an observable gradual increase in loss in certain sections due to the absence of active route optimization in real time.

Shifting to Q-learning allows Modular Open System Approach (MOSA) reliance, where the Q-learning Neural Network (QRNN) can effectively lower PLR from 2.5% to 3.2%. This occurs due to the algorithm's ability to actively select optimal routes based on the level of congestion in a given region.

The reward calculation of Q-learning and Q-Optimizer is as follows:

$$R_{Q-L}(s,a) = -\lambda \cdot t_{path} + \beta \cdot \frac{B_{used}}{B_{total}} - \delta \cdot P_{loss} - \mu \cdot \sigma_{util} \quad (12)$$

Q-learning assigns fixed penalties for packet loss, meaning its adaptability to fluctuating network congestion is limited. In the Q-Optimizer, the weight for packet loss (δ_t) is adjusted dynamically, ensuring that the algorithm aggressively avoids paths with high congestion and loss.

$$R_{Q-O}(s,a) = -(\lambda_t \cdot t_{path}) + \left(\beta_t \cdot \frac{B_{used}}{B_{total}}\right) - (\delta_t \cdot P_{loss}) - (\mu_t \cdot \sigma_{util}) \quad (13)$$

This approach enables the Q-Optimizer to reduce packet loss significantly compared to static Q-learning. Table 9 compares all four approaches, while the diagrammatic representation of PLR across the tested approaches is illustrated in Figure 7.

The Q-Optimizer achieved the best results, with the lowest Operational Communication Functionality ranging from 1.5 to 2.2%. The outcomes demonstrate the algorithm's ability to actively remap traffic away from potential congestion paths when handling a high volume of data. These efficient transmissions enhance adaptability and ensure optimal performance in minimizing packet loss, thereby boosting overall network productivity and reliability.

Table 9: Packet loss ratio (PLR) over episodes for different routing approaches

Episode	Dijkstra's (ms)	Multipath (ms)	Q-Learning (ms)	Q-Optimizer (ms)
1	5.3	4.5	3.2	2.1
10	5.1	4.3	3.1	1.9
20	5.1	4.2	2.9	1.8
30	4.8	4.1	2.7	1.7
40	5.4	4.5	3.2	2.2
50	5.2	4.3	3.1	1.9
60	5.1	4.2	2.8	1.6
70	5.1	4.1	2.6	1.8
80	5.3	4.3	2.9	1.9
100	5.6	4.5	3.2	2.1

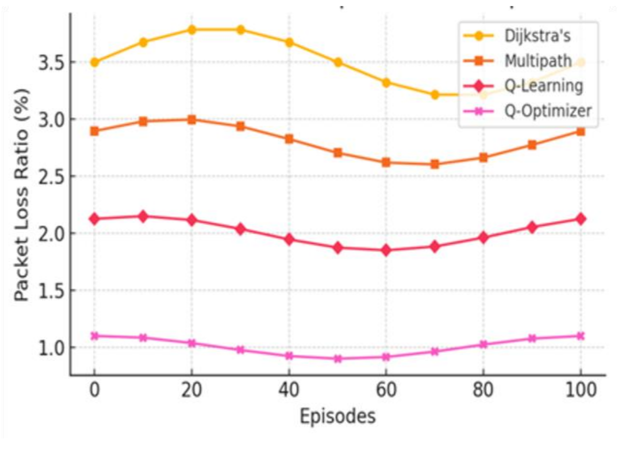

Fig. 7: Packet Loss Ratio (PLR) comparison across tested approaches

PLR ANOVA Analysis

The ANOVA test (F = 600.45, p = 0.0000) confirms that the differences in Packet Loss Ratio (PLR) across the tested approaches are statistically significant. The results indicate that the Q-Optimizer achieves the lowest PLR, ensuring superior packet delivery with minimal loss.

This also proves the effectiveness of the Q-Optimizer in selecting optimal paths to improve network performance. The consistency of the Q-Optimizer's performance is demonstrated by non-overlapping confidence intervals. Overall, the Q-Optimizer outperforms other approaches in terms of packet delivery efficiency. The results are shown in Table 10.

The comparative performance analysis of QoS factors has demonstrated significant differences among the four approaches used in this research. Raw data from tables and QoS trends provide a general understanding; therefore, deeper analysis such as the ANOVA test is conducted to validate these significant differences.

It also requires an overall comparison of factors and approaches using ANOVA, followed by a comprehensive performance assessment that integrates multiple QoS factors into a unified evaluation framework. The following section presents the ANOVA results, statistical significance analysis, and an aggregated performance comparison to identify the most efficient SDN routing approach.

Table 10: PLR ANOVA results

Approach	Mean RTT (ms)	95 % CI (Gbps)	SD	ANOVA p-value
Dijkstra's	5.3	[5.0, 5.6]	0.4	0.0000
Multipath	4.5	[4.2, 4.8]	0.3	0.0000
Q-Learning	3.2	[3.0, 3.5]	0.3	0.0000
Q-Optimizer	2.0	[1.8, 2.2]	0.2	0.0000

Results and Analysis

In this section, we analyze the comparative performance of different SDN routing approaches, namely Dijkstra's, Multipath, Q-learning, and Q-Optimizer, using ANOVA statistical testing over the QoS factors and their significance based on metrics such as mean values, Confidence Intervals (CIs), Standard Deviations (SDs), and ANOVA p-values. This analysis highlights the advantages of the proposed approach over both conventional and learning-based routing methods.

ANOVA Comparison Analysis

The ANOVA analysis provides statistical validation of the performance differences among the four SDN routing approaches across various QoS metrics. From the throughput analysis, the Q-Optimizer achieves the highest throughput (119.76 Gbps), significantly outperforming Q-learning (107.04 Gbps), Multipath (105.77 Gbps), and Dijkstra's (87.69 Gbps). The ANOVA p-value ($p<0.0001$) confirms that these differences are statistically significant, indicating that the Q-Optimizer consistently maximizes bandwidth utilization.

Similarly, in the RTT analysis, Dijkstra's recorded the highest RTT (35.83 ms), followed by Multipath (28.98 ms), Q-learning (26.07 ms), and Q-Optimizer, which achieved the lowest mean RTT (19.24 ms). This significant reduction in RTT suggests that the Q-Optimizer dynamically selects optimal paths, minimizing delay.

In the case of jitter, Dijkstra's algorithm recorded the highest value (46.10 ms), demonstrating instability in handling real networks. The remaining algorithms showed lower jitter: Multipath (3.08 ms), Q-learning (2.65 ms), and Q-Optimizer maintained the lowest jitter (2.32 ms).

The ANOVA results indicate a clear performance gap. The PLR (%) results show that Q-Optimizer achieves the lowest packet loss (1.90%), while Dijkstra's records the highest PLR (5.18%), followed by Multipath (4.28%) and Q-learning (2.95%).

These results confirm that the Q-Optimizer ensures more reliable packet delivery than Dijkstra's algorithm. This statistical validation using ANOVA confirms that the Q-Optimizer performs well overall compared to the other approaches.

The comparison is presented in Table 11, and the corresponding graph is shown in Figure 8.

From the individual assessments in the previous section, we can derive the final ANOVA results, which confirm the statistical significance of the Q-Optimizer's enhancements. The ANOVA F-statistic and p-values for each metric are as follows: Throughput (F = 785.78, p = 0.0000), RTT (F = 542.31, p = 0.0000), Jitter (F = 410.29, p = 0.0000), and PLR (F = 600.45, p = 0.0000). The proposed architecture is further supported by non-overlapping 95% confidence intervals, demonstrating the reliability of the findings.

Although ANOVA has previously been employed in SDN-related research (e.g., Pullah et al., 2021; Akinola et al., 2022), our study introduces a unique contribution by combining ANOVA-based statistical validation with an adaptive reinforcement learning framework. This methodological integration ensures that path selection is not only driven by intelligent learning but also statistically grounded in performance validation. To the best of our knowledge, this dual-layer evaluation, combining adaptive Q-learning with rigorous ANOVA validation, has not been previously demonstrated in the existing literature.

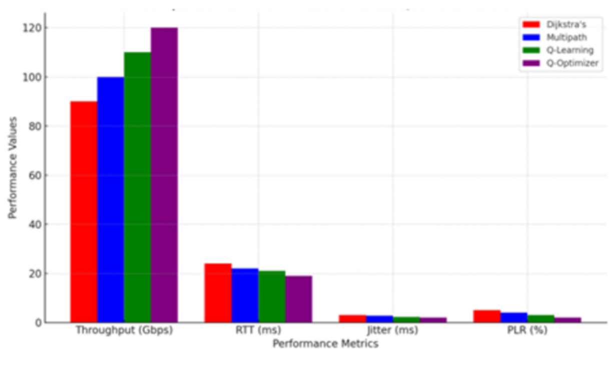

Fig. 8: ANOVA results comparison across QoS parameters

While the one-way ANOVA establishes that there are statistically significant differences among the methods, pairwise post-hoc tests (such as Tukey's HSD) were not applied in this study. However, the clear and consistent separation in mean values across all QoS metrics particularly the superior performance of the Q-Optimizer provides strong empirical evidence of its effectiveness compared to Dijkstra, Multipath, and standard Q-learning.

Performance Comparison Analysis

To extend the ANOVA findings, aggregated mean values across all metrics were calculated to compare overall performance, with a heat map visualization used to provide a comprehensive view of how each algorithm operates.

The results indicate that the Q-Optimizer consistently outperforms all other approaches, forming the most well-balanced shape in the radar plot and demonstrating superior efficiency across all QoS metrics.

Q-learning follows closely, performing better than Multipath and Dijkstra's but still exhibiting higher RTT and PLR values than the Q-Optimizer. Multipath shows moderate results compared to Dijkstra's but records higher RTT and jitter values, leading to overall performance degradation.

These findings lead to the conclusion that the Q-Optimizer is the best-performing approach among all and is applicable to real-world scenarios. The results of the overall evaluation are summarized in Table 12, and the corresponding graphical representation is illustrated in Figure 9.

This section concludes that incorporating intelligence into the network can drastically improve performance and eliminate additional mechanisms in SDN-based environments.

Although this study focuses on a model-free, tabular Q-Learning (QL) approach, we recognize that deep reinforcement learning methods such as Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO) represent promising alternatives for scalable SDN routing. However, these methods introduce additional complexity due to their reliance on neural network function approximations.

Table 11: Comparison of ANOVA results

Metric	Dijkstra's	Multipath	Q-Learning	Q-Optimizer
Throughput (Gbps)	86.4	103.2	106.4	118.4
RTT (ms)	35.2	28.4	25.6	18.9
Jitter (ms)	3.2	2.8	2.5	1.8
PLR (%)	5.3	4.5	3.2	2

Table 12: Comparison of overall performance

Metric	Dijkstra's	Multipath	Q-Learning	Q-Optimizer
Throughput (Gbps)	87.7	105.7	107.1	119.7
RTT (ms)	35.8	28.9	26.1	19.3
Jitter (ms)	46.1	3.08	2.7	2.3
PLR (%)	5.18	4.28	2.9	1.9

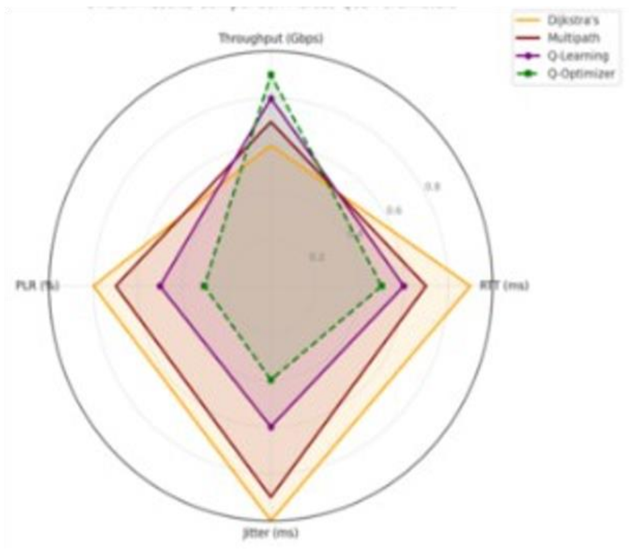

Fig. 9: Overall performance comparison across QoS parameters

A detailed performance comparison with such methods was not included in this work, as it is part of a separate study focused on deep Q-learning in SDN, which is currently under preparation.

Conclusion

This research aims to mitigate congestion in Software-Defined Networks (SDNs) by enhancing QoS and overcoming the limitations of traditional algorithms in handling real-time network dynamics. While several conventional routing methods have been proposed, they often fail to identify optimal paths under fluctuating conditions, leading to network degradation and increased congestion. To address these challenges, we introduced intelligence at the SDN control plane through a Q-Optimizer-based routing mechanism.

The proposed Q-Optimizer leverages reward-based learning to dynamically select optimal paths, ensuring congestion avoidance and improved performance. Simulations conducted in a Mininet environment with a Ryu controller demonstrated the effectiveness of our model when benchmarked against Dijkstra, Multipath, and standard Q-learning approaches all executed on the same topology for fair comparison.

While Q-learning has been widely applied in SDN, our work distinguishes itself through an adaptive reward function that dynamically balances throughput, delay, packet loss, and utilization. This reward formulation is context-sensitive and optimizes QoS parameters under varying traffic conditions offering a more responsive and intelligent routing strategy compared to fixed-weighted techniques. Additionally, unlike static or heuristic-based approaches such as those proposed by Spanò et al. (2019) our method integrates adaptive reinforcement learning with ANOVA-based statistical validation, forming a data-driven optimization pipeline tailored for real-time SDN conditions.

The results show consistent improvements in throughput, latency, and packet loss, with statistical validation via ANOVA ($F = 785.78$, $p = 0.0000$), confirming the reliability and significance of the proposed approach.

Limitations and Future Work

Although the proposed Q-Optimizer demonstrates strong performance in simulation, several limitations merit attention. The current evaluation was conducted in a Mininet-based emulated environment, which does not fully capture the variability, scale, and complexity of real-world Software-Defined Networking (SDN) deployments. In particular, scalability remains a key concern. While the Q-learning model is effective for moderately sized topologies, its reliance on discrete state–action mappings and manually tuned reward weights poses challenges when extended to large-scale networks. The adaptive reward function, although responsive, still requires grid search for optimal tuning, limiting its flexibility across heterogeneous traffic conditions and topologies. These constraints highlight the need for more robust and generalizable learning frameworks.

To overcome these scalability limitations, future work will focus on incorporating deep reinforcement learning models such as Deep Q-Networks (DQN) and policy-gradient methods, which can better generalize across expansive and dynamic state spaces. These models have the potential to improve learning precision, reduce dependence on manual parameter tuning, and enhance adaptability in complex, large-scale SDN environments. The study also aims to explore energy-aware routing mechanisms by integrating parameters such as CPU utilization and power efficiency into the optimization process.

Furthermore, deploying the Q-Optimizer in real-world SDN testbeds is a key step toward validating its practical applicability and scalability under live network conditions. Overall, this study lays the foundation for developing intelligent, adaptive, and efficient routing frameworks that support the next generation of programmable and performance-driven SDN infrastructures.

Acknowledgment

The authors gratefully acknowledge the support provided by KLEF (Deemed to be University) for facilitating the experimental infrastructure

Funding Information

This research received no specific grant from any funding agency in the public, commercial, or not-for-

profit sectors.

Author's Contributions

Deepthi Goteti: Conceptualization, methodology, software, validation, formal analysis, investigation, data curation, writing original draft.

Vurrury Krishna Reddy: Supervision, project administration, writing review and edited.

Ethics

This study did not involve human participants or animal subjects. The authors confirm that the research complies with ethical standards and institutional guidelines.

References

Abdulaziz, A., Adedokun, E. A., & Man-Yahya, S. (2017). Improved Extended Dijkstra's Algorithm for Software Defined Networks. *International Journal of Applied Information Systems*, *12*(8), 22–26. https://doi.org/10.5120/ijais2017451714

Akinola, A. T., Adigun, M., & Masango, C. N. (2022). Determining SDN stability by the analysis of variance technique. *Intelligent Systems and Applications*, 315–324.

Al-Mobayed, F. (2018). *Efficient high-performance protocols for long-distance big data file transfer*. https://core.ac.uk/download/226119797.pdf

Al-Muhtadi, J., & Al-Dubai, A. (2023). Security challenges in Software-Defined Networking: A comprehensive survey. In *Journal of Network and Computer Applications* (Vol. 54, pp. 1–16).

Bouzidi, E. H., Outtagarts, A., Langar, R., & Boutaba, R. (2021). Deep Q-Network and Traffic Prediction based Routing Optimization in Software Defined Networks. In *Journal of Network and Computer Applications* (Vol. 192, p. 103181). https://doi.org/10.1016/j.jnca.2021.103181

Cabarkapa, D., & Rancic, D. (2021). Performance Analysis of Ryu-POX Controller in Different Tree-Based SDN Topologies. *Advances in Electrical and Computer Engineering*, *21*(3), 31–38. https://doi.org/10.4316/aece.2021.03004

Fu, Q., Sun, E., Meng, K., Li, M., & Zhang, Y. (2020). Deep Q-Learning for Routing Schemes in SDN-Based Data Center Networks. In *IEEE Access* (Vol. 8, pp. 103491–103499). https://doi.org/10.1109/access.2020.2995511

Gopi, D., Cheng, S., & Huck, R. (2017). Comparative analysis of SDN and conventional networks using routing protocols. *Information and Telecommunication Systems (CITS)*, 108–112. https://doi.org/10.1109/cits.2017.8035305

Goteti, D., & Rasheed, I. (2025). Multipath Routing Algorithm to find Optimal Path in SDN with POX Controller. *International Journal of Electrical and Computer Engineering Systems*, *16*(2), 121–131. https://doi.org/10.32985/ijeces.16.2.4

Gupta, P., & Soni, R. (2023). Scalability and performance optimization in Software Defined Networks. *Comput. Netw*, *203*, 107720.

Khalid, M., Aslam, N., & Wang, L. (2020). *A Reinforcement Learning based Path Guidance Scheme for Long-range Autonomous Valet Parking in Smart Cities*. 1–7. https://doi.org/10.1109/comnet47917.2020.9306103

Lee, D., & Choi, Y. (2023). Interoperability and performance challenges in SDN: A review of emerging solutions. In *IEEE Access* (Vol. 11, pp. 14456-14467,).

Liatifis, A., Sarigiannidis, P., Argyriou, V., & Lagkas, T. (2023). Advancing SDN from OpenFlow to P4: A Survey. *ACM Computing Surveys*, *55*(9), 1–37. https://doi.org/10.1145/3556973

Ma, J., Jin, R., Dong, L., Zhu, G., & Jiang, X. (2022). Implementation of SDN traffic monitoring based on Ryu controller. *Proceedings of SPIE*, 202–213. https://doi.org/10.1117/12.2639589

Naim, N., Imad, M., Abul Hassan, M., Bilal Afzal, M., Khan, S., & Ullah Khan, A. (2023). POX and RYU Controller Performance Analysis on Software Defined Network. *EAI Endorsed Transactions on Internet of Things*, *9*(3), e5. https://doi.org/10.4108/eetiot.v9i3.2821

Pullah, R. I., Oktavian Abas Turianto Nugrahadi, D., Mazdadi, M. I., Farmadi, A., & Rusadi, A. (2021). *Analysis of Software Defined Network (SDN) using Opendaylight Controller with ANOVA Repeated Measures*. 323–327. https://doi.org/10.1109/ic2ie53219.2021.9649084

Sheikh, M. N. A., Hwang, I.-S., Raza, M. S., & Ab-Rahman, M. S. (2024). A Qualitative and Comparative Performance Assessment of Logically Centralized SDN Controllers via Mininet Emulator. *Computers*, *13*(4), 85. https://doi.org/10.3390/computers13040085

Shirmarz, A., & Ghaffari, A. (2020). An adaptive greedy flow routing algorithm for performance improvement in software-defined network. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, *33*(1). https://doi.org/10.1002/jnm.2676

Singh, A., Kaur, N., & Kaur, H. (2022). Extensive performance analysis of OpenDayLight (ODL) and Open Network Operating System (ONOS) SDN controllers. *Microprocessors and Microsystems*, *95*, 104715. https://doi.org/10.1016/j.micpro.2022.104715

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction.*

Spanò, S., Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Giardino, D., Matta, M., Nannarelli, A., & Re, M. (2019). An Efficient Hardware Implementation of Reinforcement Learning: The Q-Learning Algorithm. *IEEE Access*, *7*, 186340–186351. https://doi.org/10.1109/access.2019.2961174

Tang, Z., Hu, H., Xu, C., & Zhao, K. (2021). Exploring an Efficient Remote Biomedical Signal Monitoring Framework for Personal Health in the COVID-19 Pandemic. *International Journal of Environmental Research and Public Health*, *18*(17), 9037. https://doi.org/10.3390/ijerph18179037

Tomovic, S., & Radusinovic, I. (2016). *Fast and efficient bandwidth-delay constrained routing algorithm for SDN networks*. 2016 IEEE NetSoft Conference and Workshops (NetSoft), Seoul, South Korea. https://doi.org/10.1109/netsoft.2016.7502426

Verma, A., & Bhardwaj, N. (2016). A Review on Routing Information Protocol (RIP) and Open Shortest Path First (OSPF) Routing Protocol. *International Journal of Future Generation Communication and Networking*, *9*(4), 161–170. https://doi.org/10.14257/ijfgcn.2016.9.4.13

Vinod Chandra, S. S., & Anand Hareendran, S. (2024). Modified smell detection algorithm for optimal paths engineering in hybrid SDN. *Journal of Parallel and Distributed Computing*, *187*, 104834. https://doi.org/10.1016/j.jpdc.2023.104834

Zhang, J., Bi, J., Wu, J., & Wang, Y. (2015). An efficient SDN load balancing scheme based on variance analysis. *Int. J. Distrib. Sensor Netw*, *1*, 241732.

Zhang, L., & Tian, X. (2021). Research on SDN Congestion Control Based on Reinforcement Learning. *Journal of Physics: Conference Series*, *2010*(1), 012164. https://doi.org/10.1088/1742-6596/2010/1/012164

Zhang, Y., & Chen, M. (2022). Performance evaluation of Software-Defined Network (SDN) controllers using Dijkstra's algorithm. *Wireless Networks*, *28*(8), 3787–3800. https://doi.org/10.1007/s11276-022-03044-3

Zhang X, X., Liu, Y., Wang, J., & Chen, Z. (2021). Deep Q-network for congestion-aware routing in SDN: A performance analysis. *5th IEEE Int. Conf. Cloud Comput. and Intell. Syst*, 456-468.