Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.058480

ARTICLE

Effective Controller Placement in Software-Defined Internet-of-Things Leveraging Deep Q-Learning (DQL)

Jehad Ali[1,*] **and Mohammed J. F. Alenazi**[2]

[1]Department of AI Convergence Network, Ajou University, Suwon, 16499, Republic of Korea

[2]Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Riyadh, 11362, Saudi Arabia

*Corresponding Author: Jehad Ali. Email: jehadali@ajou.ac.kr

Received: 13 September 2024 Accepted: 04 November 2024 Published: 19 December 2024

ABSTRACT

The controller is a main component in the Software-Defined Networking (SDN) framework, which plays a significant role in enabling programmability and orchestration for 5G and next-generation networks. In SDN, frequent communication occurs between network switches and the controller, which manages and directs traffic flows. If the controller is not strategically placed within the network, this communication can experience increased delays, negatively affecting network performance. Specifically, an improperly placed controller can lead to higher end-to-end (E2E) delay, as switches must traverse more hops or encounter greater propagation delays when communicating with the controller. This paper introduces a novel approach using Deep Q-Learning (DQL) to dynamically place controllers in Software-Defined Internet of Things (SD-IoT) environments, with the goal of minimizing E2E delay between switches and controllers. E2E delay, a crucial metric for network performance, is influenced by two key factors: hop count, which measures the number of network nodes data must traverse, and propagation delay, which accounts for the physical distance between nodes. Our approach models the controller placement problem as a Markov Decision Process (MDP). In this model, the network configuration at any given time is represented as a "state," while "actions" correspond to potential decisions regarding the placement of controllers or the reassignment of switches to controllers. Using a Deep Q-Network (DQN) to approximate the Q-function, the system learns the optimal controller placement by maximizing the cumulative reward, which is defined as the negative of the E2E delay. Essentially, the lower the delay, the higher the reward the system receives, enabling it to continuously improve its controller placement strategy. The experimental results show that our DQL-based method significantly reduces E2E delay when compared to traditional benchmark placement strategies. By dynamically learning from the network's real-time conditions, the proposed method ensures that controller placement remains efficient and responsive, reducing communication delays and enhancing overall network performance.

KEYWORDS

Software-defined networking; deep Q-learning; controller placement; quality of service

1 Introduction

Software-defined networking (SDN) offers a centralized and programmable networking architecture. As compared to the traditional distributed approach adopted by legacy networking, the SDN

separates the control plane from the data plane. The core of this architecture is the SDN controller, which provides programmability features that allow policies to be deployed on the underlying network devices. This centralization offers several benefits, including a global view of the network, centralized management, global statistics collection, and policy enforcement on network devices. These advantages make SDN an ideal solution for 5G and beyond networks, the Internet of Things (IoT), vehicular *ad hoc* networks, and networking solutions integrated with artificial intelligence.

In SDN, the controller plays an intelligent role in directing packets to their destinations via various routes. The switches contain rules, known as flow rules, that determine how packets are forwarded from their source to their destination. When a packet arrives at a switch's incoming port, it is forwarded according to these flow rules stored in flow tables. However, if a packet arrives at a switch and no matching flow rule exists in the flow table, the packet is sent to the controller to determine its destination. The controller uses the address resolution protocol (ARP) to find the destination and then updates the switch's flow table with the appropriate flow rule. This communication between the controller and the switches is crucial; if the controller is not optimally placed within the SDN, it can increase the end-to-end delay (E2E) in the communication process [1–3].

Due to proliferation of devices connected to the internet, the IoT devices generate large volumes of traffic. Moreover, the IoT devices are continuing its growth. The IoT analytics [4] forecasted 30 billion devices by 2027. Hence, in SD-IoT when new packets arrive at switches without corresponding flow rules that specify the path from source to destination, the controller must process these packets to find the destination route. As more packets are sent to the controller, it needs to handle an increasing number of requests, leading to higher E2E delays in SD-IoT networks. Therefore, placing the controller in optimal locations is crucial to reducing E2E delay in SD-IoT environments.

In this paper, we present a controller placement solution for SD-IoT employing Deep Q-learning (DQL) considering the hop count and propagation delay as metrics for placement of the controller at appropriate optimal locations to reduce the E2E delay. By applying DQL to controller placement in SDN, it allows for intelligent and adaptive optimization, significantly reducing E2E delay by considering both hop count and propagation delay. This approach ensures that the SDN can dynamically and efficiently respond to changes, maintaining optimal performance and minimizing latency in the network.

The main contributions of our paper are as follows:

1. First, we formulate the problem of controller placement in SDN with DQL.
2. We provide a DQL-based controller placement solution in SDN considering the hop count and propagation delay.
3. The proposed solution is verified through experimental evaluation and compared with other benchmark solutions.
4. We evaluate the proposed framework in various real-Internet topologies and demonstrate its comparison with state-of-the-art (SOTA) methods for controller placement.
5. We evaluate the proposed method by obtaining the results from emulated environment and also perform simulations to show the efficacy of the suggested scheme.

The rest of the article is organized as follows. In Section 2, we discuss the related works. In Section 3, we formulate the problem of controller placement leveraging DQL. In Section 4, we provide the details of our suggested DQL solution and suggest an algorithm to explain it with step-by-step process. In Section 5, the experimental analysis and results are discussed. Finally, Section 6 concludes the paper.

2 Related Work

Controller placement strategies in SDN aim to optimize the network by focusing on various performance metrics such as cost, reliability, delay, load balancing, and energy efficiency. These metrics help determine where controllers should be positioned within the network to ensure efficient communication between switches and controllers. However, addressing all these metrics simultaneously is challenging, as optimizing for one may negatively impact others.

For example, if a controller placement strategy focuses solely on minimizing delay, it may suggest placing controllers in locations that are centrally located relative to the network switches. While this reduces the hop count and propagation delay, it might not account for cost considerations, such as the expense of setting up and maintaining controllers in those central locations. Similarly, a placement strategy that prioritizes cost efficiency by placing controllers in regions with lower operational expenses can lead to higher communication delays if those locations are further from the network's busiest switches.

Additionally, optimizing for energy efficiency often involves placing controllers in locations that minimize energy consumption, which may conflict with delay minimization strategies. For instance, a controller placed in a remote, energy-efficient data center may result in higher delays due to increased physical distance from the switches it manages. In the same way, optimizing for load balancing—to distribute the network traffic evenly among controllers—may require placing controllers in areas that increase delay for certain switches or increase the overall network cost.

Hence, most controller placement strategies in SDN do not attempt to optimize all of these metrics simultaneously, as achieving an optimal balance between them is highly complex. Instead, researchers tend to focus on specific subsets of these metrics based on the particular goals and constraints of the network in question. For instance, a placement strategy for a real-time, latency-sensitive application might prioritize delay reduction, while a strategy for a large, distributed IoT network might focus on cost and energy efficiency.

In essence, the trade-offs between these metrics make it difficult to develop a one-size-fits-all controller placement strategy. This is why existing approaches typically optimize for certain metrics, while accepting that others may not be fully addressed. Researchers continue to explore multi-objective optimization techniques that can better balance these trade-offs, but finding a placement strategy that satisfies all metrics equally remains a significant challenge in SDN.

The placement of the controller in SDN is a challenging issue that has garnered considerable interest from researchers and network engineers [5–7]. The latency between switches and controllers is a critical factor in determining optimal controller placement. High delays can impair the controllers' ability to respond to network events promptly. Some studies [8,9] have examined propagation delays in the context of assigning switches to SDN controllers.

Some studies considered the controller placement with K-center, K-means [10,11], and modified versions with novelty in the K-means. The K-center and K-center based approaches initialize the centers randomly and iteratively assign switches to new centers until the clusters stabilize. However, this method does not ensure minimal propagation delay. For instance, as noted in [12], the delay with a new center can be higher than that of the previous cluster. Similarly, the controller placement problem in SD-IoT is illustrated in [13], by using a sub-modularity strategy leveraging a heuristic technique. However, the study did not evaluate the E2E latency. Further, the work lacks demonstration in real Internet topologies.

Moreover, some studies [14,15] consider multiple objective methods to place the controller in SDN. Authors in [14] use a multi-criteria analysis method analytic hierarchy process to evaluate the

placement of controller using several criteria such as hop count, delay, and load metric. Similarly, the study in [15] uses analytical network process methodology to place the controller in SD-IoT clusters at proper locations with the aim of minimizing the E2E delay.

The suggested studies don't dynamically adjust the placement of the controller in SD-IoT according to the network conditions to minimize the E2E delay. The suggested approach leveraging DQL provides a scalable, adaptive solution for SDN environments, ensuring improved network performance and reliability. The implications of this research extend to the broader field of network management, offering a robust framework for intelligent decision-making in complex, dynamic network scenarios.

3 Problem Formulation and Proposed Methodology

In SDN, the optimal placement of controllers is crucial for minimizing communication delays between switches and controllers. The challenge is to determine the best locations for controllers within the network topology, aiming to reduce overall delay by taking into account two key factors: hop count (the number of nodes a packet must pass through) and propagation delay (the time it takes for data to travel between nodes).

For example, imagine a network with switches spread out across multiple geographical locations. If the controller is placed too far from the majority of switches, the data traveling between switches and the controller will have to traverse many intermediate nodes, increasing both hop count and propagation delay. This leads to higher E2E delay, which can degrade the network's performance, especially in time-sensitive applications like real-time video streaming or IoT device communication.

The problem of optimal controller placement can be effectively formulated as a reinforcement learning problem. In this context, DQL is used to optimize the placement strategy. DQL is well-suited for this task because it allows the system to dynamically learn and improve based on the network's real-time conditions, making it highly adaptable to changing network topologies and traffic patterns.

Consider a network with 10 switches arranged in a complex mesh topology, where the distances (and delays) between switches vary due to physical distance and network conditions. Placing one controller in this topology involves choosing a location that minimizes the total delay between the switches and the controller. If the controller is placed too close to one cluster of switches, the switches farther away will suffer from high delays due to increased hop count and propagation delay. On the other hand, placing the controller at a more central location can help balance the communication delay across all switches.

In our approach, the DQL agent acts as the decision-maker, or "agent," which learns to place the controllers at the optimal locations within the network. The network's configuration, including the current positions of the switches and the delay metrics (hop count and propagation delay), forms the "state" of the environment. The agent's "actions" involve placing or adjusting the position of controllers in the network.

The goal of the DQL agent is to maximize a "reward," which, in this case, is the negative of the total E2E delay (a lower delay yields a higher reward). Over time, the agent learns from its decisions by simulating different controller placements and observing how they impact the delay. For example, placing a controller closer to a densely connected group of switches may initially seem beneficial, but the agent might realize that this leads to increased delays for switches on the network's periphery. By continually learning from such scenarios, the agent becomes adept at balancing hop count and propagation delay, arriving at an optimal solution.

The DQL-based approach excels in dynamic environments where network conditions or topology can change over time. For instance, in a scenario where new switches are added to the network or traffic

patterns shift, the DQL agent can quickly adjust its placement strategy to accommodate these changes, ensuring that the communication delays remain minimized. This dynamic optimization is one of the key advantages of using reinforcement learning techniques like DQL, as it allows the system to adapt in real-time, unlike traditional static placement methods. By framing the controller placement problem as a DQL task, the system intelligently learns to optimize controller locations within the network, minimizing overall communication delay. This results in a more efficient SDN, with reduced hop counts and propagation delays, leading to enhanced performance across various network applications.

Fig. 1 provides a visual representation of how SDN controllers are strategically placed to minimize delay in a network of switches. In this scenario, the key goal is to attach controllers to the most critical switches in the network, ensuring that the communication delay between switches and controllers is kept as low as possible. In the left part of Fig. 1, we see a simple SDN topology with multiple switches, each responsible for routing data through the network. A single controller is placed within this network, responsible for managing the data flow and communication between the switches. The controller's placement is crucial because if it is placed too far from the majority of switches, the data packets must travel a longer path, increasing both the hop count and propagation delay. Ideally, the controller should be positioned centrally or close to the most active switches to minimize this delay.

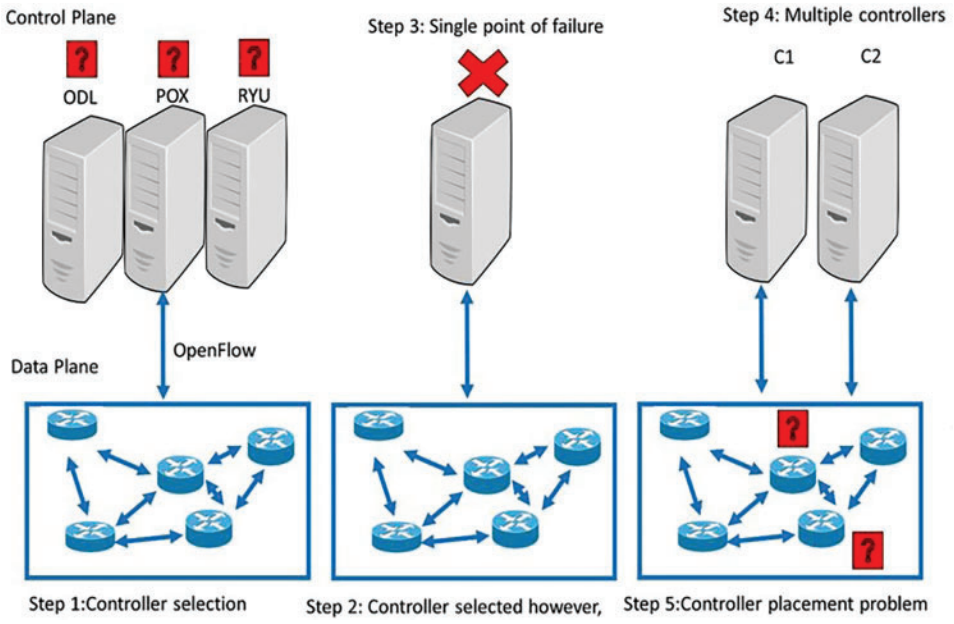

Figure 1: Problem description of controller placement in SD-IoT

However, there is a significant risk involved with relying on a single controller, as shown in the figure. If this controller fails—due to either software or hardware failure—the entire network can shut down, since the switches will lose their connection to the controller and be unable to route data efficiently. This single point of failure highlights the importance of having redundancy in controller placement to ensure network reliability. To address this issue, the right side of Fig. 1 shows an improved design where two controllers, labeled c1 and c2, are placed in the network to enhance both reliability and performance. By distributing the control responsibilities across multiple controllers, the risk of total network failure due to a single controller failure is mitigated. If one controller fails, the other can take over, ensuring continuous network operation. However, simply adding more controllers is not enough. The placement of these controllers—c1 and c2—is critical to ensuring minimal delay. The

figure suggests that both controllers should be placed in strategic locations that minimize the distance between them and the switches they manage. For instance, placing c1 near a cluster of frequently communicating switches and c2 near another active cluster ensures that the hop count and propagation delay are minimized. If c1 and c2 are placed too far from the switches they control, or too close to each other while leaving parts of the network distant, the delay might still be high, defeating the purpose of having multiple controllers. Consider a network with 10 switches spread out over a geographical area. If a single controller is placed at one edge of the network, switches on the opposite side will experience higher delays because they must communicate over a larger number of hops. Now, consider the scenario in which two controllers, c1 and c2, are placed in optimal central positions within their respective clusters of switches. By doing so, each controller can manage a portion of the switches more effectively, reducing the overall distance that data packets must travel and, consequently, lowering the communication delay. If c1 fails, c2 can take over, ensuring that the network continues to function without significant disruption. Thus, Fig. 1 effectively illustrates the importance of both controller redundancy and strategic placement within SDN. Not only does this approach reduce the end-to-end delay, but it also enhances network reliability by minimizing the impact of potential controller failures.

The SDN can help implement DQL-based controller placement in resource-limited environments by decoupling the control and data planes, which centralizes decision-making. This allows the DQL agent to have a global view of the network state, optimizing controller placement dynamically. By using SDN's centralized control, computationally expensive DQL tasks can be handled at the controller level, while switches handle basic packet forwarding. Additionally, SDN's programmability allows for efficient reconfiguration of network paths and resources, minimizing the overhead and complexity in resource-constrained environments.

4 Proposed Deep Q-Learning-Based Method for Controller Placement

The DQL can be leveraged to reduce E2E delay between controllers and switches in SDN by optimizing controller placement, considering metrics such as hop count and propagation delay. The DQN is trained using experience replay and target networks to enhance learning stability and efficiency. The reward function is designed to integrate both hop count and propagation delay, ensuring comprehensive delay minimization. Our approach dynamically adapts to real-time network conditions, continuously optimizing controller placements to respond to traffic fluctuations and topology changes. The problem formulation and the detailed steps are discussed in the next subsections. Here's how it can be effectively applied.

4.1 Problem Description with DQL

We model the controller placement problem as a Markov Decision Process (MDP):

- **State:** Configuration of the network, i.e., the SDN network topologies such as Abilene, OS3E, etc., including controller locations, switch-to-controller assignments, hop counts, i.e., the number of routers or SDN switches, and propagation delays. The equation for the propagation delay is given in Eq. (2).
- **Action:** Possible changes to controller placements or switch-to-controller assignments.
- **Reward:** Negative of the end-to-end delay, which includes both hop count and propagation delay. The aim is to maximize cumulative reward, effectively minimizing delay. The Eq. (1) shows the reward with respect to the hop count and propagation delay.

4.2 DQL Network State Representation

Represent the network state with a feature vector encompassing:

- Current controller locations.
- Network topology details (e.g., hop counts and propagation delays between nodes).
- Current assignments of switches to controllers.
- Historical delay metrics and traffic load information.

4.3 DQL Algorithm Setup with Step-by-Step Explanation

- **Deep Q-Network (DQN):** Use a neural network to approximate the Q-function, estimating the expected cumulative reward (negative delay) for each action in a given state.
- **Experience Replay:** Store the agent's experiences (state, action, reward, next state) in a replay buffer. This allows the agent to learn from past experiences and stabilize training.
- **Target Network:** Maintain a separate target network for calculating target Q-values, updated periodically to match the main network's weights for stable learning.

The procedure to training the DQL agent involves the initialization of weights, exploration and exploitation of the environment or the SDN topology where the controller has to be placed. These steps are given as follows:

- **Initialization:** Randomly initialize the weights of the neural network.
- **Exploration *vs*. Exploitation:** Implement an ε-greedy policy to balance exploration (random actions) and exploitation (actions that maximize the Q-value). Start with a high ε (favoring exploration) and decrease it over time to focus on exploitation.
- **Action Execution:** In each step, select an action based on the current state and ε-greedy policy.
- **State Transition and Reward Calculation:** Execute the action, observe the new state, and measure the end-to-end delay (considering hop count and propagation delay) to compute the reward.
- **Experience Storage:** Store the transition (state, action, reward, next state) in the replay buffer.
- **Mini-batch Training:** Sample mini-batches from the replay buffer to train the DQN. Update the Q-network using gradient descent to minimize the loss between predicted and target Q-values.

4.3.1 Incorporating Hop Count and Propagation Delay

- **Reward Function:** Define the reward function to incorporate both hop count and propagation delay. For example:

$$Reward\,(R) = -(\alpha \times hop\,count + \beta \times Propagation\,delay + \gamma \times L) \tag{1}$$

- where α and β are weight parameters to balance the impact of hop count and propagation delay on the overall delay.
- α is the weight parameter that controls the impact of hop count,
- β is the weight parameter that controls the impact of propagation delay,
- γ is the weight parameter for traffic load L. In Eq. (1), the L is a penalty factor to show an increase in the traffic load.
- L shows the traffic between the switch and the controller.
- Hop Count is the number of hops (routers, nodes, etc.) a packet takes.
- Propagation delay refers to the time taken for a signal to travel from the sender to the receiver.
- The formula to calculate the propagation delay is as given below:

$$Propagation\,delay = d/c \tag{2}$$

where d is the distance between two nodes or routers in a given network in which we want to place the controller and c refers to the speed through which a signal or data transmits via a medium.

We want to optimize our goal, i.e., to minimize the overall delay by finding the optimal controller placement in SDN. Eq. (3) shows the objective function. The N denotes the number of switch-controller pairs.

$$Min \sum_{i=1}^{N} (\alpha \times hop\ count + \beta \times Propagation\ delay) \tag{3}$$

- **State Features:** Ensure that state representation includes detailed information on hop counts and propagation delays for accurate decision-making.

4.3.2 Deployment and Real-Time Adaptation

- **Deployment:** Deploy the trained DQL model to manage controller placements dynamically in the live network.
- **Continuous Monitoring:** Continuously monitor the network for changes in topology, traffic patterns, and delays. Use the DQL agent to adapt placements in real-time.
- **Periodic Retraining:** Retrain the DQL model periodically with updated data to adapt to evolving network conditions and traffic characteristics.

The detailed description of the DQL for **controller placement in SDN** is illustrated in the following steps:

1. **Learning Rate (α):** The parameter controls how much the Q-values are updated during learning. A smaller learning rate ensures more stable learning but may slow convergence. In our experiments we kept a moderate value of 0.01 as a starting point. We kept this value because it balances the trade-off between faster convergence and stable learning
2. **Discount Factor (γ):** It balances the importance of future rewards *vs*. immediate ones, with a typical value around 0.9 for controller placement in our experiments, ensuring the agent accounts for long-term network stability.
3. **Neural Network Architecture:** Our DQL architecture includes:
   - o **Input Layer:** which represents network states (e.g., hop count, propagation delay, and traffic load).
   - o **Hidden Layers:** which consists of multiple fully connected layers with ReLU activations help model complex network dynamics. The number of hidden layers is often 3 with 64–128 neurons to balance complexity and computational cost.
   - o **Output Layer:** which provides Q-values for each possible action (controller placements).
4. **Overfitting Avoidance:** To avoid overfitting, we follow the following strategy in our DQL.
   - o **Experience Replay:** Stores past experiences in memory to break correlation in training data, making the network more robust.
   - o **Target Network:** A secondary network is updated less frequently to stabilize learning and reduce fluctuations during training.
5. **Optimization Techniques:**
   - o **Batch Normalization** is used to prevent overfitting and improve generalization.
   - o The **Adam optimizer** accelerates convergence by adjusting the learning rate adaptively during training.

Our choices for the above-mentioned parameters aim to achieve an efficient and scalable DQL agent that optimizes controller placement, minimizes network delays, and handles traffic load dynamically.

The following algorithm illustrates the procedure of placement in details:

Algorithm 1: Controller placement in SDN using deep Q-learning (DQL)

Input:

- Network topology $G = (V, E)$, where V is the set of nodes (switches) and E is the set of edges (links).
- Number of controllers C.
- Maximum episodes N.
- Discount factor γ.
- Learning rate α.
- Weight parameters for reward function β_{hop}, β_{delay}.

Output:

Optimal controller placement minimizing E2E delay.

1. Initialization:
   - Initialize Q-network with random weights.
   - Initialize target Q-network Q_{target} with the same weights as the Q-network.
   - Set exploration rate epsilon ϵ to ϵ_{max}.
   - For each node v $\in$ V, initialize states representing the current topology and delay metrics.
   - Initialize replay memory D.
2. Define Reward Function:
   - For each placement action, define the reward as:
     $Reward = -(\beta \times hopcount + \beta \times Propagation\ delay + \gamma \times L)$.
   - β_{hop}, β_{delay} are weight parameters for balancing hop count and delay.
3. Training Process:

For each episode $e = 1, 2, \ldots, N$:

1. Reset Environment:
   Initialize the network configuration, where no controllers are placed.
2. For each time step t = 1, 2,...,T:
   a. Exploration *vs*. Exploitation:
      - With probability ϵ, select a random action a_t (controller placement at node v).
      - Otherwise, select the action a_t that maximizes the Q-value:
   b. Take Action and Observe Reward:
      - Place the controller at the chosen node v.
      - Calculate hop count and propagation delay for all switch-controller pairs.
      - Compute the reward using the defined reward function.
   c. Store Experience:
      - Store transition (S_t, a_t, r_t, S_{t+1}) in replay memory D.
   d. Sample from Replay Memory:
      - Randomly sample a mini-batch of experiences from D.
   e. Update Q-Network:
      - For each experience, calculate the target value:
      - Update the Q-network by minimizing the loss:
   f. Update Target Network:
      - Periodically update the target Q-network weights to match the Q-network.
   g. Reduce Exploration Rate:
      - Decrease ϵ towards ϵ_{min} using an exponential decay schedule.
4. Output Optimal Controller Placement:

After completing the training, the trained Q-network is used to determine the optimal placement of controllers, minimizing hop count and propagation delay.

End

5 Experimental Setup, Results, Discussion and Comparison with Benchmark Method

The results are analyzed in various networks such as Interoute, OS3E, USNet and Abilene leveraging MATLAB and Mininet tools [16–20]. These topologies are represented as graph $G = (V, E)$, such that V denotes the vertices and E shows the edges in these topologies. The number of V and E are shown in Table 1. Fig. 2 shows the results for the delay of the proposed DQN based mechanism with the standard benchmark K-means placement for the controller in SDN.

Table 1: V and E in the placement network for controller in SDN

Name of network	*Vertices (V)*	*Edges (E)*
Interoute	110	149
OS3E	34	41
USNet	24	42
Abilene	11	14

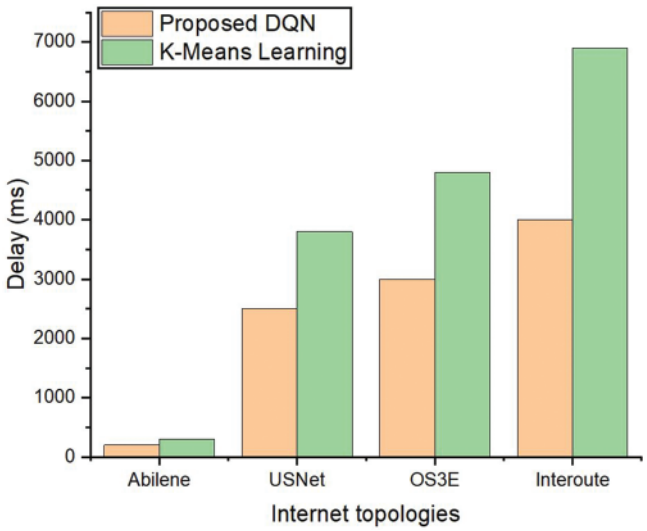

Figure 2: Delay analysis for 6 clusters from switches to controllers in the cluster

Table 2 explains the ratio of switches and the controllers in the proposed DQN and benchmark K-means clustering method. The results are indicated in the five real networks. Table 2 shows that the distribution has a greater balance or a fair distribution in the proposed DQN as compared with the benchmark approach. Moreover, the scalability of the networks with respect to an increase in the number of vertices and edges also shows that the suggested DQN scheme has the more balanced distribution of the switches against each controller. It means that if we have 6 clusters then in each cluster the number of switches assigned to a controller are balanced. Fig. 2 demonstrates that the DQN results in reduction of the delay in five networks, i.e., Interoute, OS3E, USNet and Abilene. The proposed method gives less delay although the V and E are increasing in these networks. For example, the V and E in Abilene are 11, and 14. Similarly, the vertices and edges increase in USNet and OS3E, i.e., 24 in USNet and 34 in OS3E. Moreover, the V and E in Interoute are 110 and 149. However, the DQN based strategy has less delay as compared to with benchmark K-means based placement scheme.

Table 2: The switch and controller ratio

Network topology	K-means clustering	Proposed DQN
Interoute	0.60	0.80
OS3E	0.69	0.88
USNet	0.71	0.90
Abilene	0.85	0.96

Figs. 3–5 shows the delay in 7, 8, and 9 clusters. It means that 7 controllers are placed in 7 different clusters in these topologies. Then, 8, and 9 controllers are placed. Figs. 3–5 shows that even with an increase in the number of clusters the delay for the suggested DQL scheme is less than the benchmark method. Figs. 3–5 denote the delay between controllers and switches in the five network topologies. The results show that the delay is less for the DQL-based suggested method as compared with the benchmark approach.

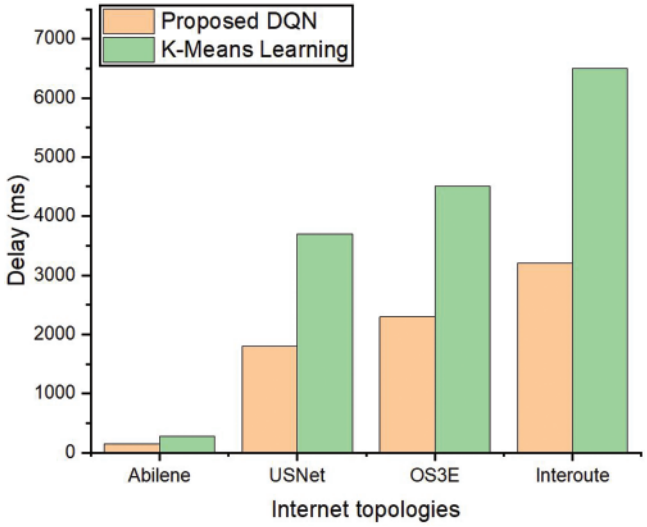

Figure 3: Delay with 7 clusters between switches and controller

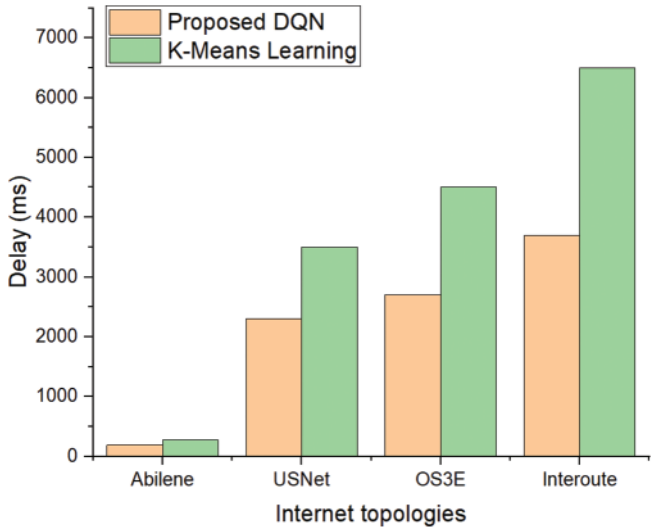

Figure 4: Delay with 8 clusters of the switches with controllers

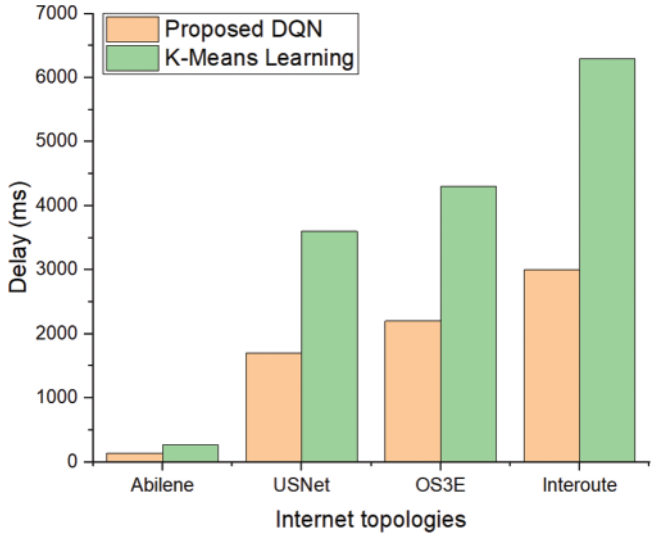

Figure 5: Delay in 9 clusters between controllers and switches

Figs. 6 and 7 provide a comparison of the communication delay between controllers across two network configurations: one with 7 clusters and another with 6 clusters. The data shows a clear reduction in delay when using the proposed DQN method with 7 clusters, compared to the K-means algorithm in the same configuration. Figs. 6 and 7 show that, the DQN approach consistently outperforms K-means in minimizing delay. In the case of K-means, there is an irregular pattern where the delay in some clusters actually increases rather than decreases, which is counterproductive to the goal of minimizing delay. This trend is particularly evident in the 7-cluster configuration, where several clusters exhibit increased delay under the K-means approach. Conversely, the DQN approach demonstrates a significant reduction in delay in this same configuration, as illustrated in the figure. The key reason for K-means' suboptimal performance in this scenario lies in its method of selecting initial cluster centers. K-means initializes these centers randomly, which can lead to poor placement of SDN

controllers, especially when the network topology and traffic patterns are not considered. As a result, this randomness can cause certain clusters to experience higher delays due to inefficient controller placements. In contrast, the DQN approach dynamically learns and optimizes controller placements by taking into account both the network structure and delay metrics, leading to more effective and consistent delay reduction.

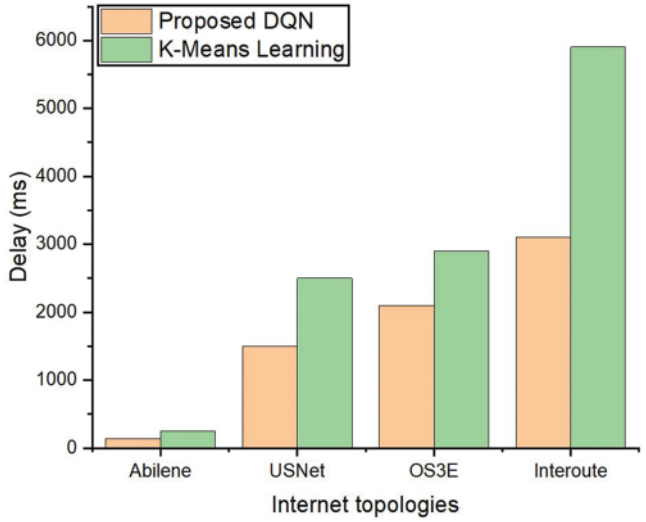

Figure 6: Controller-to-controller delay in 6 clusters

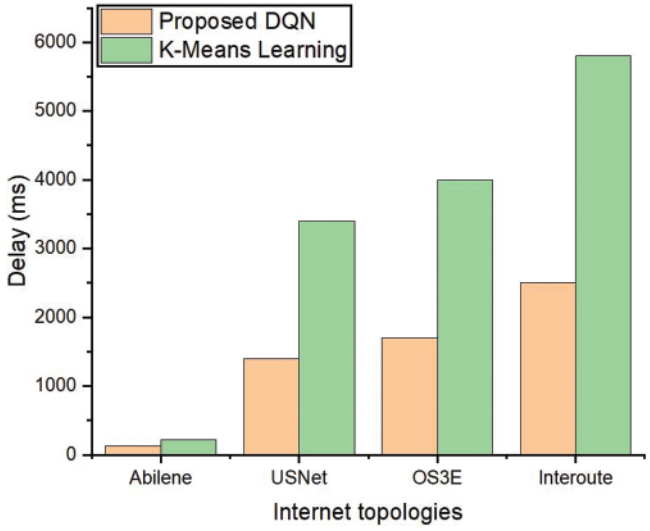

Figure 7: Controller-to-controller delay in 7 clusters

Fig. 8 illustrates the communication cost in terms of packets processed per second between switches and the SDN controller, evaluated across five different network topologies. The results show a significant difference between the benchmark method and the proposed DQN-based approach.

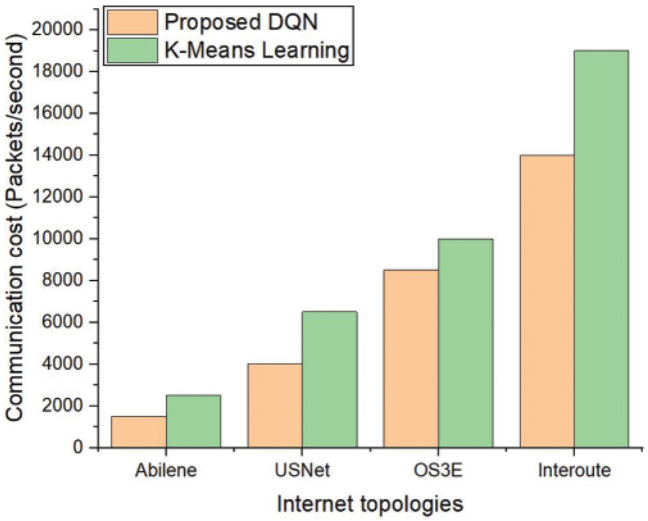

Figure 8: Cost (communication) between switches to the controller

In the benchmark method, there is a high volume of packet exchanges between switches and controllers, particularly in networks with larger topologies that have more vertices (switches) and edges (links). As the complexity of the network increases, the communication cost rises substantially. This increase occurs because the benchmark method does not efficiently optimize the controller placement, leading to more frequent and longer-distance communications between switches and the controller.

Moreover, the DQN-based scheme effectively reduces the communication cost. The DQN algorithm optimizes the placement of controllers by taking into account the network's structure, minimizing both hop count and propagation delay. As a result, fewer packets need to be exchanged between switches and controllers, even in larger or more complex network topologies. The intelligent placement of controllers reduces the number of hops and the total distance that packets need to travel, leading to a more efficient communication process. This difference is particularly evident in the larger topologies, where the DQN-based scheme demonstrates a significant reduction in communication cost compared to the benchmark. Thus, Fig. 6 highlights the advantage of the proposed DQN-based approach in optimizing controller placement, leading to reduced communication overhead and improved scalability as network size and complexity increase.

In Fig. 9, we present the communication cost between SDN controllers placed in different clusters across five network topologies. The controller placement is determined using both the proposed Deep Q-Learning (DQN) method and the benchmark K-means clustering approach. To quantify the communication cost, we measure the exchange of packets between controllers within each cluster.

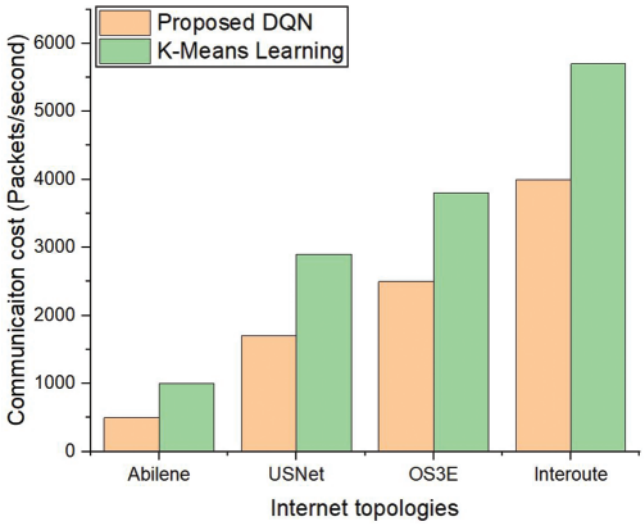

Figure 9: Cost (communication) between controller to controller

As depicted in the figure, the benchmark K-means method results in a noticeably higher communication cost across all networks. This is particularly evident in topologies with a greater number of vertices (switches) and edges (links). The increased cost arises due to the random initialization of cluster centers in the K-means algorithm, which often leads to suboptimal controller placements, causing inefficient inter-controller communication. In contrast, the proposed DQN-based approach achieves a significantly lower communication cost. The DQN scheme intelligently places the controllers by learning from the network's structure, minimizing the hop count and propagation delay between controllers. This results in fewer packet exchanges between controllers, even in more complex networks with a larger number of vertices and edges. The optimized placement ensures that communication among controllers remains efficient, regardless of network size. Fig. 9 highlights the superiority of the DQN-based approach in reducing inter-controller communication cost compared to the K-means benchmark, demonstrating its scalability and effectiveness in large and complex network topologies.

Figs. 10 and 11 present the comparison of switch-to-controller ratio across multiple real-world network topologies such as Abilene, Interoute, OS3E, and USNet. These results are analyzed using 6 and 7 clusters, respectively, to show how the placement of controllers affects the network's balance. The proposed DQN-based controller placement consistently delivers a more even distribution of controllers across switches when compared to the K-means clustering algorithm, particularly in large-scale topologies. This reflects the DQN's ability to adaptively minimize propagation delay and optimize network performance through intelligent controller placement. Figs. 8 and 9 show that with the DQN, the number of controllers is more proportional to the number of switches, resulting in better load balancing, lower latency, and improved quality of service. In contrast, K-means tends to create uneven clusters, which can lead to controller overload or inefficient resource utilization.

Figs. 10 and 11 show the potential of reinforcement learning to outperform traditional clustering algorithms, especially in complex, heterogeneous network environments, and demonstrates how DQN can enhance scalability and overall network efficiency.

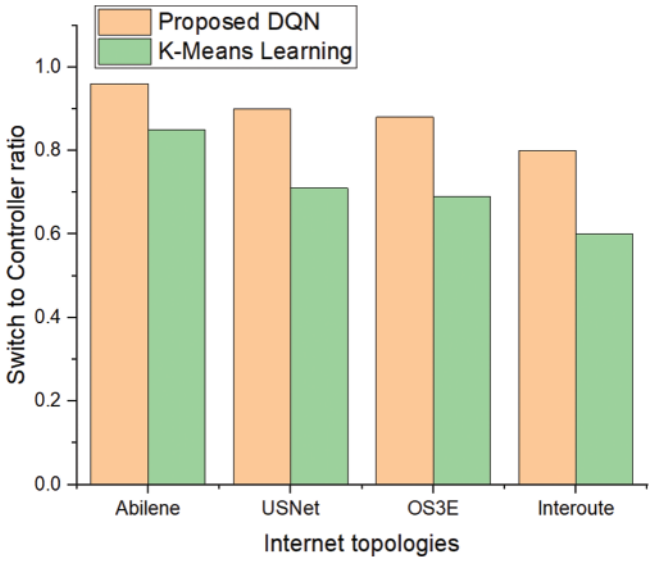

Figure 10: The switches to the controller ratio in different topologies within 6 clusters

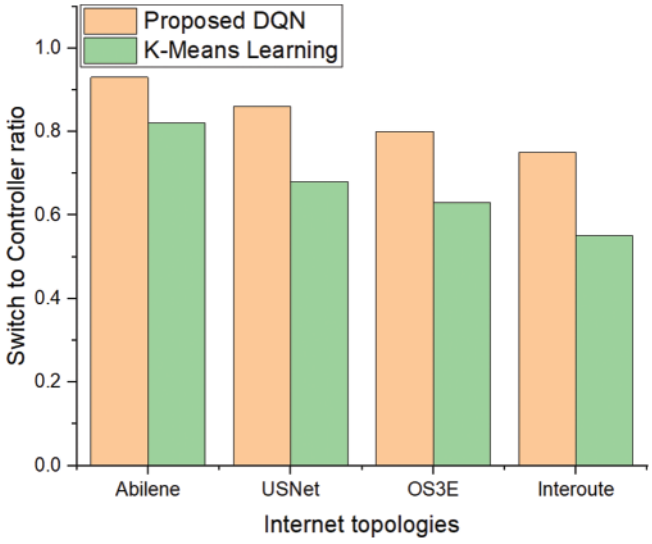

Figure 11: The switches and controller ratio in Internet topologies within 7 clusters

6 Conclusion and Future Works

Applying DQL to controller placement in SDN offers an intelligent and adaptive approach to optimizing network performance. By taking into account both hop count and propagation delay, DQL helps to significantly reduce E2E delay, a critical factor affecting the overall QoS in SDN-based networks. The placement of the controller in SDN is pivotal, as poor placement can lead to increased delays in communication between switches and controllers, thereby degrading QoS and increasing network latency. In this paper, we propose a DQL-based mechanism to effectively place controllers within the SDN to minimize delays between switches and controllers, as well as between controllers themselves. The DQL algorithm learns the optimal placement by continuously analyzing the network's

topology, traffic patterns, and delay factors. Through a dynamic learning process, the system identifies the best locations for controllers, ensuring minimal communication delay.

We conducted extensive simulations to validate the effectiveness of the proposed DQL-based scheme. The results show a significant reduction in delay compared to traditional benchmark methods, such as K-means clustering. This demonstrates that DQL can outperform static or less adaptive methods by intelligently adjusting to the changing network conditions, thereby ensuring lower latency and improving the overall network efficiency. The main advantage of using DQL in this context is its ability to adaptively respond to variations in the network. As the network topology evolves or traffic patterns shift, the DQL-based scheme continuously updates the controller placement, maintaining optimal performance without manual intervention. This adaptability ensures that latency remains minimal, even in dynamic and complex SDN environments.

Our future work will focus on extending this approach to consider additional factors such as link load, where the controller placement can further be optimized based on traffic distribution and network congestion. Moreover, we plan to evaluate the performance of the DQL-based scheme in larger and more diverse Internet topologies to further generalize its effectiveness. Moreover, we will evaluate the standard deviation and confidence intervals to check variability of the results. Moreover, Implementing DQL-based controller placement in real-world SDN environments can be challenging due to resource limitations like processing power, memory, and time constraints. However, with the use of lightweight models, experience replay, and decentralized learning approaches, it is feasible however, it requires careful optimization.

Acknowledgement: The authors extend their appreciation to Researcher Supporting Project number (RSPD2024R582), King Saud University, Riyadh, Saudi Arabia.

Funding Statement: This study was supported by the Researcher Supporting Project number (RSPD2024R582), King Saud University, Riyadh, Saudi Arabia.

Author Contributions: The authors confirm contribution to the paper as follows: Study conception and design: Jehad Ali, Mohammed J. F. Alenazi; data collection: Jehad Ali; analysis and interpretation of results: Jehad Ali, Mohammed J. F. Alenazi; draft manuscript preparation: Jehad Ali, Mohammed J. F. Alenazi. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Data available within the article.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

[1] M. N. A. Sheikh, I. S. Hwang, E. Ganesan, and R. Kharga, "Performance assessment for different SDN-based controllers," in *2021 30th Wireless Opt. Commun. Conf. (WOCC)*, Oct. 2021, pp. 24–25.

[2] J. Ali and B. H. Roh, "A novel scheme for controller selection in software-defined internet-of-things (SD-IoT)," *Sensors*, vol. 22, no. 9, 2022, Art. no. 3591. doi: 10.3390/s22093591.

[3] J. Ali and B. H. Roh, "Quality of service improvement with optimal software-defined networking controller and control plane clustering," *Comput. Mater. Contin.*, vol. 67, pp. 849–875, 2021. doi: 10.32604/cmc.2021.014576.

[4] F. Brügge *et al.*, *State of IoT*. Springer, 2023.

[5] G. Wang, Y. Zhao, J. Huang, Q. Duan, and J. Li, "A K-means-based network partition algorithm for controller placement in software defined network," in *2016 IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.

[6] J. Zhao, H. Qu, J. Zhao, Z. Luan, and Y. Guo, "Towards controller placement problem for software-defined network using affinity propagation," *Electron. Lett.*, vol. 53, no. 14, pp. 928–929, 2017. doi: 10.1049/el.2017.0093.

[7] A. Sallahi and M. St-Hilaire, "Expansion model for the controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 21, no. 2, pp. 274–277, 2016. doi: 10.1109/LCOMM.2016.2621746.

[8] A. Jalili, V. Ahmadi, M. Keshtgari, and M. Kazemi, "Controller placement in software-defined WAN using multi objective genetic algorithm," in *2015 2nd Int. Conf. Knowl.-Based Eng. Innov. (KBEI)*, 2015, pp. 656–662. doi: 10.1109/KBEI.2015.7436121.

[9] S. Lange *et al.*, "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Trans. Netw. Serv. Manage.*, vol. 12, no. 1, pp. 4–17, 2015. doi: 10.1109/TNSM.2015.2402432.

[10] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 18, no. 8, pp. 1339–1342, 2014. doi: 10.1109/LCOMM.2014.2332341.

[11] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 473–478, 2012. doi: 10.1145/2377677.2377767.

[12] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller placement in software defined wide area networks," *IEEE Trans. Netw. Serv. Manage.*, vol. 15, no. 1, pp. 344–355, 2017. doi: 10.1109/TNSM.2017.2785660.

[13] A. K. Tran, M. J. Piran, and C. Pham, "SDN controller placement in IoT networks: An optimized submodularity-based approach," *Sensors*, vol. 19, no. 24, 2019, Art. no. 5474. doi: 10.3390/s19245474.

[14] A. Jalili, M. Keshtgari, R. Akbari, and R. Javidan, "Multi criteria analysis of controller placement problem in software defined networks," *Comput. Commun.*, vol. 133, pp. 115–128, 2019. doi: 10.1016/j.comcom.2018.08.003.

[15] J. Ali and B. H. Roh, "An effective approach for controller placement in software-defined Internet-of-Things (SD-IoT)," *Sensors*, vol. 22, no. 8, 2022, Art. no. 2992. doi: 10.3390/s22082992.

[16] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *2014 IEEE Colomb. Conf. Commun. Comput. (COLCOM)*, Jun. 2014, pp. 1–6.

[17] J. Ali, H. H. Song, and B. H. Roh, "An SDN-based framework for E2E QoS guarantee in Internet-of-Things devices," *IEEE Internet Things J.*, 2024. doi: 10.1109/JIOT.2024.3465609.

[18] D. Y. Setiawan, S. N. Hertiana, and R. M. Negara, "6LoWPAN performance analysis of IoT software-defined-network-based using mininet-Io," in *2020 IEEE Int. Conf. Internet of Things Intell. Syst. (IoTaIS)*, BALI, Indonesia, 2021, pp. 60–65. doi: 10.1109/IoTaIS50849.2021.9359714.

[19] A. Ram, M. P. Dutta, and S. K. Chakraborty, "A flow-based performance evaluation on RYU SDN controller," *J. Inst. Eng. (India): Ser. B*, vol. 105, no. 2, pp. 203–215, 2024. doi: 10.1007/s40031-023-00982-0.

[20] M. N. A. Sheikh, I. S. Hwang, M. S. Raza, and M. S. Ab-Rahman, "A qualitative and comparative performance assessment of logically centralized SDN controllers via mininet emulator," *Computers*, vol. 13, no. 4, 2024, Art. no. 85. doi: 10.3390/computers13040085.