Received January 17, 2022, accepted February 2, 2022, date of publication February 15, 2022, date of current version February 18, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3151081

Deep Reinforcement Learning-Based Routing on Software-Defined Networks

GYUNGMIN KIM[1], (Graduate Student Member, IEEE), YOHAN KIM[2], (Member, IEEE), AND HYUK LIM[3], (Member, IEEE)

[1]School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology (GIST), Gwangju 61005, Republic of Korea
[2]Division of Data Analysis, Korea Institute of Science and Technology Information (KISTI), Daegu 41515, Republic of Korea
[3]Korea Institute of Energy Technology (KENTECH), Naju-si 58217, Republic of Korea

Corresponding author: Hyuk Lim (hlim@kentech.ac.kr)

This work was supported the Korea Government (MSIT) through the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant (2021-0-00379, privacy risk analysis and response technology development for AI systems).

ABSTRACT With an exponential increase in network traffic demands requiring quality of services, the need for routing optimization has become more prominent. Recently, the advent of software-defined networking (SDN) technology enables centralized management and operation, and the networking resources such as switches become flexibly configurable through programmable interfaces. In this paper, we propose a deep reinforcement learning (DRL)-based routing optimization on an SDN. In the proposed method, the DRL agent learns the interdependency between the traffic load of network switches and the network performance, and decides an optimal set of link weights to make a balance between the end-to-end delay and packet losses of the network. The SDN controller determines the routing paths using the set of link weights and installs the flow-rules on the SDN-enabled switches. To overcome an extensively long learning process of DRL in a case of topology change, we develop an M/M/1/K queue-based network model and perform the learning process of DRL using the network model in an offline manner until it is converged. The simulation results demonstrate the proposed routing method outperforms a conventional hop-count routing and a traffic demand-based RL algorithm in several network topologies.

INDEX TERMS Routing optimization, deep reinforcement learning, software-defined networking.

I. INTRODUCTION

To deal with the constant increase in network traffic volume, network operators have continued to improve resource management performance through traffic engineering techniques. Traffic engineering in communication networks coordinates the packet forwarding paths of multiple flows in the network to improve the overall quality of service for network users. Routing optimization has long been steadily studied as one of the major challenges in traffic engineering to maximize network utility [1]–[3].

In traditional routing methods, each router makes its own packet forwarding decision without any regard for other router decisions. Although this distributed routing method provides scalability because it can be applied regardless of the network scale, it is difficult to approach routing optimization of the entire network and manage network resources efficiently and flexibly. For these reasons, there is a greater need for a better network management model, and software-defined networking (SDN) has been proposed as an effective means of managing the entire network by separating control and data planes in the network, which distinguishes data transmission from control operations [4], [5]. SDN provides a global view of the entire network and improves network programmability for network operation and management by logically decoupling the control plane and data plane in the network [6]. This SDN paradigm can achieve efficient network monitoring and flexibly deploying network policies. However, while SDN enables centralized control of packet forwarding with a global view of the network, designing an optimal routing solution is not trivial. Many existing works formulate the routing problem as a constrained shortest path problem, but an optimal solution to these problems is usually NP-hard [7]. In addition, even though there is a typical solution by considering the network operation as a fixed model with varying traffic for the general multi-commodity

The associate editor coordinating the review of this manuscript and approving it for publication was Bijoy Chand Chatterjee.

flow problem, such models cannot accurately exhibit good network operation under complex and dynamic traffic [8].

Deep reinforcement learning (DRL), which combines reinforcement learning (RL) with deep neural networks, has been introduced to develop traffic engineering techniques in recent years [1]. The advancement of the DRL technique provides a new way to solve the optimization of highly complicated routing problems. DRL-based routing schemes can learn and adapt to complex networks by improving routing policy performance in an experience driven and model-free manner. Recent research has demonstrated an impressive advance in routing optimization performance by utilizing the DRL technique in an SDN-based network [8]–[15]. It should be noted here that, due to the nature of RL, which involves exploration in the process of determining the best policy, networking performance degradation may occur during the learning process, particularly in the early stages. Because an inappropriate routing policy directly increases end-to-end delay and packet loss in a network, risking network performance degradation for the sake of training reduces the system's reliability. In particular, if there is a change in the network topology, the DRL agent should re-learn for routing optimization. The more complex the characteristics of network traffic, the longer it takes to converge, resulting in long-term network performance degradation. Furthermore, in networks where QoS-sensitive traffic is transmitted, the use of DRL-based routing optimization that necessitates exploration can be disastrous.

In this paper, we propose a DRL-based routing optimization on an SDN. The DRL agent learns the interdependency between the traffic load of network switches and the network performance using a deep deterministic policy gradient (DDPG) algorithm, and decides an optimal set of link weights to reduce the end-to-end delay and packet losses of the network using the aggregated traffic volume matrix (ATVM) as an input of DRL. ATVM represents the amount of traffic volume assigned to each switch in the network. The SDN controller determines the routing paths using the set of link weights and installs the flow-rules on the SDN-enabled switches. In the propose method, we develop an M/M/1/K queue-based network model and perform the learning process of DRL using the network model in an offline manner until it is converged to overcome the network performance degradation problem during the learning process of DRL. Because an action for exploration is applied to the modeled network, the proposed method allows unrestricted exploration without affecting the data network's performance. Furthermore, because a reward for the corresponding action is obtained through interaction with the modeled network, synchronization with the data network is not required, allowing for the rapid generation of a converged routing policy with high computing power.

The main contribution of this paper is summarized as follows.

- In the SDN-based network, we propose a deep reinforcement learning (DRL)-based routing optimization method where the DRL agent is trained to decide an optimal set of link weights to minimize the end-to-end delay and packet losses of the network.
- To overcome an performance degradation problem in learning process of DRL, we develop an M/M/1/K queue-based network model and perform the learning process of DRL using the network model in an offline manner until it is converged.
- We adopt the DDPG algorithm to automatically determine link weights of network and employ ATVM as an input of DRL to improve the routing performance.
- We evaluate the routing performance of the proposed method using simulations and demonstrate the proposed routing method outperforms a conventional hop-count routing and a traffic demand-based RL algorithm in several network topologies.

The rest of this paper is structured as follows. Section II provides an overview of related network routing research. In Section III, we describe the proposed routing method as well as the MDP model for link weight decisions. In Section IV, we discuss the DDPG for link weight allocation and present the DDPG-based routing algorithm. In Section V, we present the performance evaluation of the proposed method, and in Section VI, we provide the conclusions of the study.

II. RELATED WORK

A. TRADITIONAL ROUTING METHODS

Traditional routing methods include classic traffic forwarding protocols such as Open Shortest Path First (OSPF) [16] and Equal-Cost Multi-Path (ECMP) [17], which route traffic through the shortest paths or distribute traffic evenly across multiple transmission paths. These dynamic routing rules can route traffic regardless of network topology or traffic distribution. However, their performance is far from optimal when network and traffic characteristics are not taken into account. Many research interests have been drawn to the routing optimization problem to adapt traffic routing to network conditions [3].

Fortz and Thorup [18] demonstrated a system of techniques for addressing predicted periodic changes in traffic by adjusting OSPF weights for intra-domain routing. Sridharan *et al.* [19] proposed a routing scheme that achieves near-optimal traffic splitting by activating only a subset of ECMP next hops to forward packets to the chosen destination prefix. Xu *et al.* [20] proposed the Penalizing Exponential Flow-splitting (PEFT) protocol, which forwards packets hop-by-hop based on link weights. PEFT-enabled switches exponentially distribute traffic across all possible paths, but longer paths are penalized based on total link weights along the paths. Michael and Tang [21] demonstrated the Hop-by-hop Adaptive Link-state Optimal algorithm, which results in an optimal iterative distributed procedure for quasi-static demands.

B. SDN-BASED ROUTING METHODS

SDN (Software-Defined Networking) broadens the horizon for resolving TE-related issues [22]. SDN-based routing methods can be designed based on a global view of the network and real-time traffic characteristics by using the SDN controller, which monitors overall network conditions and determines the forwarding path for each flow in the network. Agarwal *et al.* [6] developed a Fully Polynomial Time Approximation Scheme to solve the SDN controller's optimization problem for forwarding packets in an incrementally deployed SDN. Vissicchio *et al.* [23] investigated the collaboration of distributed (OSPF-based) and centralized (SDN-based) routing control planes in hybrid SDN switches. Celenlioglu and Mantar [24] proposed a routing and resource management model for SDN-based intra-domain networks that uses pre-established paths with resource reservation. This type of scheme boosts routing scalability and network resource utilization. Based on the simulated annealing algorithm, Deng and Wang [25] proposed an application-aware QoS routing algorithm for SDN-based IoT systems. Rezende *et al.* [26] concentrated on the limitations of traditional multistream protocols and used SDN to distribute packets across multiple paths. The authors also provided application interfaces and improved the QoS provided to end-users.

C. MACHINE LEARNING-BASED ROUTING METHODS

Recently various methods related to machine learning-based network routing have been proposed. Mao *et al.* [27] employed a supervised deep learning technique to construct the routing table for deciding the next routing node. Zhou *et al.* [28] proposed a Q-learning-based localization-free any path routing protocol to prolong the life as well as reduce the end-to-end delay for underwater sensor networks. Pham *et al.* [15] exploited a DRL agent with convolutional neural networks in the context of knowledge-defined networking to enhance the performance of QoS-aware routing. Huang *et al.* [14] proposed a DDPG-based quality of experience optimization method for multimedia traffic. Xu *et al.* [29] proposed an experience-driven routing scheme for multiple end-to-end communication sessions. The proposed combined traffic engineering-aware exploration and actor-critic-based Prioritized Experience Replay methods for optimizing network delay. Suarez-Varela *et al.* [30] proposed feature engineering for DRL-based routing in the context of optical transport networks and IP networks. Zhang *et al.* [11] introduced a critical flow rerouting reinforcement learning scheme (CFR-RL). CFR-RL balances link utilization of the network by forwarding the majority of flows using equal-cost multi-path (ECMP) and rerouting a few critical flows selected by the agent to utilize the network link optimally. Zhang *et al.* [31] also proposed SmartEntry, which is a destination-based routing solution that reduces the number

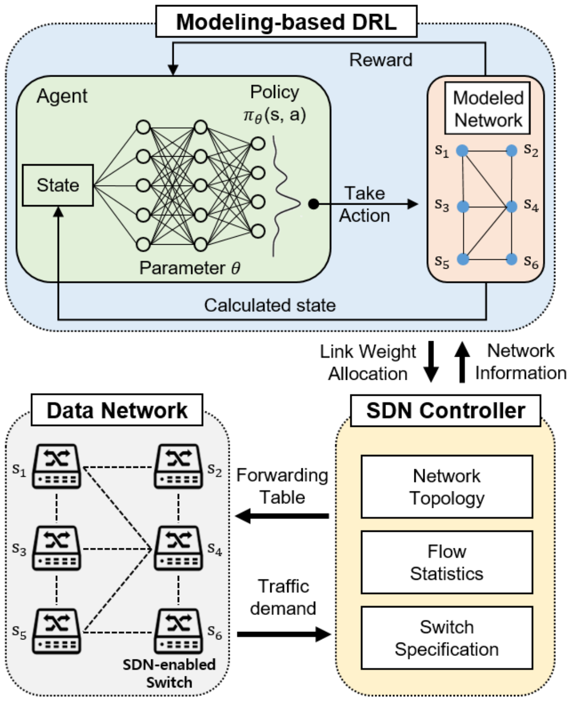


FIGURE 1. Overall system architecture of proposed DRL-based routing on SDN.

of forwarding entries that need to be updated to respond to dynamic changes of traffic demands using RL and linear programming. Fu *et al.* [12] proposed a routing strategy based on deep Q-learning designed for data center networks. The authors assumed that mice and elephant flows had different requirements and focused on meeting these traffic demands in respect of throughput, latency, and packet loss. For SDN-IoT, Guo *et al.* [13] proposed DQSP which is a DDPG based secure and QoS-aware routing method. Chen *et al.* [9] thoroughly examines the need for optimized routing in SDN. The authors presented RL-Routing and proved that their method offers better results than other routing algorithms like OSPF and Least Loaded after an extensive evaluation based on a real SDN controller and networks. Sun *et al.* [10] leveraged the idea from the pinning control theory to select a subset of links in the network to solve the scalability problem that occurs when DRL is adopted to learn the link weights for huge network. Later, Sun *et al.* [8] proposed ScaleDeep to automatically adjust routing policy to improve network performance and tolerate network topology changes by dividing the nodes on the network into two classes: driver nodes and follower nodes. The authors employed pinning control to alleviate the curse of dimensionality problem when the network scale rises.

Despite numerous research efforts to improve routing performance through learning techniques, particularly RL techniques, there is still a problem in which the performance of the data network is degraded due to exploration problems in the learning process and the longer it takes to converge the more complex the traffic characteristics. In this paper, by employing the M/M/1/K queue-based network model in learning process, we can prevent the performance degradation during the learning process of RL. In addition, we can obtain a converged neural model rapidly by acquiring and utilizing the state and reward for training from the modeled network.

III. NETWORK ROUTING SYSTEM MODEL USING MDP

A. OVERALL SYSTEM ARCHITECTURE

As shown in Figure 1, we consider an SDN-based routing system architecture where the SDN controller manages packet forwarding by following the link weight allocation provided by the DRL agent with the M/M/1/K queue-based network model. The modeled network in the proposed system is constructed based on network information such as network topology and switch specification provided by the SDN controller of the control plane and thus reflects the characteristics of the data network of the data plane.

Considering the importance of trust in the network system and the sensitivity of quality of service for flows, the proposed routing system is designed to solve exploration issues, particularly in the early stages of learning technique, via modeling-based training. In the proposed system, the DRL agent learns through rewards of actions applied to the modeled network rather than direct interaction with the data network. The sophisticated network modeling-based method allows the learning agent to obtain an effectively trained neural model without degrading the performance of the data network. At every iteration in the learning process, the agent obtains the state and reward values for the action applied to the data network from the model of the data network rather than the measurement values on the data network in real-time. Note that an iteration time step does not correspond to an actual time value. As a result, the agent can obtain a converged neural model much faster than the online method that interacts with the actual data network in real-time. The amount of time for the learning process depends on the computing power for the computation. In addition, it is possible to train the neural model for the various traffic demands using the model of the data network. We generate the synthetic traffic demands and train the neural model using the traffic to make the model have the capability of responding to various traffic states without re-performing the learning process for a different traffic demand. Therefore, the hyper-parameters of the neural model should be appropriately adjusted depending on the data network size and the dynamics of the traffic demands.

In the proposed method, the learning agent trains the neural networks using the modeled network based on network topology and traffic demand of the data network. The DRL agent employs the ATVM of each switch as the network state and aims to learn the weight for each link as the action. The DRL agent updates neural networks of actors and critics to optimize the link weight allocation based on the DDPG algorithm and provides the tuned link weights to the SDN controller when expected routing performance improves and converges. Here it is worth noting that the learning process to find the optimal link weights in the proposed architecture is performed using only the modeled network so that it does not take the risk of causing network performance degradation by directly applying the uncertainty risk to the target network.

B. SYSTEM MODEL

In this paper, we consider a backbone network composed of SDN-enabled switches and edge switches are connected to traffic sources such as multiple clients or servers. Here, it is assumed that each edge switch is the departure or arrival point of the backbone network. We assume that there are $\mathcal{N}$ switches in the SDN-based network and the switches are denoted by $\boldsymbol{V} = [v_1, v_2, \ldots, v_N]^\intercal$. The service rate of the n-th switch is denoted as μ_n. Here, we focus on the routing of traffic within a single autonomous system. For a communication network $\boldsymbol{G}$, we use $\boldsymbol{E}$ to denote the set of the links e among the switches, then we have $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$. For any pair of switches $v_i, v_j \in \boldsymbol{V}$ $(i \neq j)$, it is assumed that there is at least one forwarding path $p_{i,j} = \{e_1, e_2, \ldots, e_{|p|}\}$ that can forward the traffic from v_i to v_j. Among the paths, we denote the shortest path as $p^*_{i,j} = \{e^{i,j}_1, e^{i,j}_2, \ldots, e^{i,j}_{|p^*|}\}$. Here, the shortest path is calculated with a weighted shortest path algorithm on $\boldsymbol{G}$, where the weight value for link e_m is denoted by w_m $(0 \leq m \leq |\boldsymbol{E}|)$. This path also can be expressed as $p^*_{i,j} = \{v^{i,j}_1, v^{i,j}_2, \ldots, v^{i,j}_{|p^*|}\}$ where $v^{i,j}_l$ is the l-th switch when forwarded through the shortest path. $v^{i,j}_1$ and $v^{i,j}_{|p^*|}$ are the same as v_i and v_j, respectively.

At the time step t, let f^k_t $(0 \leq k \leq M_t)$ denote the k-th traffic flow where M_t is the number of flows in the network at t. Here, a flow is defined as a source-destination pair. In the network routing system, f^k_t is forwarded via selected switches in accordance with the calculated shortest path, and the data processing and transmission delay occur as the flow passes through switches along the path. In this paper, we assume Poisson arrivals with a rate λ^k_t and exponentially distributed service times. Here, the data processing and transmission delay can be modeled as M/M/1/K queue when each switch has a limited system capacity size. Then, the expected delay in v_n at time step t can be obtained as follows:

$$\mathrm{E}[d_n(t)] = \frac{\mathrm{E}[N_n(t)]}{\lambda_n(t)(1 - P^n_b(t))}, \tag{1}$$

where $\lambda_n(t)$ is the aggregate arrival rate into v_n at t. $P^n_b(t)$ is the probability that a packet is lost due to a buffer overflow in v_n at t, and $N_n(t)$ is the queue occupation that represents the number of packets in the switch at t. Here, $P^n_b(t)$ can be obtained as follows:

$$P^n_b(t) = \frac{(1 - \rho_n(t))(\rho_n(t))^{K_n}}{1 - (\rho_n(t))^{K_n+1}}, \tag{2}$$

where $\rho_n(t) = \lambda_n(t)/\mu_n$ and K_n denote the traffic intensity at switch n and the total system capacity of the switch n, respectively. In addition, the expected queue occupation can be obtained as follows:

$$E[N_n(t)] = \begin{cases} \frac{\rho_n(t)}{1-\rho_n(t)} - \frac{(K_n+1)(\rho_n(t))^{K_n+1}}{1-(\rho_n(t))^{K_n+1}} & \text{if } \rho_n(t) < 1 \\ \frac{K_n}{2} & \text{if } \rho_n(t) = 1. \end{cases} \tag{3}$$

Under the assumption that the propagation delay over the link is negligibly small, the end-to-end delay of f_t^k forwarded through $p_{i,j}^*$ can be obtained as follows [32]:

$$D_{e2e}^k(t) = \sum_{n \in path(p_{i,j}^*)} E[d_n(t)], \tag{4}$$

where $path(p_{i,j}^*)$ represents the set of switches on the forwarding path from v_i to v_j. Then, the average end-to-end delay of flows in the network can be obtained as follows:

$$D_{e2e}^{avg}(t) = \frac{1}{M_t} \sum_{k=1}^{M_t} D_{e2e}^k(t). \tag{5}$$

In addition, the expected loss traffic of switch n and expected total loss traffic of network at time step t can be obtained as follows:

$$E[L_n(t)] = \lambda_n(t) P_b^n(t) \tag{6}$$

$$E[L_{tot}(t)] = \sum_{n=1}^{\mathcal{N}} \lambda_n(t) P_b^n(t). \tag{7}$$

The sophisticated network modeling-based method enables the learning agent to obtain an effectively trained neural model while maintaining the data network's performance. It is worth noting that a single-path routing such as OSPF is assumed in this work, but it is can be easily extended to other routing algorithms such as multi-path routing. Depending on a routing algorithm, the data network would have different states and rewards for the same set of link weights. However, the routing algorithm is considered as a part of the environment in the context of RL, and the RL agent can respond to different environments if the learning is re-performed appropriately.

C. MDP MODEL FOR LINK WEIGHT ALLOCATION

In this paper, we use the ATVM as a network state representation. The ATVM shows the traffic rate transmitted from each switch to the next switch in line. Let $\boldsymbol{T} = [t_{i,j}]_{\mathcal{N},\mathcal{N}}$ denote the ATVM of the data network, where $t_{i,j}$ is given by the traffic amount from the i-th switch to the j-th switch. For a given network topology, the ATVM of the network is determined by the traffic demand of data flows and their routing paths on the network. This representation of network states allows the agent to learn the interdependency between the links because it reflects both the link utilization of the network and the interconnection topology among switches of the network. In the proposed modeling-based method, there is no interaction between the DRL agent and the SDN controller during the training process. It means that the agent does not measure aggregated traffic volume at switches during

TABLE 1. Frequently used parameters.

Parameter	Definition
t	Time step
n	Index of switch
k	Index of flow
$\mathcal{N}$	Number of switches
M_k	Number of flows
v_n	The n-th switch
μ_n	Service rate of the n-th switch
f_t^k	The k-th traffic flow at t
$P_b^n(t)$	Packet loss probability in v_n at t
K_n	Total system capacity of v_n
$N_n(t)$	Queue occupation at t
$\lambda_n(t)$	Aggregate arrival rate into v_n at t
$L_n(t)$	Expected loss traffic of v_n at t
$\rho_n(t)$	Utilization of v_n at t
$d_n(t)$	Expected delay in v_n at t

the learning process. Instead, the ATVM is computed by exploiting the data network model and the decision of routing paths, which is determined by a routing algorithm and the link weights of the previous action.

In the proposed method, it is worth noticing that the aggregated traffic volume value measured by each switch in the actual network is almost the same as the value calculated in the fine modeling-based learning environment for the same traffic demand and deployed link weight allocation. Based on this sameness, the DRL agent can train a neural model suitable for application to the data network, and it makes the agent able to utilize the unrestricted routing policies without affecting the performance of the data network. In addition, these meaningful pairs of state and reward are stored in the replay buffer of the off-policy-based DRL agent and can be utilized in the experience replay process later.

1) STATE AND ACTION SPACE

The observation of each switch v_n at time step t, denoted as $o_n(t)$, can be represented by

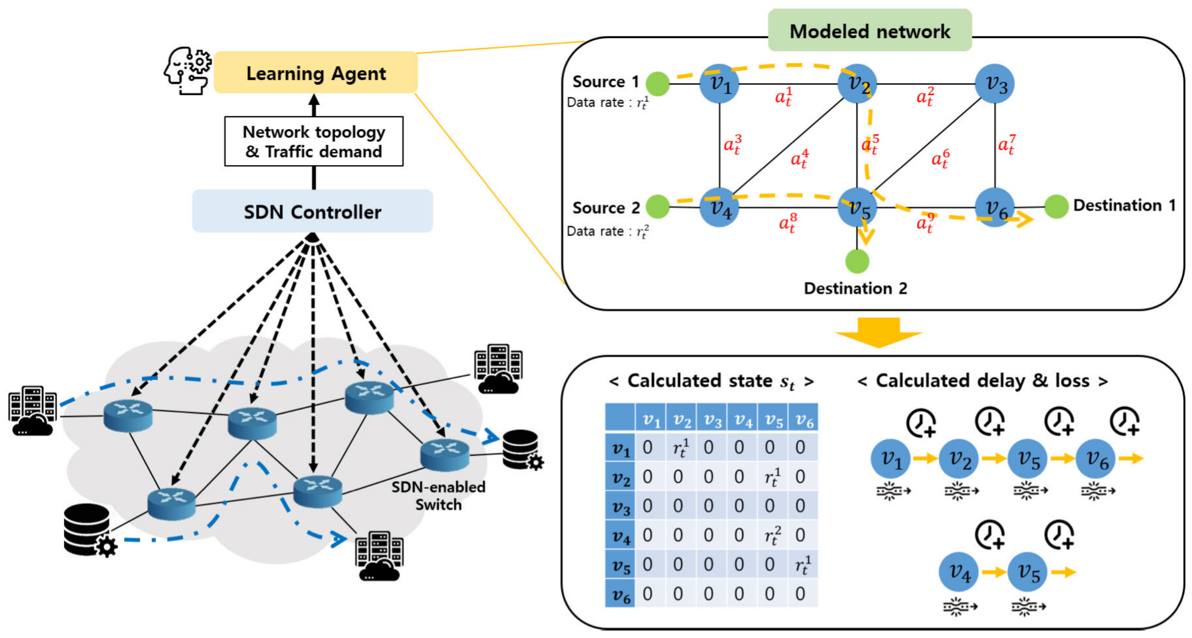
$$o_t^n = [t, K_n, N_n(t), \lambda_n(t), L_n(t), \rho_n(t), d_n(t)]. \tag{8}$$

The parameters are listed in Table 1. In the proposed method, instead of measurement on each switch in the data plane, these observations are obtained from modeled network in the agent. The modeled network is constructed based on network information, such as network topology and specification of switches, reported by the SDN controller. At every time step t, the shortest paths of flows according to the link weights assigned by the previous action is calculated in modeled network, and the network state at time step t, denoted as s_t, is be obtained as follows:

$$s_t = \{ s_t^{i,j} \mid i\,,\ j \in \boldsymbol{V} \} \tag{9}$$

$$s_t^{i,j} = \min(1, \frac{1}{\mu_{max}} \sum_{k=1}^{M_t} \lambda_t^{k(i)} x_{ij}^k(t)) \tag{10}$$

where $\lambda_t^{k(i)} = \prod_{l \in \text{path}(src(k) \to i)} (1 - P_b^l(t)) \cdot \lambda_t^k$ and $x_{ij}^k(t)$ represents the binary indicator to indicate whether the link from v_i to v_j is included in the path through which the k-th



< Calculated state s_t >

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	r_t^1	0	0	0	0
v_2	0	0	0	0	r_t^1	0
v_3	0	0	0	0	0	0
v_4	0	0	0	0	r_t^2	0
v_5	0	0	0	0	0	r_t^1
v_6	0	0	0	0	0	0

FIGURE 2. Simple example of modeling-based process.

flow is transmitted according to the shortest path. μ_{max} is the maximum service rate of switches, and we use the min($\cdot$) function to adjust the element range of state to [0, 1]. Note that the state $s_t^{i,j}$ corresponds to the element $t_{i,j}$ of ATVM normalized by μ_{max}.

Meanwhile, at each time step t, the DRL agent determines the action a_t to allocate the link weights in accordance with the s_t. The action space is defined as follows:

$$a_t = a_t^1 \times a_t^2 \times \cdots \times a_t^{|\boldsymbol{E}|} \tag{11}$$

where a_t^m $(0 \leq m \leq |\boldsymbol{E}|)$ denotes the weight value assigned to m-th link at time step t. The range of each weight is defined as $a_t^m \in [w_{min}, w_{max}]$ where w_{min} and w_{max} denote the minimum and maximum values of link weight, respectively. The action determined by the DRL agent indicates the weight of links, and link weight allocation indicates the routing policy of the network in that packet forwarding is determined by the weighted shortest path algorithm.

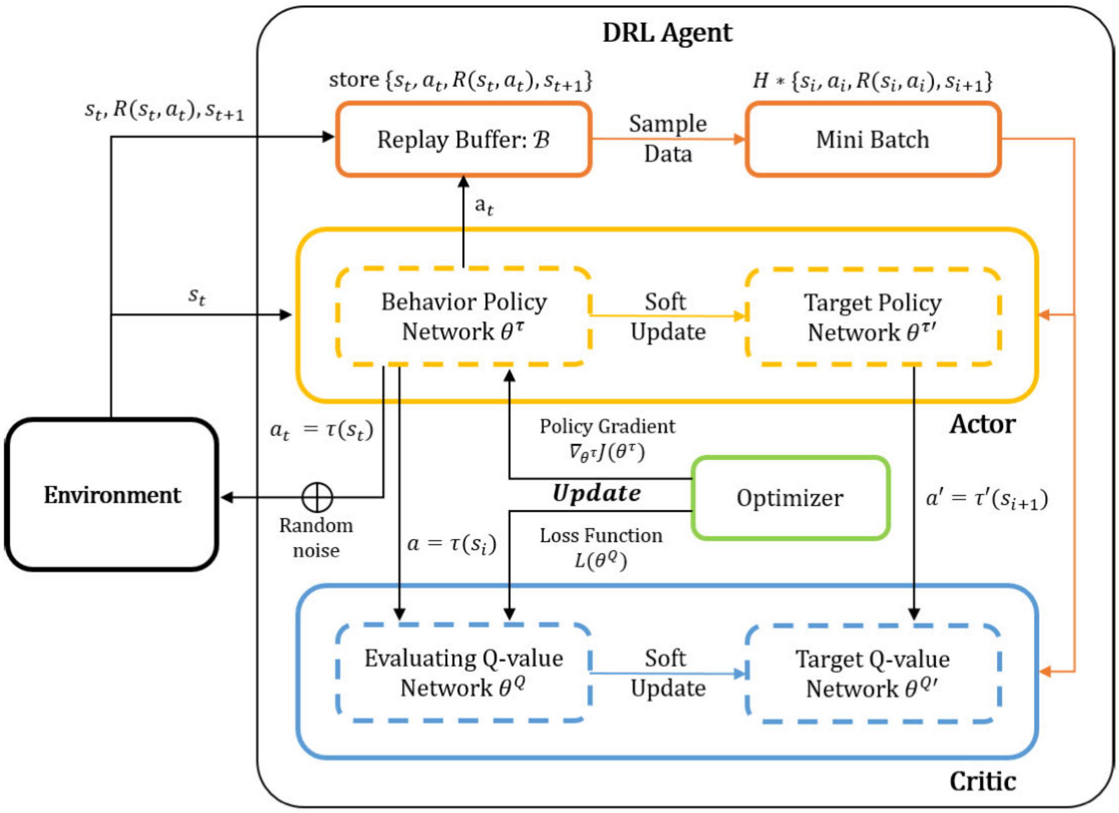
Figure 2 shows a simple example of calculating the state and its corresponding delay and loss without directly affecting the data network using a modeling-based environment. The network has six switches and there are two traffic demands in the network. The SDN controller manages packet forwarding of the data network following its routing policy and reports the information of data network to the learning agent. For a given network topology, the DRL agent trains the neural network without disturbing the data network. The information for DRL such as the network state, average end-to-end delay, and expected total packet loss is calculated using modeled network. Unlike the traffic demand matrix (TDM) in [15] that corresponds to the traffic volume between source-destination pairs of edge switches, ATVM is a summary of the traffic volume aggregated at each router as traffic flows pass through the routers along the path as shown in Figure 2. Since the routing paths of the data network and the paths applied by the modeled network are different, the selected actions in the training process do not degrade the data network. In addition, since learning is performed based on information of data network, meaningful experiences are accumulated in the replay buffer during the pre-training process, and the trained neural model suitable for immediate application to the network can be obtained.

2) TRANSITION PROBABILITY AND REWARD FUNCTION

The probability that the network routing system undergoes a transition from state s_t to state s_{t+1} when action a_t is taken can be represented as $\mathbb{P}_{sa} : (s, a) \rightarrow s_{t+1}$. Since the weight allocation to each link can be changed dynamically, transition probability $\mathbb{P}_{sa}$ can be determined using the probability of selecting an action a_t at state s_t defined by policy π.

The goal of this paper is to minimize the average end-to-end delay of network flows by optimizing link weights. As a result, the routing system's reward should be inversely proportional to the average delay. However, it should be noted that overall network performance may be degraded when a high score is granted to improper link allocation that brings a short delay with reduced traffic by causing a large amount of packet loss. Therefore, the comprehensive reward of the proposed routing system is designed to achieve two main goals: (1) minimizing the average end-to-end delay of flows, (2) suppressing the increase in the amount of packet loss at switches. Here, we define the delay performance reward $r_d(t)$ and packet loss reward $r_p(t)$ at time step t as follows:

$$r_d(t) = 1 - \frac{D_{e2e}^{avg}(t)}{\sum_{n \in path(p_{max})} \frac{K_n}{\mu_n}} \tag{12}$$



DRL Agent
store $\{s_t, a_t, R(s_t,a_t), s_{t+1}\}$
$H * \{s_i, a_i, R(s_i,a_i), s_{i+1}\}$
$s_t, R(s_t,a_t), s_{t+1}$
Replay Buffer: $\mathcal{B}$
Sample Data
Mini Batch
a_t
s_t
Behavior Policy Network θ^{τ}
Soft Update
Target Policy Network $\theta^{\tau'}$
Actor
$a_t = \tau(s_t)$
Policy Gradient $\nabla_{\theta^\tau} J(\theta^\tau)$
Environment
Update
Optimizer
$a' = \tau'(s_{i+1})$
Random noise
$a = \tau(s_i)$
Loss Function $L(\theta^Q)$
Evaluating Q-value Network θ^Q
Soft Update
Target Q-value Network $\theta^{Q'}$
Critic

FIGURE 3. **DDPG training process diagram.**

and

$$r_p(t) = 1 - \frac{L_{tot}(t)}{\sum_{n=1}^{\mathcal{N}} \lambda_n(t)}. \tag{13}$$

Here, it is assumed that p_{max} is the path through the most hops. Then, let R denote the reward function that returns a value indicating whether link weights are allocated to minimize the average end-to-end delay by the routing algorithm while taking into account reducing packet loss due to bottlenecks. When action a_t is taken in the state s_t, the reward function is defined as follows:

$$R(s_t, a_t) = \alpha\, r_d(t) + (1-\alpha)\, r_p(t) \tag{14}$$

where the range of $R(s_t, a_t)$ is [0, 1], and α denotes the weight factor for balancing the weight on delay and packet loss of the network. The reward function in (14) aims to reduce a weighted linear combination of performance metrics for network delay and packet loss. However, depending on the needs of network operation, the reward function can be defined in other forms such as maximizing throughput or minimizing maximum link utilization.

To consider the impact of current action on future rewards, we define the total expected discounted reward under policy π as follows:

$$R_t^{\pi} = R(s_t, a_t) + \sum_{i=1}^{\infty} \gamma^i \cdot R(s_{t+i}, a_{t+i}), \tag{15}$$

where $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards from time step $t+1$ until the infinite time step. In other words, R_t^{π} is defined as the sum of the current reward at time step t and the discounted reward from time step $t+1$ to the infinite time step. For example, $\gamma = 0$ indicates that the network routing system takes account of only the current reward, whereas $\gamma = 1$ means that the system considers the long-term reward at an infinite time step with the same weight as the current reward. The objective of the network routing system is to identify an action-selection policy that increases the total expected discounted reward R_t^{π} to improve the routing performance of the SDN-based network.

IV. NETWORK ROUTING ALGORITHM USING DDPG

A. DDPG FOR LINK WEIGHT ALLOCATION

In the proposed system, the DRL agent trains the neural model to identify the optimal action-selection policy for packet forwarding with balanced link weight allocation leading to reducing the delays and packet loss in the network. In order to formulate this objective, we employ the action-value function that returns the total expected discounted reward when action a_t is taken in the state s_t in accordance with the policy π as:

$$Q(s_t, a_t) = \mathrm{E}\left\{ R(s_t, a_t) + \sum_{i} \gamma^i \cdot R(s_{t+i}, a_{t+i}) \right\}. \tag{16}$$

Here, the optimal action-value function Q^* can be approximated as follows:

$$Q^*(s_t, a_t) = \mathrm{E}\Big\{R(s_t, a_t) + \sum_i \gamma^i \max_{a_{t+i}} R(s_{t+i}, a_{t+i})|_{s=s_t, a=a_t}\Big\}. \quad (17)$$

In this paper, we use a DDPG algorithm with model-free, off-policy, and actor-critic properties to learn the action-selection policy that maximizes the total expected discounted reward. Using two deep neural networks, as shown in Figure 3, this algorithm can be used to approximate the optimal action-value function in continuous action space. The first neural network is an actor-network that approximates behavior policy. Its input is observation and its output is action-value, i.e., $\tau(s_t)$. The second neural network is a critic network that approximates the action-value function. Its inputs are action and observation, and the output is the value of the action-value function, i.e., $Q(s_t, a_t)$.

Let $\tau(s|\theta^\tau)$ and $Q(s, a|\theta^Q)$ denote the actor network and critic network, respectively. In addition, let us denote τ' and Q' as the network functions for the actor and critic target network, respectively. Then, the DDPG employs θ^τ for the actor network and θ^Q for the critic network to parameterize non-linear function approximators. Here, θ^τ and θ^Q are updated in accordance with the policy gradient and loss function, respectively, as shown in Figure 3. The weight of critic network θ^Q is updated in the direction of minimizing loss as follows:

$$L(\theta^Q) = \frac{1}{H}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2, \quad (18)$$

where $y_i = R(s_i, a_i) + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\tau'})|\theta^{Q'})$, and H denotes the number of randomly sampled tuples from the replay buffer $\mathcal{B}$. That is, H means the size of the mini batch. The DDPG optimizes θ^τ by updating $J(\theta^\tau)$, which is the objective function of the policy gradient method in the direction of $\nabla_{\theta^\tau} J(\theta^\tau)$ defined as follows:

$$\nabla_{\theta^\tau} J(\theta^\tau) \approx \frac{1}{H}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\tau(s_i)} \nabla_{\theta^\tau}\tau(s|\theta^\tau)|_{s=s_i}. \quad (19)$$

To update the target networks gradually, the DRL agent updates $\theta^{\tau'}$ and $\theta^{Q'}$ as follows:

$$\theta^{Q'} = \epsilon_c \theta^Q + (1-\epsilon_c)\theta^{Q'}, \quad (20)$$

$$\theta^{\tau'} = \epsilon_a \theta^\tau + (1-\epsilon_a)\theta^{\tau'}, \quad (21)$$

where ϵ_c and ϵ_a denote positive small numbers between 0 and 1 used to learn rates corresponding to the actor network and critic network, respectively.

B. DDPG-BASED ROUTING ALGORITHM

The proposed DDPG-based routing policy update solution is described in Algorithm 1. First, the neural network models in DRL agent are initialized with randomly generated weights and initial observations. The M/M/1/K queue-based network model is built using network data from the SDN controller. The initial observation is derived from the network topology and traffic demand information reported by the SDN controller, as well as the state and reward information calculated in the modeled network environment. The DRL agent trains the neural model using synthetic traffic demands that are randomly generated. Here, the learning performance can be significantly improved by exploiting the pre-collected network statistics that have been observed in the target data network.

Algorithm 1 The Proposed DDPG-Based Routing Algorithm

```
1:  // Initialization
2:  Set the critic-network $Q(s, a|\theta^Q)$ and actor-network $\tau(s|\theta^\tau)$ with randomly generated weight $\theta^Q$ and $\theta^\tau$
3:  Set target parameters equal to main parameters $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\tau'} \leftarrow \theta^\tau$
4:  Empty experience replay buffer $\mathcal{B}$
5:  Construct the M/M/1/K queue-based network model using network information from SDN controller.
6:  Set initial state $s_0$ in accordance with the initial routing policy
7:  // Parameter updating
8:  **repeat**
9:      Select action $a_t = \tau(s_t|\theta^\tau) + \mathcal{N}$ following the parameter noise for exploration
10:     Take action $a_t$ on modeled network and calculate $R(s_t, a_t)$, $s_{t+1}$
11:     Store transition $\{s_t, a_t, R(s_t, a_t), s_{t+1}\}$ in $\mathcal{B}$
12:     **if** it's time to update **then**
13:         Update the network information from SDN controller.
14:     **end if**
15:     Randomly sample a batch of $H$ transitions $\{s_i, a_i, R(s_i, a_i), s_{i+1}\}$ from $\mathcal{B}$
16:     Set $y_i = R(s_i, a_i) + \gamma Q'(s_{i+1}, \tau'(s_{i+1}|\theta^{\tau'})|\theta^Q)$
17:     Update critic $\theta^Q$ and actor $\theta^\tau$ in (18) and (19)
18:     Update the targets softly in (20) and (21)
19: **until** convergence
```

The algorithm consists of a loop for time step t. Lines 7–19 show the loop for each time step t. At every time step t, the agent selects an action a_t following the actor-network policy with adaptive noise process $\mathcal{N}$ in line 9. To make DDPG policies explore better, adaptive noise is added to the selected action during the training time because the action-network policy is deterministic. In line 10, the DRL agent takes action a_t on modeled network and calculates the reward $R(s_t, a_t)$ and the next state s_{t+1}. The observed transition is stored to the replay buffer $\mathcal{B}$ for each action. When network information is reported from the SDN controller every T cycle, the previous network information is updated with the currently reported information. T is the period at which the SDN controller observes network information and delivers it to the DRL agent. The value for T is a design parameter that can be

TABLE 2. Simulation parameters.

Simulation parameter	Grid	GEANT	InternetMCI
The number of switches	25	40	19
The number of links	40	61	33
The number of flows	150	150	100
α	0.9	0.9	0.9
w_{min}	1	1	1
w_{max}	5	5	5

TABLE 3. Hyper-parameter values used for the DDPG algorithm.

Hyper-parameter	Value
Discount factor γ	0.99
Replay buffer $\mathcal{B}$	50,000
Batch size H	100
Critic learning rate ϵ_c	0.00001
Actor learning rate ϵ_a	0.00001

adjusted in a long term or a short term according to the needs of the network operation. Based on the randomly sampled mini batch from $\mathcal{B}$, θ^Q and θ^τ are updated in accordance with the updated rules (20) and (21) in lines 15 – 18.

The proposed algorithm can mitigate the performance degradation that occurs in the learning stage and is suitable for a situation where the traffic demand dynamically changes over time in the backbone network environment. However, for an environment with highly-dynamic topology changing (e.g., wireless and ad hoc networks, Internet of Things, etc.), training a new model each time may not be computation-efficient. In this case, we can improve the efficiency of computation by adopting a graph neural networks to generate a generalized model for multiple topologies [33], [34].

V. PERFORMANCE EVALUATION

A. SIMULATION SETUP

In this section, we present the results of simulations designed to evaluate the performance of the proposed DDPG-based routing algorithm in an SDN-based network. The simulations were conducted using the Python networkX library for network topology generation and a stable-baseline framework for the DDPG algorithm [35], [36]. We obtained all numerical results from the executions on a server equipped with an Intel i9-10940X CPU @ 3.3 GHz and NVIDIA Quadro RTX6000 with 24 GB of memory and with CUDA 11.1. We considered three kinds of network topologies: a gird topology where 25 switches were configured in the 5×5 grid shape with 40 full-duplex links, a GEANT [37] topology of 40 nodes and 61 full-duplex links, and an InternetMCI [37] topology of 19 nodes and 33 full-duplex links as shown in Figure 4(a), 4(b), and 4(c), respectively.

The system capacities and the service rate for topologies are set to 10,000 packets and 3,000 packets/second, respectively. The arrival rate of k-th flow entering the network is governed by a Poisson process with average λ^k where λ^k is set by a uniform random distribution in the interval from 10 to 300 packets/second. To consider network flow uncertainty, we randomly selected source and destination switch pairs and changed the λ^k at every iteration. The weight factor α was set to 0.9 and link weights are selected in the range from 1 to 5. The shortest path of each flow was calculated with Dijkstra's algorithm, which is widely used as the weighted shortest path algorithm. The simulation parameters are listed in Table 2.

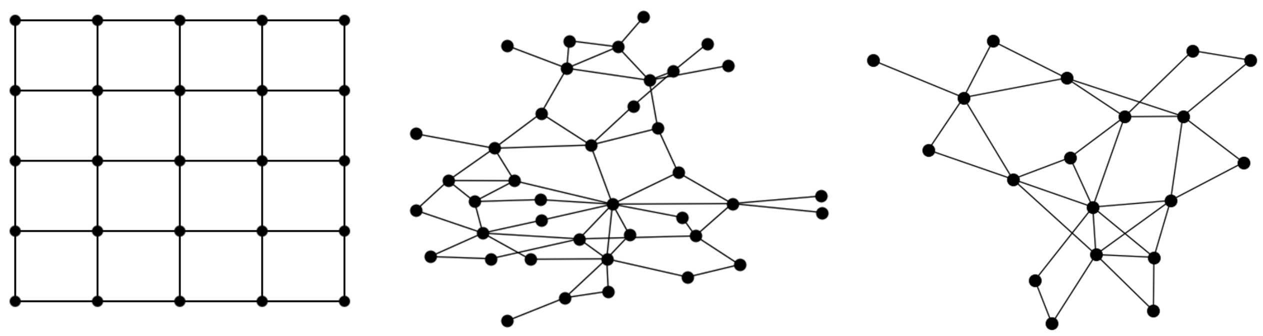
We used two fully-connected hidden layers with 400 and 300 units for the actor network and the critic network, respectively, to implement the DDPG-based algorithm. We used the rectified linear units activation function, which is reliable and fast. The Adam [38] optimizer is used to train neural networks. The discount factor γ was set to 0.99. The batch size of the mini batch was set to 100, and learning began after 100 steps to collect transitions before learning began. The learning rate for the Adam optimizer and both actor and critic networks are set to 10^{-5}. We use the Ornstein-Uhlenbeck process [39] to produce noise that is added in the exploration policy to help the agent explore the environment thoroughly. We set the hyper-parameter values of the DDPG algorithm as shown in Table 3.

B. SIMULATION RESULTS

Figures 5(a), 5(b), and 5(c) show the rewards of the agent with respect to the number of iterations when using the proposed method. In the figures, the reported line is a moving average of 500-time steps. In the figures, it is seen that the rewards keep increasing over the iterations in three topologies and it indicates the algorithm can choose better actions as it experiences iteration. In Figures 5(a) and 5(b), since the number of flows is the same, the reward is larger in the GEANT topology with larger network size, and the variation of reward is small in the grid topology where the diversity of paths is relatively low. In addition, it can be seen that in the GEANT topology with a relatively large network size, more time is required for convergence as the state space is larger and complicated.

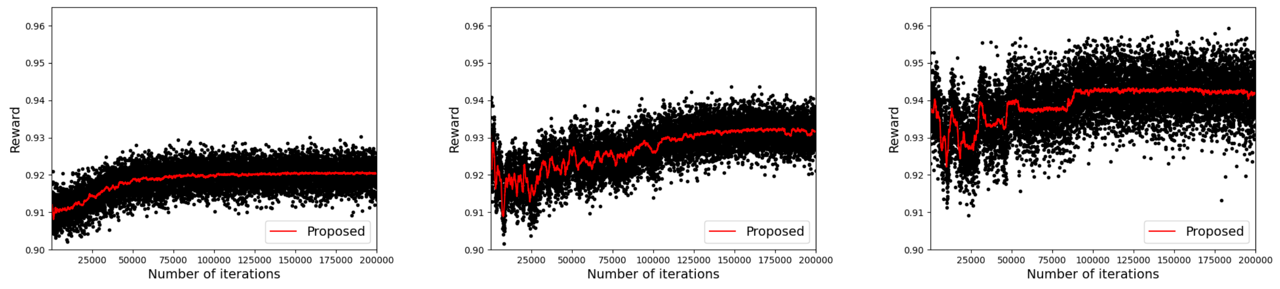
We compared the performance of the proposed routing method against that of the naive method and DDPG-based routing using TDM as a state. The naive method performs packet forwarding with the minimum number of hops, which is equivalent to performing routing using Dijkstra's algorithm in a situation where the weights of all links are the same. In general, the optimal performance in QoS routing is not easily defined. We evaluated the performance of the proposed method by comparing it with the de facto hop-count-based routing method, which forwards packets with the fewest number of hops. DDPG-based routing using TDM is a method to learn link weight allocation by defining the state in DRL as traffic demand between source and destination pairs [40].

Figures 6, 7, and 8 show the average network performance with respect to the number of iterations in a grid topology, a GEANT topology, and an InternetMCI topology, respectively. The reported lines in each figure represent a moving average of 500 time steps. Figures 6(a), 7(a), and 8(a) represent the average end-to-end delay of flows. As shown in the figures, it is seen that the delay of DDPG-based routing methods gradually decreases as the agent experiences iterations. Both DDPG-based methods achieved better performance



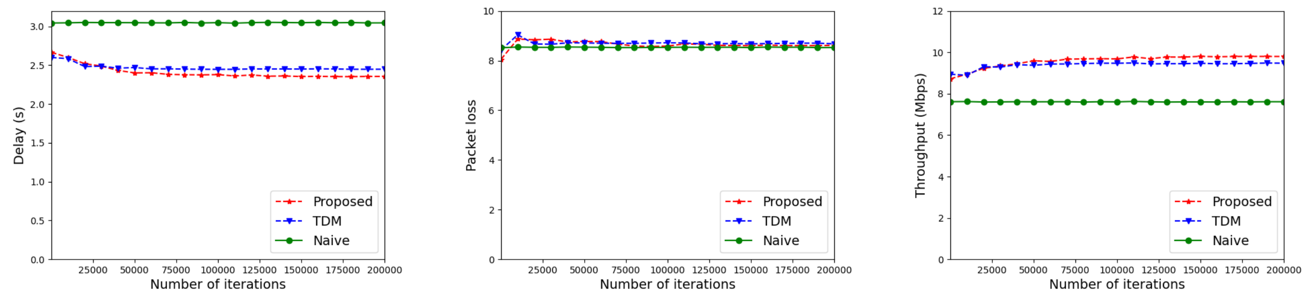
(a) Grid topology with 25 switches (b) GEANT topology with 40 switches (c) InternetMCI topology with 19 switches

FIGURE 4. **Network topologies for performance evaluation.**



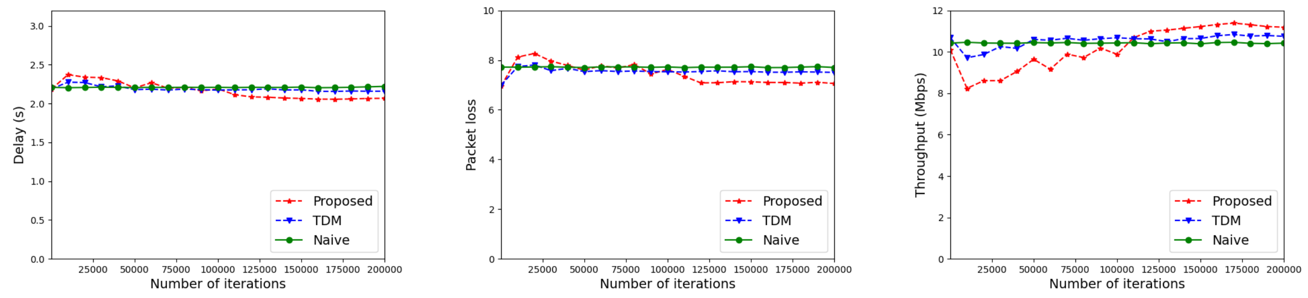
(a) Grid topology (b) GEANT topology (c) InternetMCI topology

FIGURE 5. **Rewards of the agent with respect to the number of time steps.**



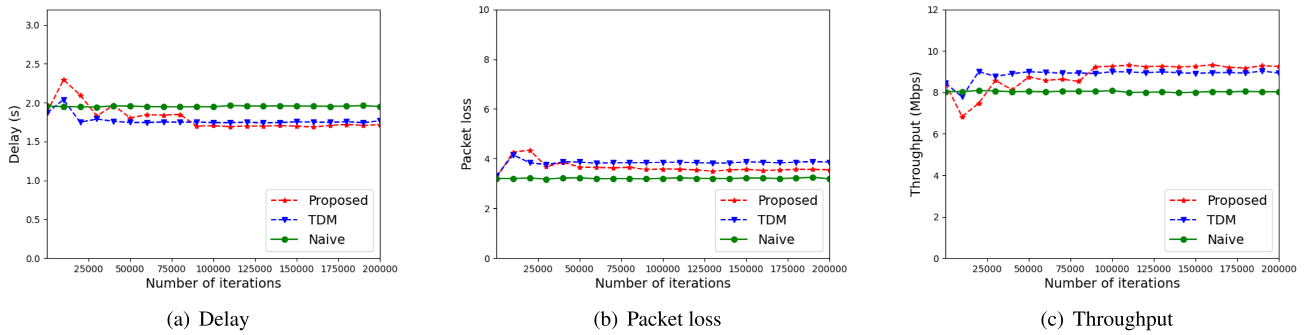
(a) Delay (b) Packet loss (c) Throughput

FIGURE 6. **Average network performance with respect to the number of iterations in grid topology.**



(a) Delay (b) Packet loss (c) Throughput

FIGURE 7. **Average network performance with respect to the number of iterations in GEANT topology.**



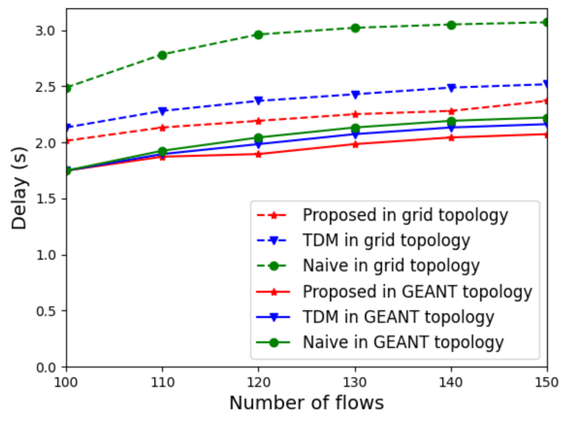
(a) Delay (b) Packet loss (c) Throughput

FIGURE 8. Average network performance with respect to the number of iterations in InternetMCI topology.

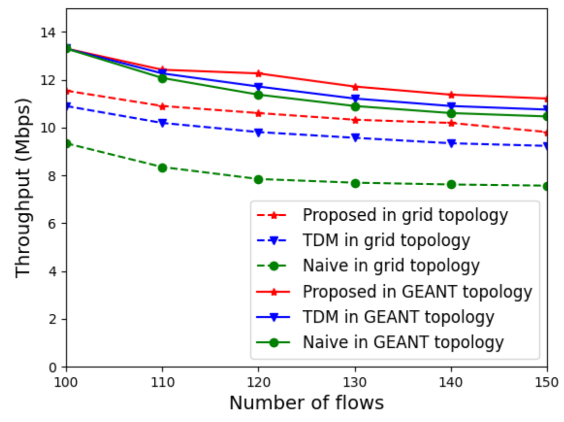
than the naive method as learning gradually progresses, and the proposed method showed better performance than the DDPG-based method using TDM as a state. In a general network where traffic demand can change at every time step like in the simulation, the proposed method can improve learning performance by providing more sophisticated information to the agent by utilizing ATVM as a state.

Figures 6(b), 7(b), and 8(b) show the packet loss for one second as a function of the number of iterations in three topologies. Three topologies exhibit slight increases in packet loss at the start of training, but the packet loss gradually decreases as training progresses. Here, the increase in packet loss was negligible in absolute terms. Since the reward of the proposed method is defined by combining network delay and packet loss metrics, the performance of one index can be slightly worse as shown in Figure 6(b) and 8(b). However, in terms of the overall performance, it can be confirmed that the proposed method is better as shown in Figure 6(c) and 8(c). As learning converged in the case of the GEANT topology, the packet loss of proposed method was smaller than that of the naive method. This is the result of learning in a direction that reduces packet loss if a similar delay occurs even if the number of hops increases. Figures 6(c), 7(c), and 8(c) show the network throughput with respect to the number of iterations in three topologies. In three topologies, as the learning progresses, it is seen that the network throughput increases as the end-to-end delay and packet loss gradually decrease. In addition, when comparing Figure 6(c) and 7(c) with the same number of flows, it is seen that the convergence of throughput is faster in the case of a simple grid topology with a relatively small network size. As shown in the figures, the performance of DDPG-based methods was superior to that of the naive method as learning progressed, and the performance of the proposed method was the best.

Figure 9 depicts the average network performance of the number of flows in a grid topology and a GEANT topology. In this case, the converged network performance of each algorithm is compared by increasing the number of flows by ten from 100 to 150. Figure 9(a) shows the network delay concerning the number of flows. As shown in Figure 9(a), the delays gradually increased as the number of flows increased,



(a) Delay with respect to the number of flows



(b) Throughput with respect to the number of flows

FIGURE 9. Average network performance with respect to the number of flows.

and the delay of the learning-based methods was smaller than the naive method. The delay of the proposed method was the smallest regardless of the number of flows, and the delay in the grid topology with a relatively small network size was larger than that of GEANT topology. When the number of flows is small, the gap between algorithms is reduced.

This is because, when the amount of traffic is small, transmission with the minimum hop is close to optimal, so the performance improvement through learning gradually decreases. Figure 9(b) shows the network throughput concerning the number of flows. As shown in Figure 9(b), as the number of flows increases, the throughput gradually decreases as the delay increases. The throughput of the proposed method was the highest regardless of the number of flows, and the throughput in the GEANT topology with a relatively large network size was larger than that of grid topology.

VI. CONCLUSION

In this paper, we consider an SDN-based routing system in which the DRL agent trains the neural network to select an action for link weight allocation via M/M/1/K queue-based network model. The proposed routing system can solve exploration issues by utilizing the modeled network, especially in the early stages of the learning technique. The DRL agent in the proposed system learns through rewards for actions performed on the modeled network rather than through direct interaction with the data network. Through offline learning with modeled network, the agent can train neural networks without causing the performance degradation of the data network. In the proposed method, the ATVM is employed as a state for DRL learning. The DRL agent updates neural networks to optimize the link weight allocation based on the DDPG algorithm which is one of the off-policy based actor-critic algorithms. Simulation results on three different network topologies demonstrated that the proposed routing method can improve network performance by reducing end-to-end delay and suppressing the increase in the amount of packet loss at switches.

REFERENCES

[1] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3133–3174, 4th Quart., 2019.

[2] Z. Mammeri, "Reinforcement learning based routing in networks: Review and classification of approaches," *IEEE Access*, vol. 7, pp. 55916–55950, 2019.

[3] S. K. Singh, T. Das, and A. Jukan, "A survey on internet multipath routing and provisioning," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2157–2175, 4th Quart., 2015.

[4] O. N. Fundation, "Software-defined networking: The new norm for networks," ONF White Paper, 2012.

[5] S. Ortiz, "Software-defined networking: On the verge of a breakthrough?" *Computer*, vol. 46, no. 7, pp. 10–12, 2013.

[6] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2211–2219.

[7] G. Trimponias, Y. Xiao, X. Wu, H. Xu, and Y. Geng, "Node-constrained traffic engineering: Theory and applications," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1344–1358, Aug. 2019.

[8] P. Sun, Z. Guo, J. Li, Y. Xu, J. Lan, and Y. Hu, "Enabling scalable routing in software-defined networks with deep reinforcement learning on critical nodes," *IEEE/ACM Trans. Netw.*, early access, Dec. 1, 2021, doi: 10.1109/TNET.2021.3126933.

[9] Y.-R. Chen, A. Rezapour, W.-G. Tzeng, and S.-C. Tsai, "RL-routing: An SDN routing algorithm based on deep reinforcement learning," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 4, pp. 3185–3199, Oct. 2020.

[10] P. Sun, Z. Guo, J. Lan, J. Li, Y. Hu, and T. Baker, "ScaleDRL: A scalable deep reinforcement learning approach for traffic engineering in SDN with pinning control," *Comput. Netw.*, vol. 190, May 2021, Art. no. 107891.

[11] J. Zhang, M. Ye, Z. Guo, C.-Y. Yen, and H. J. Chao, "CFR-RL: Traffic engineering with reinforcement learning in SDN," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2249–2259, Oct. 2020.

[12] Q. Fu, E. Sun, K. Meng, M. Li, and Y. Zhang, "Deep Q-learning for routing schemes in SDN-based data center networks," *IEEE Access*, vol. 8, pp. 103491–103499, 2020.

[13] X. Guo, H. Lin, Z. Li, and M. Peng, "Deep-reinforcement-learning-based QoS-aware secure routing for SDN-IoT," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6242–6251, Jul. 2020.

[14] X. Huang, T. Yuan, G. Qiao, and Y. Ren, "Deep reinforcement learning for multimedia traffic control in software defined networking," *IEEE Netw.*, vol. 32, no. 6, pp. 35–41, Nov./Dec. 2018.

[15] T. A. Q. Pham, Y. Hadjadj-Aoul, and A. Outtagarts, "Deep reinforcement learning based QoS-aware routing in knowledge-defined networking," in *Proc. Springer Int. Conf. Heterogeneous Netw. Qual., Rel., Secur. Robustness*, 2018, pp. 14–26.

[16] J. Moy, "OSPF version 2," STD 54, RFC 2328, Apr. 1998, Tech. Rep., doi: 10.17487/RFC2328.

[17] C. Hopps, "Analysis of an equal-cost multi-path algorithm," RFC 2992, Nov. 2000, doi: 10.17487/RFC2992.

[18] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 4, pp. 756–767, May 2002.

[19] A. Sridharan, R. Guerin, and C. Diot, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks," *IEEE/ACM Trans. Netw.*, vol. 13, no. 2, pp. 234–247, Apr. 2005.

[20] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," *IEEE/ACM Trans. Netw.*, vol. 19, no. 6, pp. 1717–1730, Dec. 2011.

[21] N. Michael and A. Tang, "Halo: Hop-by-hop adaptive link-state optimal routing," *IEEE/ACM Trans. Netw.*, vol. 23, no. 6, pp. 1862–1875, Dec. 2015.

[22] D. Kreutz, F. M. V. Ramos, and P. E. Verissimo, and C. E. Rothenberg, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2014.

[23] S. Vissicchio, L. Cittadini, O. Bonaventure, G. G. Xie, and L. Vanbever, "On the co-existence of distributed and centralized routing control-planes," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 469–477.

[24] M. R. Celenlioglu and H. A. Mantar, "An SDN based intra-domain routing and resource management model," in *Proc. IEEE Int. Conf. Cloud Eng.*, Mar. 2015, pp. 347–352.

[25] G.-C. Deng and K. Wang, "An application-aware QoS routing algorithm for SDN-based IoT networking," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2018, pp. 186–191.

[26] P. Rezende, S. Kianpisheh, R. Glitho, and E. Madeira, "An SDN-based framework for routing multi-streams transport traffic over multipath networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–6.

[27] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "Routing or computing? The paradigm shift towards intelligent computer network packet transmission based on deep learning," *IEEE Trans. Comput.*, vol. 66, no. 11, pp. 1946–1960, Nov. 2017.

[28] Y. Zhou, T. Cao, and W. Xiang, "Anypath routing protocol design via Q-learning for underwater sensor networks," *IEEE Internet Things J.*, vol. 8, no. 10, pp. 8173–8190, May 2021.

[29] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 1871–1879.

[30] J. Suárez-Varela, A. Mestres, J. Yu, L. Kuang, H. Feng, A. Cabellos-Aparicio, and P. Barlet-Ros, "Routing in optical transport networks with deep reinforcement learning," *J. Opt. Commun. Netw.*, vol. 11, no. 11, pp. 547–558, Nov. 2019.

[31] J. Zhang, Z. Guo, M. Ye, and H. J. Chao, "SmartEntry: Mitigating routing update overhead with reinforcement learning for traffic engineering," in *Proc. Workshop Netw. Meets AI ML*, Aug. 2020, pp. 1–7.

[32] C. J. Bovy, H. T. Mertodimedjo, G. Hooghiemstra, H. Uijterwaal, and P. V. Mieghem, "Analysis of end-to-end delay measurements in internet," in *Proc. Passive Act. Meas. Workshop-PAM*, 2002.

[33] M. Ye, J. Zhang, Z. Guo, and H. J. Chao, "DATE: Disturbance-aware traffic engineering with reinforcement learning in software-defined networks," in *Proc. IEEE/ACM 29th Int. Symp. Quality Service (IWQOS)*, Jun. 2021, pp. 1–10.

[34] M. Ye, J. Zhang, Z. Guo, and H. J. Chao, "Federated traffic engineering with supervised learning in multi-region networks," in *Proc. IEEE 29th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2021, pp. 1–12.

[35] A. Hagberg, P. Swart, and D. S. Chult, "Exploring network structure, dynamics, and function using NetworkX," in *Proc. Python Sci. Conf. (SciPy)*, 2008, pp. 11–15.

[36] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. (2019). *Stable Baselines3*. GitHub Repository. [Online]. Available: https://github.com/DLR-RM/stable-baselines3

[37] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[39] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the Brownian motion," *Phys. Rev.*, vol. 36, p. 823, Sep. 1930.

[40] C. Yu, J. Lan, Z. Guo, and Y. Hu, "DROM: Optimizing the routing in software-defined networks with deep reinforcement learning," *IEEE Access*, vol. 6, pp. 64533–64539, 2018.

GYUNGMIN KIM (Graduate Student Member, IEEE) received the B.S. degree from the School of Electronic Engineering, Ajou University, Suwon-si, Republic of Korea, in 2016. He is currently pursuing the Ph.D. degree with the School of Electrical Engineering and Computer Science (EECS), Gwangju Institute of Science and Technology (GIST), Gwangju, Republic of Korea. His research interests include artificial intelligence, security for wireless networks, and AI applications for next generation communications.

YOHAN KIM (Member, IEEE) received the B.S. degree from the Information and Communication Engineering, Chungbuk National University, Cheongju-si, Republic of Korea, in 2015, and the Ph.D. degree in electrical engineering and computer science from the Gwangju Institute of Science and Technology (GIST), Gwangju, Republic of Korea, in 2021. He joined the Korea Institute of Science and Technology Information (KISTI), Republic of Korea, in 2022, where he is currently a Senior Researcher with the Division of Data Analysis. His current research interests include big data analysis, artificial intelligence, cloud computing, mobile edge computing, and AI-enabled resource management for next-generation communications.

HYUK LIM (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the School of Electrical Engineering and Computer Science, Seoul National University, Seoul, Republic of Korea, in 1996, 1998, and 2003, respectively. From 2003 to 2006, he was a Postdoctoral Research Associate with the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL, USA. From 2006 to 2021, he was a Professor associated with the AI Graduate School and jointly with the School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology (GIST), Gwangju, Republic of Korea. He served as the Dean for the School of Electrical Engineering and Computer Science and the Director for the GIST Institute for AI. In 2022, he joined the Korea Institute of Energy Technology (KENTECH), Naju-si, Republic of Korea, as a Full Professor. His research interests include artificial intelligence, cyber-security, big data privacy, software-defined networking, and wireless communication systems. He is also conducting active research on AI applications for cybersecurity, networks, and energy systems.