Deep Reinforcement Learning Based Probabilistic Cognitive Routing: An Empirical Study with OMNeT++ and P4

Yixing Wang, Yang Xiao*, Yuqian Song, Jingli Zhou, and Jun Liu
School of Artificial Intelligence, Beijing University of Posts and Telecommunications, Beijing, China
Emails: peterwangyixing@gmail.com, {zackxy, songyuqian, wayzer, liujun}@bupt.edu.cn

***Abstract*—This paper presents an empirical study on deep reinforcement learning (DRL) based probabilistic cognitive routing using the OMNeT++ framework and programming protocol-independent packet processors (P4). The proposed algorithm combines the power of DRL and cognitive routing to achieve efficient and adaptive probabilistic routing in software-defined networking (SDN) environments. To facilitate the research, we develop a dedicated network simulation environment using the OMNeT++ framework and a self-developed SDN platform based on P4. The empirical study highlights the importance of a comprehensive training and validation process in both simulation and real-world SDN environments. Through closed-loop training, the cognitive routing framework provides real-time feedback from the actual network environment to the simulation environment, allowing the agent to excel in real-world network environments. Meanwhile, the results demonstrate that solely testing the algorithm in either environment is inadequate for evaluating its performance accurately.**

***Index Terms*—Deep reinforcement learning, probabilistic cognitive routing, software-defined networking.**

I. Introduction

Software-defined networking (SDN) has emerged as a flexible and programmable network paradigm. Previous studies enhanced traditional routing algorithms with SDN features. Rego *et al.* [1] introduced SDN into OSPF, assessing throughput, packet loss, and delay. Khan *et al.* [2] compared SDN and OSPF, optimizing OSPF convergence times for delay-tolerant networks. Shirmarz *et al.* [3] improved network performance with an adaptive greedy flow routing algorithm in SDN. Wang *et al.* [4] focused on load balancing in SDN-based data centers, minimizing network overhead for improved performance. However, these approaches fall short of fully utilizing network operational knowledge for intelligent routing. Hence, they are being replaced by intelligent routing algorithms leveraging advanced techniques like deep reinforcement learning (DRL) and data-driven strategies. These intelligent algorithms aim to optimize network performance, adapt to dynamic traffic, and enhance overall efficiency.

Recent years have witnessed a surge in interest regarding the implementation of intelligent routing algorithms within SDNs. Researchers have explored various strategies, including the use of DRL techniques to optimize routing decisions. For example, Yao *et al.* [5] introduced NetworkAI, an architecture promoting self-learning control strategies in SDN. This approach was demonstrated to enhance network performance through adaptive routing decisions. Stampa *et al.* [6] leveraged deep neural networks to propose a DRL-based routing optimization technique in SDN, resulting in improved network performance. Lin *et al.* [7] presented a Q-learning-based protocol tailored for SDN networks, aiming to optimize Quality of Service (QoS) by adapting routing decisions using reinforcement learning techniques. Wang *et al.* [8] combined SDN and DRL in the SDCoR protocol, enhancing routing within vehicular ad hoc networks. Sendra *et al.* [9] introduced a distributed routing approach built upon SDN, incorporating a DRL-based algorithm into OSPF to reduce delay and jitter. Xu *et al.* [10] proposed the DRL-TE algorithm, which outperforms traditional routing algorithms as well as other DRL algorithms. Data-driven approaches have also gained attention in SDN routing, with Hope *et al.* [11] proposing the GDDR algorithm based on graph neural networks, leveraging network traffic data to improve routing decisions. The algorithm proposed by Yarin *et al.* [12] effectively addresses TE problem and improves operational efficiency solely through historical data, without the need for explicit future demand predictions. Meanwhile Rusek *el al.* [13] introduced the RBB method, demonstrating the potential to optimize OSPF routing using Graph Neural Networks (GNN). A comprehensive survey of DRL-based routing optimization was conducted by Xiao *et al.* [14], delving into various applications, challenges, and future research directions in utilizing DRL for network traffic management and resource allocation. To address SDN routing challenges, researchers have explored cognitive routing mechanisms. Chen *et al.* [15] introduced RL-Routing, an SDN routing algorithm incorporating DRL for enhanced routing decisions and adaptation to dynamic network conditions. Xiao *et al.* [16] designed an RL-based cognitive routing algorithm for autonomous networks, dynamically optimizing network performance. Extensive simulations demonstrate its effectiveness in improving network throughput, latency, and stability.

While the above studies emphasize the significance of intelligent routing algorithms in SDN, training agents directly in real-world networks is infeasible. A more suitable approach involves training agents in a simulation environment and then applying them to experimental SDN networks. For example, Liu *et al.* [17] proposed an experimental system using OpenFlow in SDN, collecting QoS parameters and forwarding flow tables to SDN switches. Chen *et al.* [18] designed a network architecture comprising forwarding, control, and management

This work was supported by the MoE-CMCC Artificial Intelligence Project under Grant MCM20190701. *(*Corresponding author: Yang Xiao.)*

978-3-903176-59-1 ©2023 IFIP

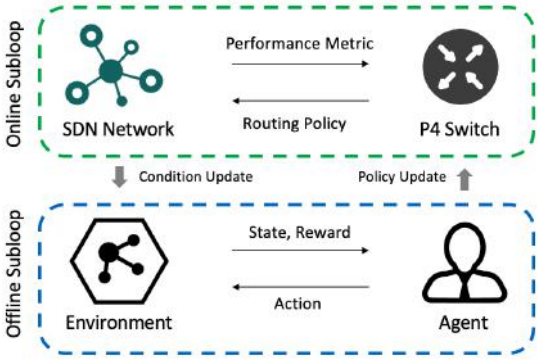


Fig. 1: Process of RL-based cognitive routing.

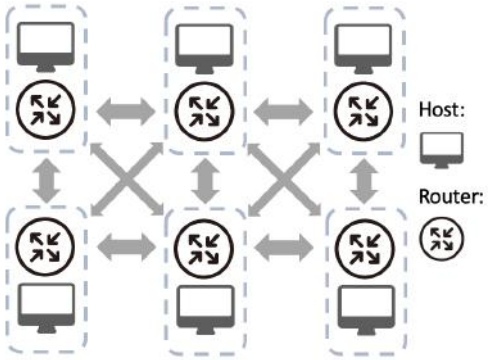


Fig. 2: An example of network topology.

planes. Another work by Casas-Velasco *et al.* [19] presented the RSIR architecture aimed at providing intelligent routing in SDN. However, none of the three mentioned experimental platforms is suitable for cognitive routing.

In this work, we propose a DRL-based probabilistic cognitive routing method. Our contributions are the following. Firstly, we introduce the cognitive routing strategy on SDN to enable adaptive routing. Secondly, we design a novel network routing platform that encompasses both a simulation environment based on OMNeT++ and an experimental environment based on P4. The developed platform facilitates the training and evaluation of routing algorithms in the simulation environment and the SDN environment. Lastly, we demonstrate the feasibility of the probabilistic routing algorithm through experiments conducted in the SDN environment. These contributions collectively advance the understanding and application of intelligent routing techniques in modern network architectures.

The rest of this paper is organized as follows. Section II describes the proposed DRL-based probabilistic cognitive routing method. Section III introduces the designed routing simulation platform. Section IV presents the numerical results.

II. RL-based Cognitive Routing

In this section, we firstly introduce the concept of cognitive routing framework [16]. On this basis, we present the RL representation for packet routing problem and design a novel DRL-based probabilistic routing algorithm.

A. RL-based Cognitive Routing Framework

RL-based cognitive routing is an intelligent routing approach that leverages RL techniques to enable autonomous decision-making in network routing. This section presents the workflow of RL-based cognitive routing and the implementation details.

We propose a closed-loop interaction between the simulation environment and the SDN environment. Fig. 1 illustrates the process of RL-based cognitive routing, which consists of two parallel subloops:

The offline subloop facilitates the cognition acquisition of the cognitive entity. In this subloop, the simulation environment and the agent exchange state, action, and reward.

The online subloop controls the operation of the SDN network without engaging in policy learning processes. Within this subloop, the P4 switch, equipped with embedded computing modules, observes the performance metrics of the network.

By combining these subloops, RL-based cognitive routing merges offline cognition acquisition with real-time decision-making. Our study implements a DRL-based probabilistic routing algorithm within this cognitive framework, enhancing agent performance by collecting and integrating data from the SDN network into the simulation environment.

B. Mathematical Model

1) Network: The network is represented as a directed graph $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ and $\mathcal{E}$ denote finite sets of nodes and transmission links, respectively [20]. Each node consists of two component, i.e., the host and the router that is connected to. The host is responsible for generating traffic demands and receiving traffic, and the router is responsible for packet forwarding. Fig. 2 illustrates an example of the network topology. Each packet originates from a source node s, and is routed to a sink node d. A path $p_{s,d}$ is defined as a walk in the graph $G = (\mathcal{N}, \mathcal{E})$ that connects the source to the destination through a sequence of routers. $bw(e_{i,j})$ represent the bandwidth of link $e_{i,j}$, where i, j are indexes of the two endpoints of an edge, respectively.

2) Routing: The objective of packet routing is to forward each packet through multiple routers to the destination. The routers follow a first-in-first-out (FIFO) queuing. Router n continuously forwards the head-of-line (HOL) packet to its neighboring node v until the packet reaches its destination. The TE problem can be formally defined as determining a set of paths that efficiently forward packets from the sources to the sinks according to a certain optimization objective. Our goal is to minimize the average delivery time T calculated by $\sum_{p \in P} T_p / K$, where P represents the set of packets and K represents the number of packets in P. In addition, the packet loss rate, given by K_{loss}/K, where K_{loss} represents the number of lost packets, also needs to be reduced. In practice, network traffic exhibits temporal variations. Therefore, our objectives should also adapt to dynamic traffic.

C. DRL-based Probabilistic Routing

On basis of the mathematical model of network and routing, we present a DRL-based probabilistic routing algorithm. Typically, an RL problem is modeled by $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ denotes the set of states, $\mathcal{A}$ denotes the set of actions, $\mathcal{P}$ denotes the state transition probability, and $\mathcal{R}$ denotes the reward function. Our definitions of $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{R}$ are introduced as follows:

1) State: The state space is defined as

$$S \subseteq \mathbb{R}^{n \times n+m}, \tag{1}$$

where n represents the number of switches, and m represents the hours of a day. The state at time interval t is defined as

$$s_t = [f_1, f_2], \tag{2}$$

which encompasses the traffic information at t. The link utilization feature is obtained by

$$f_{1,i} = \left\{ \frac{flow_{t,ij}/bw_{ij}}{\triangle t} \right\}, \tag{3}$$

which implies the link utilization referring to the ratio of the traffic on each output port of a switch to the bandwidth of the link, as shown in equation (3). Here, i represents the switch identifier, and j represents the various output ports of the switch, where the number of output ports may vary for each switch. The link utilization contains both the absolute value information of the traffic and the remaining capacity information of the links. By observing the link utilization, the agent can gain insights into the load condition and perform load-balancing scheduling for the traffic. The time zone feature is defined as

$$f_2 = [0, 0, \ldots, 1, 1, \ldots, 0, 0], \tag{4}$$

which is a 24-dimensional vector representing 24 hours. We divide a day into 24 non-overlapping time intervals. The entry in the vector corresponding to the current time interval is set to 1, while others are set to 0.

2) Action: The action space is expressed as

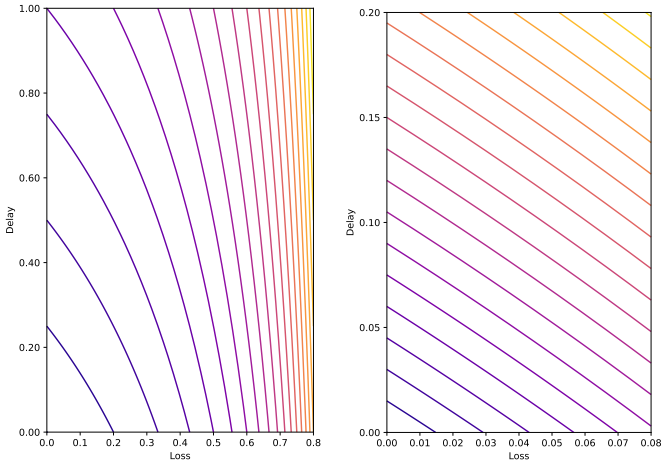
$$A \subseteq \mathbb{R}^{e \times 2}, \tag{5}$$

where e represents the number of links. The action at t is defined as

$$\vec{a}_t = \{w_{1,jt}, w_{2,jt}\}, \tag{6}$$

which consists of the directed weight for each link. Each switch forwards the received packets based on the weight proportion, using the ratio as a probability. In which, $w_{1,jt}$ represents the forward weights metric and $w_{2,jt}$ represents the reversed direction weights metric.

3) Reward: We propose two alternative approaches for different situations. One approach balances average delay and packet loss, while the other emphasizes delay. The reason for this design is that probabilistic routing strategies often result in longer path selections. To enable the agent to find a feasible path that is relatively shorter, we need to emphasize control over delay. We firstly define the delay utility as



(a) Preliminary stage (b) Later stage

Fig. 3: The contour plot of overall delay and packet loss rate.

$$du = \frac{(\sum_{p \in P} T_p/K)^{1-\mu_d}}{1-\mu_d}, \tag{7}$$

where P represents the sets of packets, K represents the number of packets, T_p represents the propagation delay of packet p, and μ_d represents the coefficient of delay.

$$lr = \sum_{t \in T} K_{loss,t}/K_t. \tag{8}$$

$$r_1 = \mu_r \times e^{-du - \mu_l \times lr}. \tag{9}$$

In equation (8), K_t represents the total number of packets sent within t, $K_{loss,t}$ represents the number of packets lost during that period, and T represents the total duration. In equation (9), we define the first type of reward function, where μ_r, μ_l are coefficient of the reward and loss rate, respectively.

$$r_2 = -u_t = -(\sum_{p \in P_t} T_p/K + timeout \times \frac{lr_t}{1-lr_t}). \tag{10}$$

Additionally, we design another type of reward function. The action reward is set as the negative value of the overall delay in equation (10). The overall delay can comprehensively reflect the impact of end-to-end average delay and packet loss from the perspective of users. By utilizing queuing theory, we propose the process of calculating the comprehensive queuing delay (referred to as overall delay) as follows:

$$u = \sum_{p \in P} T_p/K \times (1-lr) + (timeout + u) \times lr, \tag{11}$$

Assuming that users need to retransmit after the timeout, the average delay for users can be obtained from equation (11). By transformation, we also have

$$u = \sum_{p \in P} T_p/K + timeout \times \frac{lr}{1-lr}. \tag{12}$$

By setting the timeout to 1 second and utilizing equation (10), we generate contour lines (as depicted in Fig. 3) that illustrate the relationship between overall delay and packet loss. In both figures, overall delay and packet loss exhibit a monotonic increasing relationship. When overall delay and packet loss are high, the impact of packet loss is significant, prompting the agent to optimize the packet loss rate. Conversely, the agent prioritizes optimizing overall delay.

Two reward functions accommodate differences between simulation and SDN environments. The first, apt for high packet loss scenarios, swiftly reduces losses and stabilizes policies but has limits on delay reduction. The second, designed for low packet loss, prioritizes delay reduction and seeks optimal routing paths.

D. Learning Algorithm

We employ the proximal policy optimization (PPO) [21] to train our policy network. The training process involves iterative updates of the policy parameters using minibatch gradient descent. A batch of experiences is collected using the current policy π_θ for each training iteration. During each optimization epoch, the advantages $A(s_t, a_t)$ are computed using value function estimates

$$\mathcal{A}(s_t, a_t) = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t), \tag{13}$$

where γ is the discount factor of action value $r_{t'}$ and $V_\phi(s_t)$ is computed through a value network, which consists of a neural network with the same structure as the policy network but with different parameters ϕ. Then, $A(s_t, a_t)$ is used to approximate the discounted rewards from the current state to the terminal state [22].

The target policy parameters θ are updated by minimizing the PPO objective function, which consists of a policy loss $\mathcal{L}_p$, a value function loss $\mathcal{L}_v$, and an entropy regularization loss $\mathcal{L}_e$. Let $r_t(\theta)$ denotes the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, \tag{14}$$

where θ is the policy parameters and θ_{old} is the policy parameters before the update. The surrogate objective is

$$\mathcal{L}_p = \mathbb{E}_t[min(r_t(\theta)\mathcal{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\mathcal{A}_t)], \tag{15}$$

where ϵ is a hyperparameter, the second term $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)$ involving the clipping of the probability ratio, modifies the surrogate objective by restricting the range of the probability ratio. The value function loss is a squared-error loss defined as

$$\mathcal{L}_v = (V_\phi(s_t) - V_t^{targ})^2. \tag{16}$$

The entropy regularization loss is defined as

$$\mathcal{L}_e = S[\pi_\theta](s_t), \tag{17}$$

where S denotes an entropy bonus. Combining these terms, we obtain the following objective

$$\mathcal{L}_t^{p+v+e} = \hat{\mathbb{E}}_t[\mathcal{L}_p - c_1\mathcal{L}_v + c_2\mathcal{L}_e], \tag{18}$$

Algorithm 1: Proximal Policy Optimization Training

Input: Initialize policy network π with random weights θ, value network V with random weights ϕ;
Output: Policy network π;

```
Initialize a buffer to store trajectories D ← ∅;
for t = 1 to T do
    Collect flow information tf from SDN environment;
    for i = 1 to N do
        Select action with action noise;
        Collect a batch of experiences using the current policy π_θ;
        Compute advantages A(s_t, a_t) using value function estimates;
        for j = 1 to K do
            Update policy parameters θ using minibatch gradient descent;
            Compute policy loss L_p;
            Compute value function loss L_v;
            Compute entropy regularization loss L_e;
            Compute total loss: L_t^{p+v+e} = Ê_t[L_p − c_1 L_v + c_2 L_e];
            Update θ using gradient descent: θ ← θ − α∇_θ L;
        end
        Update action noise;
        Update the policy network parameters θ to old policy network θ_old;
    end
end
```

where c_1, c_2 are coefficients. The policy parameters are updated using gradient descent: $\theta \leftarrow \theta - \alpha\nabla\theta\mathcal{L}$. This process are repeated for multiple training iterations until convergence.

III. Design of Simulation and SDN Platform

In this section, we describe the implementation and technical details of our self-developed routing validation platform.

A. Simulation Environment

Our simulation environment is developed based on OMNeT++, which allows distributing DRL-based probabilistic routing weights, collecting environmental states, and offering a customizable UDP packet traffic generation module that updates the simulation environment based on the SDN environment. To enable interaction between the agent and the simulation environment, we employ ZeroMQ as an interface tool, ensuring the sequential reception of states and rewards.

B. SDN Platform

We implement the network environment based on the P4 protocol. Fig. 4 shows the architecture of the whole system. The SDN platform is divided into three parts: the control plane, the data plane, and the management plane.

1) Control plane: The control plane consists of the P4 controller and the north API server. The controller handles information distribution, including marker bits, multicast settings, and link weights for the routing algorithm. Meanwhile, the north API server manages configuration data, offers network status feedback, provides interfaces for agents to access

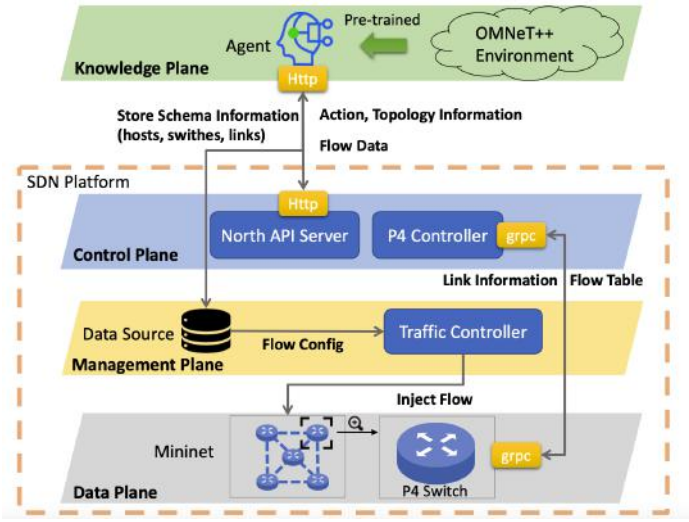


Fig. 4: Architecture of the network routing validation system.

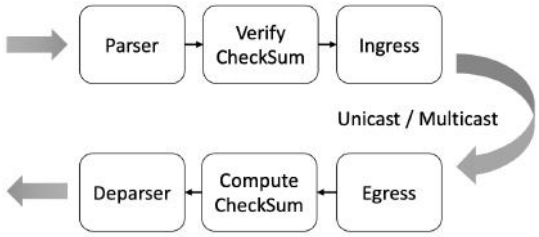


Fig. 5: Processing flow of V1Model.

network data and control operations, and takes care of cleanup and reporting duties for the experimental network.

2) Management plane: The management plane includes a traffic controller component that controls the traffic generators in the network by issuing instructions to all traffic generators. The specific process involves the controller first setting the configuration information for each traffic generator and then waiting for all controllers to receive and process the instructions before issuing a command to activate the new configuration. In this part, we implement event publishing and subscription using Redis.

3) Data plane: The data plane consists of Mininet and P4 switches. We start by defining the network topology with Python scripts, converting config files into Mininet-compatible data structures. Then, we invoke the Python interface to create hosts, switches, and links. Finally, we use the P4 BMV2 software switch and extend Mininet's switch class with P4RuntimeSwitch.

P4 algorithms rely on a specific P4 model, typically V1Model, which defines the switch's processing pipeline (see Fig. 5).

Our switch selects the next hop based on link weights, achieved through two custom data segments at the data link layer: a custom routing header and a flooding broadcast header. During unpacking, we examine the Ethernet header's type field to determine whether to parse the custom routing header and check for the presence of the flooding header for analysis.

TABLE I: Agent Design

Agent Design	Reward Funtion	Activation Function
Agent1	r_1	softmax
Agent2	r_2	ReLU
Agent3	r_2	softmax

TABLE II: Structure of ActorCritic Neural Network

Layer	Nerual Network Structure	Activation Function
Input	$state\_dim \times 256 \times 128 \times 128 \times 64$	tanh
Actor	$64 \times 32 \times 32 \times action\_dim$	tanh
Critic	$64 \times 32 \times 32 \times 1$	tanh

IV. Numerical Result

A. Experimental Environment Setup

This section presents the experimental results of the DRL-based probabilistic routing algorithm under the RL-based cognitive routing framework. NSFNet is used as the network topology that consists of 13 nodes and 20 bidirectional links, as shown in Fig. 7. We deploy the experimental environment within container that consists an 8-core Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz, 32GB of memory and a GeForce RTX 2080 GPU.

B. OMNeT++ Training Experiments

In the training phase of the simulation environment, we design three different agents by combining different reward functions and activation functions, as shown in TABLE I, and conduct 144 hours of training for each agent. The two reward functions in the table are described in section II, focusing on delay and packet loss respectively.

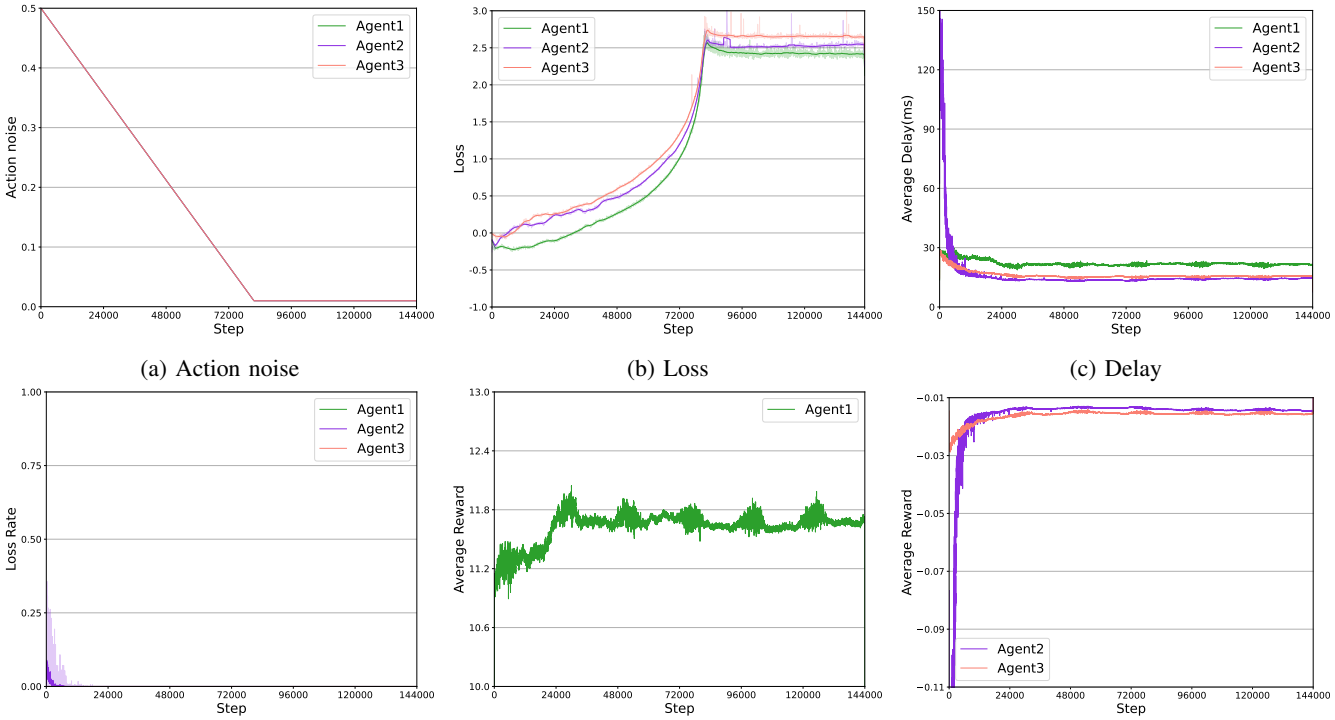
The structure of the ActorCritic neural network primarily consists of three main components: the input layer, the actor layer, and the critic layer. The neural network structure of each component is elaborated in TABLE II. Afterwards, we can construct the policy and value networks with these components in the PPO algorithm.

Regarding the training process, we gradually decrease the action noise. In the first 2/3 of the training duration, the action noise linearly decreases, and in the remaining 1/3 of the training duration, it remains at the lowest value.

Fig. 6 shows training results for the three agents in the OMNeT++ simulation environment, focusing on end-to-end delay and packet loss rate. All agents achieve zero packet loss, demonstrating excellent performance. Agent2 excels in delay reduction, thanks to its reward function prioritizing delay reduction in a lossless simulation environment. Fig. 6f and Fig. 6e illustrate the learning processes of all agents, with fluctuations during varying traffic intensity.

C. SDN Confirmatory Experiments

We collect 24-hour flow rate data from a real network with 13 nodes, where 4 nodes receive packets generated at random intervals following a Poisson distribution. We then assess trained agents in the SDN environment, running simulations



(a) Action noise (b) Loss (c) Delay

(d) Loss rate (e) Reward (r_1) (f) Reward (r_2)

Fig. 6: Training processes on OMNeT++.

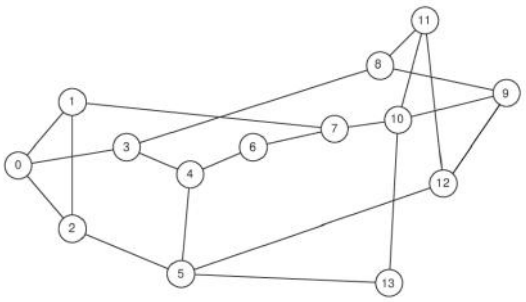


Fig. 7: Topology of NSFNet.

for 72 hours. Fig. 8b and Fig. 8c show test results. Surprisingly, Agent2, previously the top performer in simulations, proves ineffective in the SDN setup. Agent3 consistently outperforms Agent1 in delay and packet loss, aligning with their performance in simulations. This suggests that training in the simulation environment partially reflects agents' SDN performance.

The gap stems from differences between the simulation and SDN environments. Agent2's ReLU activation function, unlike the others' softmax, affects link availability. Softmax ensures values between $(0, 1)$ while ReLU sets negative weights to 0. Due to statistical variability, discrepancies exist between the environments. SDN network validation is crucial to fully reveal agent design flaws and demonstrate expected experiment results.

Finally, we validate our DRL-based cognitive routing framework's impact on agent performance in the SDN environment, as shown in Fig. 8. In Fig. 8d, cognitive routing slightly improves delay over the original agent. In Fig. 8e, it significantly reduces packet loss, effectively adapting to traffic fluctuations and mitigating loss increases with a 50% reduction, reaching below 0.01. Feedback from the SDN environment enhances the agent's performance.

V. Conclusion

We proposed a DRL-based cognitive routing method, designed a simulation and validation platform, and showed the importance of experiments in both environments. Interaction between SDN and simulation improved agent performance. Future work involves adding SDN metrics, maximizing P4's potential, and analyzing the simulation-to-SDN transition to support RL-based routing in real-world applications.

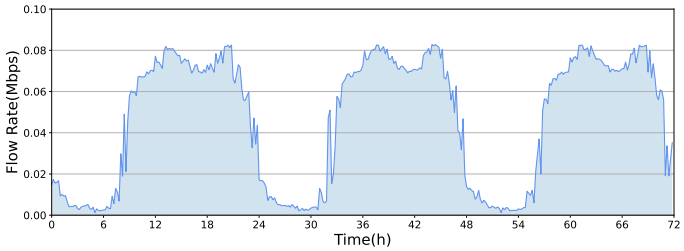
References

[1] A. Rego, S. Sendra, J. M. Jimenez, and J. Lloret, "Ospf routing protocol performance in software defined networks," in *2017 Fourth International Conference on Software Defined Systems (SDS)*. IEEE, 2017, pp. 131–136.

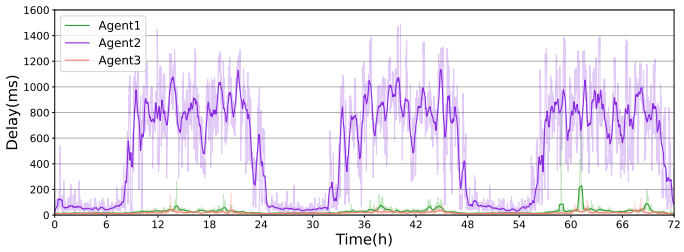
[2] A. A. Khan, M. Hussain, M. Zafrullah, and M. S. Zia, "A convergence time optimization paradigm for ospf based networks through sdn spf protocol computer communications and networks (ccn)/delay tolerant networks," in *Proceedings of the International Conference on Future Networks and Distributed Systems*, 2017, pp. 1–6.

[3] A. Shirmarz and A. Ghaffari, "An adaptive greedy flow routing algorithm for performance improvement in software-defined network," *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, vol. 33, no. 1, p. e2676, 2020.

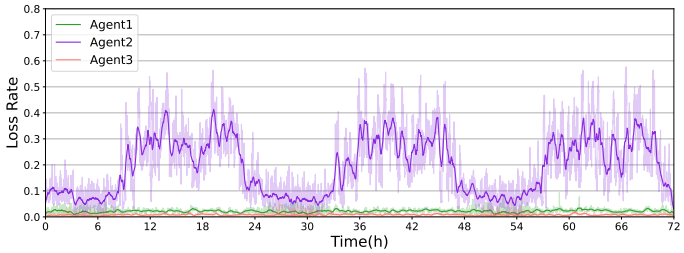
[4] Y.-C. Wang and S.-Y. You, "An efficient route management framework for load balance and overhead reduction in sdn-based data center networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1422–1434, 2018.



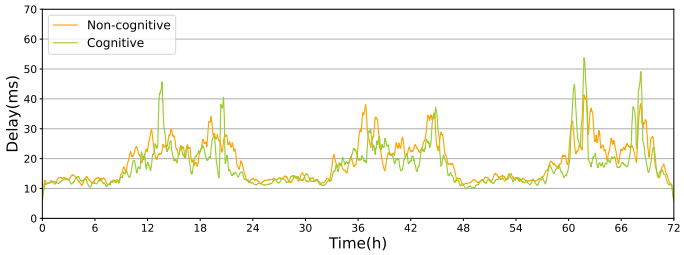
(a) Plot of flow rate over time



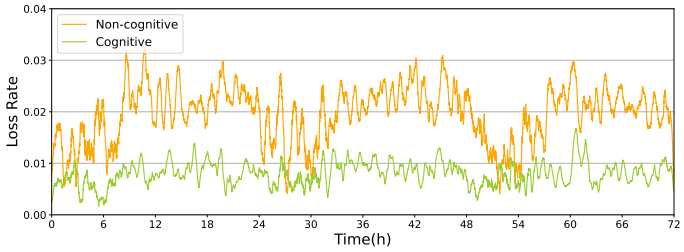
(b) Plot of latency over time



(c) Plot of loss rate over time



(d) Plot of latency over time



(e) Plot of loss rate over time

Fig. 8: SDN confirmatory experiments.

[5] H. Yao, T. Mai, X. Xu, P. Zhang, M. Li, and Y. Liu, "NetworkAI: An intelligent network architecture for self-learning control strategies in software defined networks," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4319–4327, 2018.

[6] G. Stampa, M. Arias, D. Sánchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *arXiv preprint arXiv:1709.07080*, 2017.

[7] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "QoS-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach," in *2016 IEEE International Conference on Services Computing (SCC)*. IEEE, 2016, pp. 25–33.

[8] C. Wang, L. Zhang, Z. Li, and C. Jiang, "SDCoR: software defined cognitive routing for internet of vehicles," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3513–3520, 2018.

[9] S. Sendra, A. Rego, J. Lloret, J. M. Jimenez, and O. Romero, "Including artificial intelligence in a routing protocol using software defined networks," in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2017, pp. 670–674.

[10] 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 *IEEE INFOCOM 2018-IEEE conference on computer communications*. IEEE, 2018, pp. 1871–1879.

[11] O. Hope and E. Yoneki, "GDDR: GNN-based data-driven routing," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 517–527.

[12] Y. Perry, F. V. Frujeri, C. Hoch, S. Kandula, I. Menache, M. Schapira, and A. Tamar, "DOTE: Rethinking (predictive) WAN traffic engineering," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 1557–1581.

[13] K. Rusek, P. Almasan, J. Suárez-Varela, P. Chołda, P. Barlet-Ros, and A. Cabellos-Aparicio, "Fast traffic engineering by gradient descent with learned differentiable routing," in *2022 18th International Conference on Network and Service Management (CNSM)*. IEEE, 2022, pp. 359–363.

[14] Y. Xiao, J. Liu, J. Wu, and N. Ansari, "Leveraging deep reinforcement learning for traffic engineering: A survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2064–2097, 2021.

[15] Y.-R. Chen, A. Rezapour, W.-G. Tzeng, and S.-C. Tsai, "RL-routing: An SDN routing algorithm based on deep reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 3185–3199, 2020.

[16] Y. Xiao, J. Li, J. Wu, and J. Liu, "On design and implementation of reinforcement learning based cognitive routing for autonomous networks," *IEEE Communications Letters*, vol. 27, no. 1, pp. 205–209, 2022.

[17] W.-x. Liu, J. Cai, Q. C. Chen, and Y. Wang, "DRL-R: Deep reinforcement learning approach for intelligent routing in software-defined data-center networks," *Journal of Network and Computer Applications*, vol. 177, p. 102865, 2021.

[18] J. Chen, Y. Wang, J. Ou, C. Fan, X. Lu, C. Liao, X. Huang, and H. Zhang, "Albrl: Automatic load-balancing architecture based on reinforcement learning in software-defined networking," *Wireless Communications and Mobile Computing*, vol. 2022, pp. 1–17, 2022.

[19] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. da Fonseca, "Intelligent routing based on reinforcement learning for software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 870–881, 2020.

[20] X. You, X. Li, Y. Xu, H. Feng, J. Zhao, and H. Yan, "Toward packet routing with fully distributed multiagent deep reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 2, pp. 855–868, 2020.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[22] Y. Xiao, Y. Song, and J. Liu, "Multi-agent deep reinforcement learning based resource allocation for ultra-reliable low-latency internet of controllable things," *IEEE Transactions on Wireless Communications*, vol. 22, no. 8, pp. 5414–5430, 2023.