Grace: Toward Routing in Dynamic Network Environments With Graph Embedding

Wenting Wei, *Member, IEEE*, Huaxi Gu, Liying Fu, and Baochun Li, *Fellow, IEEE*

***Abstract*—Recent efforts have explored adaptive routing via deep reinforcement learning (DRL) techniques without handcrafted parameter engineering. Intrinsically, routing decision-making is essentially a process used to find a subgraph in a graph-structured network. However, previous works seldom took topological relationships into consideration when providing adaptive routing algorithms, causing them to suffer from suboptimal routes in dynamic network environments involving both varying traffic loads and burst traffic. In this paper, we present *Grace*, a novel graph embedding-based Deep Reinforcement Learning framework tailored for distributed routing algorithm optimization within the Software-Defined Networking (SDN) paradigm. Specifically, *Grace* leverages graph embedding to translate graph-structured entities into low-dimensional vectors, thereby enabling multiple DRL agents to learn optimal routing paths under dynamic network environments. Unfortunately, training multiple agents encounters inherent challenges in complicated and dynamic network scenarios. In response, we design an adaptive incremental training method for *Grace* that makes the model adapt to task complexity in a gradual manner, while speeding up its retraining efforts when environments change. To further accelerate convergence, we integrate intrinsic curiosity into *Grace* to tackle large environments with sparse rewards. Extensive experiments conducted on two real-world topologies demonstrate the rationality and effectiveness of *Grace*, and the results show throughput improvements of up to 40.1% compared to other state-of-the-art DRL routing algorithms under bursty traffic conditions.**

***Index Terms*—Distributed routing algorithm, deep reinforcement learning, graph embedding.**

I. Introduction

ROUTING is a fundamental problem in communication networks, and its algorithm design affects the performance of a large number of network flows sharing the same network topologies, especially as the volume of network traffic scales up. Unfortunately, conventional routing mechanisms [1], [2], [3] relied heavily on fixed rules, and were unable to keep up with the dynamic network environments with time-varying and potentially bursty traffic demands over short time scales [4]. It is crucial to design a new array of routing algorithms that are able to adapt swiftly to such dynamic environments.

One of the most important features of such a new array of adaptive routing algorithms is their reaction to temporary congestion events [5]. As burst traffic leads to network congestion and packet losses, these adaptive routing algorithms can react quickly by rerouting congested flows to a different route. To design such routing algorithms, recent works in the literature have explored the possibility of using machine learning [6], [7], and in particular deep reinforcement learning (DRL) techniques [8], [9], [10]. Intuitively, the advantage of using DRL techniques lies in its ability to learn from data and experience, which can be quite easily extracted from interactions with the network environment at run-time. With sufficient experience, such DRL-based routing algorithms can learn how to adapt to temporary congestion events in dynamic environments.

In the recent literature, Casas-Velasco et al. [8] designed a DRL-based routing algorithm, called RSIR, that took into account network link-state information to make routing decisions. Similarly, Liu et al. [9] proposed a DRL-based online routing algorithm, called DRL-OR, and incorporated a reward function that took latency, throughput, and packet loss ratio into account, and was able to accommodate flows with different quality of service (QoS) requirements. Despite the general advantages of using DRL techniques for adaptive routing algorithm designs, one of the pitfalls of existing works is that they did not consider the network topology when making routing decisions, which may lead to suboptimal routes in dynamic network environments.

Indeed, making routing decisions characterized by a graph structure is fundamentally a process of finding a subgraph in a network topology. In the specific context of routing in dynamic environments with bursty traffic volumes and temporary congestion events, it is even more important to consider the network topology when computing the best possible alternative routes for rerouting congested flows. The optimality of such rerouting decisions will affect the overall performance when all of the flows in the network are considered.

In this paper, we propose *Grace*, a new DRL-based adaptive routing algorithm in dynamic network environments. *Grace* continues to use DRL techniques to exploit its power to learn from experience; yet it incorporates the knowledge of network topologies when making routing decisions with DRL agents. At a high level, *Grace* makes extensive use of *graph embedding*, which has emerged as a graph-representation

Received 29 October 2023; revised 28 June 2024 and 7 January 2025; accepted 1 July 2025; approved by IEEE TRANSACTIONS ON NETWORKING Editor O. Yagan. Date of publication 11 July 2025; date of current version 18 December 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62102302, in part by the National Key Research and Development Program of China under Grant 2025YFB3003200, and in part by the Science and Technology Innovation Team Program of Shaanxi Province under Grant 2025RSCXTD-004. *(Corresponding author: Huaxi Gu.)*

Wenting Wei, Huaxi Gu, and Liying Fu are with the State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China (e-mail: hxgu@xidian.edu.cn).

Baochun Li is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 1A1, Canada.

Digital Object Identifier 10.1109/TON.2025.3586010

learning technique that maps graph-structured entities into low-dimensional vectors [11] without burying topological information. Due to its excellent ability to represent entities and relations as numerical vectors, *Grace* uses graph embedding to understand and infer the potential rules among network elements for routing design. Highlights of our original contributions of this paper are as follows.

- We introduce a novel graph embedding-based Deep Reinforcement Learning (DRL) framework tailored for routing optimization within the Software-Defined Networking (SDN) paradigm. This framework harnesses the explicit knowledge embedded in the network topology, empowering Grace to exploit topological insights among network entities and their interconnection for dynamic routing optimization.
- We integrate the Intrinsic Curiosity Module (ICM) [12] into *Grace*'s framework, endowing its agents with the agility to swiftly adapt to the vicissitudes of network conditions. This innovative feature significantly bolsters the agents' performance in complex, sparse-reward environments, ensuring robust and effective decision-making under challenging circumstances.
- We propose an adaptive incremental training method for *Grace*, which evolves initial features into a spectrum of sophisticated and nuanced ones. This evolution begins with foundational tasks and incrementally scales in complexity, facilitating a gradual adaptation to fluctuating network scenarios and accelerating retraining processes in response to environmental shifts.
- We conducted extensive experiments using `ns.py` [13], a Python-based discrete event network simulator, under two real-world network topologies. The results indicate that Grace surpasses conventional baselines in dynamic settings, as evidenced by superior performance in delay, throughput, and packet loss ratio metrics.

II. Problem Formulation and Architecture

A. Problem Formulation

We consider a communication network with N nodes, modeled as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the set of nodes and links respectively. Thus, the network comprises V nodes and E edges, where $V = |\mathcal{V}|$ and $E = |\mathcal{E}|$. The network accommodates F flows, indexed as $f = f_1, f_2, \ldots, f_F$. Each flow f in the network originates from a specific source router and terminates at a distinct destination router, represented by $\{Src(\mathrm{f}), Dst(\mathrm{f}) | Src(\mathrm{f}), Dst(\mathrm{f}) \in \mathcal{V}\}$. Once the appropriate path $\boldsymbol{p}^f$ is computed, we can determine how packets on a flow f can be forwarded within the network from the source to the destination node. Additionally, every egress port of each router has a finite first-in-first-out (FIFO) buffer that can be used to reduce packet loss. It is worth noting that as the network becomes more congested due to multiple active flows and bursty traffic, queuing delays and packet losses will accumulate.

An example scenario is shown in Fig. 1. f_1 is routed from B to G by following the path $\boldsymbol{p}^{f_1} = \{B, A, G\}$, and f_2 follows the forwarding path $\boldsymbol{p}^{f_2} = \{A, G\}$. If $Src(f_3)$ is A, and $Dst(f_3)$ is F, there are two possible paths, i.e., $\boldsymbol{p}^{f_3} = \{A, G, F\}$ and $\boldsymbol{p}^{f_3} = \{A, C, E, F\}$. Although the path $\{A, G, F\}$ seems to be shorter than $\{A, C, E, F\}$, the link (A, G) is heavily loaded, and the packets may be discarded when the buffer is full. On the contrary, path $\{A, C, E, F\}$ has no congestion, making it have smaller delay and packet losses. In this paper, we focus on the flow-based routing, where traffic flows are delivered from source nodes to their destinations while minimizing delays and packet losses, and maximizing throughput as much as possible.

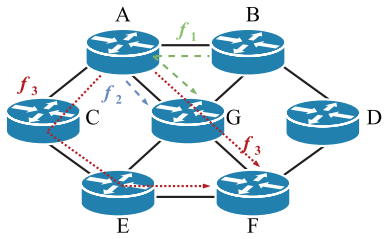


Fig. 1. An motivation example.

In communication networks, the current state is an aggregation of past routing states, so routing decisions can be represented as Markov Decision Process (MDPs), which facilitate optimization via dynamic programming and RL techniques [14]. In each time interval, the agent observes the network state and makes a routing decision. After executing traffic forwarding, the agent receives the reward value fed back from the network and optimizes its parameters. In this paper, *Grace* implements a distributed routing scheme, where each node can be regarded as a carrier of an independent RL agent.

Thus, routing decisions can be represented as multi-agent extensions of MDP (MAMDP) [15], where agents select their own routing paths following the MAMDP to optimize the network performance. Specifically, an MAMDP is defined by $\{\mathcal{S}, \{\mathcal{A}^i\}_{i \in N}, \mathcal{P}, \mathcal{R}\}$ [16], [17], where $\mathcal{S}$ and $\mathcal{A}^i$ are the finite state space and action space of agent i. $\mathbb{A} = \prod_i^N \mathcal{A}^i$ denotes the joint action space of all agents for MAMDP. $\mathcal{P}$ is the state transition probability, and $\mathcal{R}$ is the global reward. Our goal is to maximize the cumulative reward, which is defined by $\sum_{t=0}^{\infty} r_t(\boldsymbol{s}_t, \boldsymbol{a}_t)$. At each time step t, agents first observe the network environment s_t and make routing decisions $a_t \in \mathbb{A}$ according to the current policy, i.e., calculate routing paths for each flow generated at time step t. After executing a_t, the network state turns to s_{t+1} and feeds the reward value r_t back to these agents.

In detail, the agent i corresponding to the source node makes an action a_t^i to select the best next hop, and the following agent of next hop takes a similar way until the path of the flow request reaches the destination node. It is worth noting that if the routing path has a loop while selecting next hop, the routing calculation will be terminated immediately, and the path will be replaced by the shortest path. According to this paradigm, the forwarding efficiency can be greatly increased, and the decision time is reduced compared to that of selecting the next node for every packet. For example, at time step t, agent A performs an action $\boldsymbol{a}_t^A \in \mathcal{A}^A$ to select the best next hop for f_3 according to the current state $s_t \in \mathcal{S}$, which is node

C. Agent *C* uses a similar approach and reaches the destination node *F*. Thus far, the routing path of f_3 is $\boldsymbol{p}^{f_3} = \{A, C, E, F\}$.

B. RL Formulation for Routing

The DRL agents observe the network state periodically and adjust their policies according to the reward derived from the network environment. To achieve our goals of minimizing the end-to-end packet delivery time and network congestion, the main essentials of the RL model are designed as follows:

1) State Space: A state is a signal that characterizes information extracted from the network environment, and we denote the state space as $\mathcal{S}$. The agents of *Grace* take the network state as input and output a routing action. Traffic transmission requirements, which originate from random sources and are directed towards random destinations, are generated at each time step. Thus, the network state $\boldsymbol{s}_t \in \mathcal{S}$ at any time step *t* is denoted as

$$\boldsymbol{s}_t = \{\{\boldsymbol{c}_t^{f_1}, \boldsymbol{d}_t^{f_1}, \boldsymbol{z}_t\}, \{\boldsymbol{c}_t^{f_2}, \boldsymbol{d}_t^{f_2}, \boldsymbol{z}_t\}, \ldots, \{\boldsymbol{c}_t^{f_{F_t}}, \boldsymbol{d}_t^{f_{F_t}}, \boldsymbol{z}_t\}\}, \tag{1}$$

where $\mathbf{c}_t^{f_i}$ is the source node and $\mathbf{d}_t^{f_i}$ is its destination node of flow f_i, $i = 1, 2, \ldots, F_t$; both $\boldsymbol{c}_t^{f_i}$ and $\boldsymbol{d}_t^{f_i}$ are represented with one-hot coding. F_t is the total number of flows generated at time step *t*. $\boldsymbol{z}_t$ is the representation of the current network topology obtained using the graph embedding method.

2) Action Space: At each time step, the agents compute routes for flows on a hop-by-hop basis. Thus, the action of each agent is its egress port, and the joint action space $\mathbb{A}$ for flow requests contains their forwarding paths. For any time step *t*, $\boldsymbol{a}_t \in \mathbb{A}$ is denoted as

$$\boldsymbol{a}_t = \{\boldsymbol{p}_t^{f_1}, \boldsymbol{p}_t^{f_2}, \ldots, \boldsymbol{p}_t^{f_{F_t}}\}, \tag{2}$$

where $\mathbf{p}_t^{f_i}$ is the next hop node from $\mathbf{c}_t^{f_i}$ to $\mathbf{d}_t^{f_i}$ of flow f_i. Agents learn to route packets from their sources to their destinations in a hop-by-hop manner through continuous iteration with the goal of minimizing delay and network congestion.

3) Reward: A reward function $r(\cdot)$ is employed to provide a steering direction for making better routing decisions. Each router in the forwarding plane collects its own received packets and queue information, which is periodically uploaded to the AI plane's agent. In each step, an agent selects an action according to the current policy and global reward, aiming at maximizing the cumulative reward. For *Grace*, the basic reward function r_{b_t} at any time step *t* is composed as

$$r_{b_t} = -\omega_d \times log(D_t) + \omega_t \times log(T_t) - \omega_q \times log(Q_t), \tag{3}$$

where D_t is the average delay of all received packets which is recorded at the destination router when the packet is successfully received. T_t is the network throughput which is counted as the average number of packets successfully received per unit of time per router. Q_t is the average queue length which is periodically counted by the average number of queued packets across the router's ports. And $\omega_d, \omega_t, \omega_q$ are the weights of the above three respectively. These metrics provide a comprehensive view of the network's performance and are essential for the reward calculations performed by the agents. Each agent perform a reward value calculation based on this information. To enhance relative fairness across all elements, we additionally apply the logarithmic function $log(\cdot)$ for these metrics to mitigate the influence of varying magnitudes.

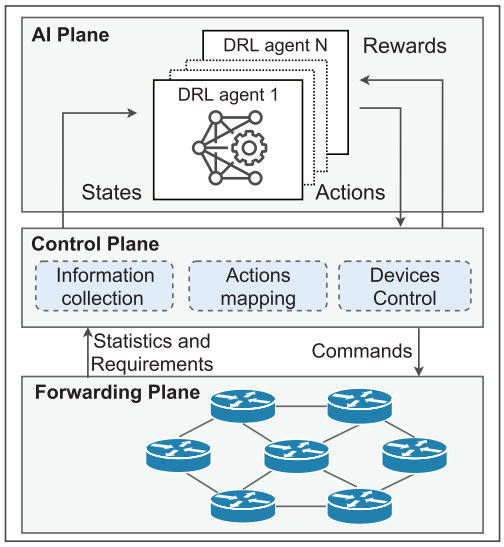


Fig. 2. *Grace* architecture.

C. Architectural Design

As shown in Fig. 2, the proposed *Grace* is deployed under the software-defined networking (SDN) architecture [18], including the forwarding plane, control plane and AI plane. Specifically, the control plane collects network requirements and information periodically, and the DRL-agents in the AI plane calculate forwarding paths. Afterward, routes are sent to the forwarding plane, and the traffic is transmitted accordingly. The detailed components are described as follows:

1) Forwarding Plane: This plane is composed of forwarding devices (i.e., routers) and links. When traffic transmission requirements arrive at the network, these devices are responsible for traffic forwarding based on flow tables, which are updated according to calculated routes from the AI plane.

2) Control Plane: This plane builds a connection between the AI plane and the forwarding plane. On the one hand, the control plane translates the collected global information into the network state by collecting network statistics and requirements from the forwarding plane and feeds them to the AI plane. On the other hand, it also dynamically updates the forwarding rules for the forwarding device by mapping the actions of the AI plane.

3) AI Plane: This plane is responsible for learning forwarding policies and generating routing paths by interacting with the network environment through the control plane. Specifically, this plane contains *N* DRL agents (corresponding to *N* routers in the network) working in a distributed manner. These agents obtain the network states from the control plane and calculate the forwarding path for the traffic transmission requirements.

III. Algorithm

In this section, we propose a graph embedding-enabled distributed DRL framework for flow-based routing, where a

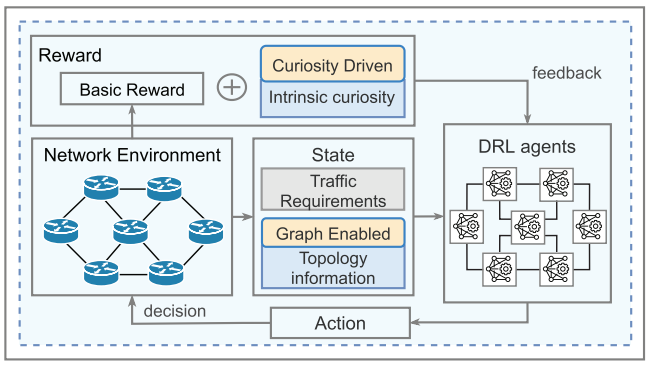


Fig. 3. DRL-based framework.

graph embedding method is used to represent the network topology information. Since the *Grace* model is characterized by its sparse rewards, the intrinsic curiosity is implemented to accelerate the training convergence by reshaping the reward. The overall *Grace* framework is shown in Fig. 3, and its main components are described in detail as follows:

A. DRL Scheme

We use distributed DRL to train the proposed routing model. Each agent interacts with the network environment and selects the next hop to form forwarding paths for traffic flows. After the network state changes and the reward is fed back from the environment, these agents update the neural networks with the proximal policy optimization algorithm.

B. Graph Enabled Module

To increase the decision-making power of the proposed DRL agents in a dynamic network, we employ a graph embedding method to represent the network topology (i.e., the forwarding devices and their connections) and add these representations to the state space.

C. Curiosity-Driven Module

To resolve the sparse reward problem during training and increase the learning speed, a curiosity-driven method is exploited to guide DRL agents toward the desired decision through routing steps. Specifically, this method is implemented to guide agents in their exploration by adding an additional signal to the basic reward signal naturally received from the environment. In such cases, curiosity can serve as an intrinsic reward signal to enable the agent to explore and learn effectively.

D. Mapping Entity Relationships by Graph Embedding

Since computer networks are fundamentally represented as graphs, their network topologies can be also represented as such [19]. However, due to the neglect of topological attributes, traditional DRL-based routing algorithms may encounter challenges in changing environments. Recently, graph analysis has attracted increasing attention. As shown in Fig. 4, graph embedding [11] is the process of mapping

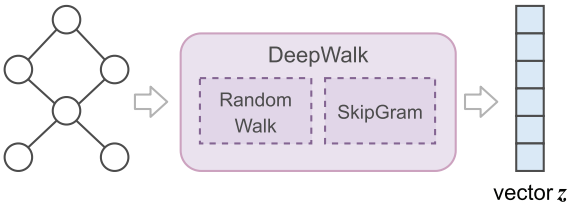


Fig. 4. Graph embedding method.

high-dimensional graph-structured data into a low-dimensional vector space, where each vertex v_i is represented as a low-dimensional vector $\boldsymbol{z}_i$.

To enhance the decision-making capabilities of the proposed Deep Reinforcement Learning (DRL) agents within a dynamic network setting, we have integrated a graph embedding technique in Grace' framework. Specifically, the graph embedding approach is utilized to abstract the network entities and their inherent relational dynamics into a representational format. This method then seamlessly integrates these refined representations into the state space, enhancing the agents' capacity to make informed and strategic decisions in response to the evolving network conditions.

Furthermore, the graph embedding module is equipped to swiftly identify and process changes in the network topology, providing the agent with an updated low-dimensional vector in a timely manner. As a result, the agent can strategically minimize the forwarding of traffic data to nodes that exhibit a backlog of queued packets and have a limited number of output links during the routing path selection. This nuanced approach is highly beneficial for easing network congestion and, consequently, for boosting network throughput.

Specifically, a novel approach for learning the social representations of vertices called *DeepWalk* [20] is used to fulfill the graph embedding process. It uses information obtained from truncated random walks to learn latent representations by treating walks as the equivalent of sentences. Specifically, *DeepWalk* takes a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as input and produces a node representation vector $\boldsymbol{z}$. The algorithm consists of two main components, namely, *RandomWalk* and *SkipGram*. *RandomWalk* is a stochastic process that selects a random start node v_s and specifies the walk length l. After several random walks from v_s, we can extract a series of sequences of vertices W_{v_s} from the input graph $\mathcal{G}$. *SkipGram* is a natural language processing (NLP) method that maximizes the concurrence probability among the words that appear within a window w in a sentence. With the help of *SkipGram*, walks generated by *RandomWalk* can be thought of as short sentences and phrases composed of words, all of which can be considered a large corpus. The objective of *SkipGram* is to estimate the likelihood of vertex v_i considering all the previous vertices in the random walk, i.e.,

$$\Pr = (v_i | (v_1, v_2, \ldots, v_{i-1})). \tag{4}$$

The goal of *DeepWalk* is to learn a latent representation, so we introduce a mapping function $M : v \in \mathcal{V} \rightarrow \mathbb{R}^{|\mathcal{V}| \times d}$, where $d \ll |\mathcal{V}|$. Then, the problem is to estimate the likelihood

$$\Pr = (v_i | (M(v_1), M(v_2), \ldots, M(v_{i-1}))). \tag{5}$$

In terms of vertex representation modeling, this turns into an optimization problem:

$$\underset{M}{\text{minimize}} \ -log\Pr(\{v_{i-w},\ldots,v_{i+w}\}/v_i \mid M(v_i)). \quad (6)$$

Algorithm 1 DeepWalk

Input: graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$, embedding size d, walk length l, walks per vertex ϑ, window size w.
Output: vertex representations $M: v\in\mathcal{V}\rightarrow\mathbb{R}^{|V|\times d}$.

```
for i = 1 to ϑ do
    for v_s ∈ V do
        W_{v_s} = RandomWalk(G, v_s, l);
        SkipGram(M, W_{v_s}, w);
    end for
end for
```

Algorithm 2 SkipGram

```
for each v_i ∈ W_{v_s} do
    for each u_k ∈ W_{v_s}[i − w : i + w] do
        J(M) = −logPr(u_k|M(v_i));
        M = M − ξ ∗ ∂J/∂M;
    end for
end for
```

Solving optimization problems establishes representations that capture the common similarity of local graph structures. Finally, the representation M is updated to maximize the cooccurrence probability $\Pr(\{v_{i-w},\ldots,v_{i+w}\}/v_i|M(v_i)) = \prod_{j=i-w,j\neq i}^{i+w}\Pr(v_j|M(v_i))$. The details of the *DeepWalk* method are shown in Algorithm 1, and the detailed *SkipGram* is shown in Algorithm 2. It is worth noting that the original design of graph embedding was for recommender systems and social network prediction, which is complex in predicting models. Our main goal is not to predict random topology changes, so we simplify the hidden layers in the *SkipGram* model according to the network scale to reduce the computational cost for network problems. Due to *DeepWalk*, agents can infer the relationships among the entities in a network topology by adding the representations of network nodes to the state space, which facilitates bypassing bottleneck nodes

E. Curiosity-Driven Explorations via Intrinsic Curiosity

RL is an approach that allows an agent to optimize its behavior while interacting with its environment. After several iterations, the agent can learn the ideal solution to the given problem by maximizing the cumulative reward. However, in complex tasks or tasks with large state-action spaces, the rewards collected from the environment are extremely sparse. In such cases, curiosity can serve as an intrinsic reward signal to enable the agent to explore and learn effectively.

Generally, an agent in state s_t interacts with the environment by executing an action a_t sampled from its current policy and ends up in state s_{t+1}. The policy is trained to optimize the sum of the reward. In this paper, we implement the ICM [12] to compose the reward

$$r = r_c + r_b, \quad (7)$$

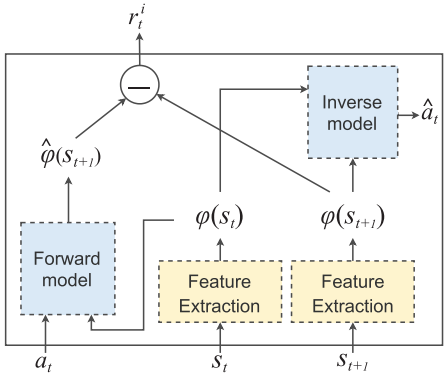


Fig. 5. Intrinsic Curiosity Module.

where r_b is the base reward provided by the environment, and the curiosity-based intrinsic reward signal r_c^i of agent i is generated by the ICM.

In Fig. 5, the ICM is used to learn feature representations and provide a good intrinsic reward signal. It first encodes the states s_t and s_{t+1} into features $\varphi(s_t)$ and $\varphi(s_{t+1})$, respectively, to predict a_t. The forward model takes a_t and $\varphi(s_t)$ as inputs and predicts the feature representation $\hat{\varphi}(s_{t+1})$ of s_{t+1}. The prediction error in the feature space is used as the curiosity-based intrinsic reward signal. Specifically, the proposed feature space can be learned by training a deep neural network with two submodules. The first module encodes the raw state s_t into a feature vector $\phi(s_t)$, and the second module takes the feature encodings $\varphi(s_t)$ and $\varphi(s_{t+1})$ of two consequent states as inputs and predicts the action a_t taken by the agent to move from state s_t to s_{t+1}. Training this neural network amounts to the learning function g is defined as

$$\hat{a} = g(\varphi(s_t),\varphi(s_{t+1});\theta_I), \quad (8)$$

where $\hat{a}$ is the predicted estimate of action a_t, and the neural network parameters θ_I are trained to optimize

$$\min_{\theta_I} L_I(\hat{a}_t, a_t) \quad (9)$$

where L_I is the loss function that measures the discrepancy between the predicted actions and actual actions. In addition to the inverse model, according to the ICM, we train another neural network that takes a_t and $\varphi(s_t)$ as inputs to predict the feature encoding of $\hat{\varphi}(s_{t+1})$

$$\hat{\varphi}(s_{t+1}) = h(\varphi(s_t), a_t;\theta_F), \quad (10)$$

where h is the learning function, and the neural network parameters θ_F are optimized by minimizing L_F:

$$L_F(\hat{\varphi}(s_{t+1}),\varphi(s_{t+1})) = \frac{1}{2}\|\hat{\varphi}(s_{t+1})-\varphi(s_{t+1})\|_2^2. \quad (11)$$

Finally, the intrinsic reward signal $r_c = r_t^i$ is computed as

$$r_{c_t}^i = \frac{\eta}{2}\|\hat{\varphi}(s_{t+1})-\varphi(s_{t+1})\|_2^2, \quad (12)$$

where $\eta > 0$ is a scaling factor.

F. DRL Agent Training Algorithm

In this work, we adopt the proximal policy optimization (PPO) algorithm [21] to train the agents under each training environment. PPO is a new family of policy gradient methods for RL that alternate between sampling data through interaction with the environment and optimizing an objective function using stochastic gradient ascent. Inspired by the work in multiagent PPO (MAPPO) [16], we can extend the single-agent RL approach to multiagent RL (MARL).

Due to the independence of each agent, the partial derivative of $J(\theta)$ with respect to the neural network parameter of each agent θ^i for the multiagent system becomes

$$\nabla_{\theta^i} J(\theta) = \mathbb{E}_{s\sim d_\theta, a\sim\pi_\theta}[A^\pi(s,a)\nabla_\theta \log \pi^i_{\theta^i}(a^i|s^i)]. \quad (13)$$

We assume that the Markov chain is irreducible and aperiodic under any π_θ and possesses a stationary distribution d_θ over state s. In a training epoch of MAPPO, trajectories $\mathcal{D}$ can be obtained from intercation with the network environment, including the history of states, actions, and rewards. A trajectory is denoted as $\tau = \{\boldsymbol{s}_0, \boldsymbol{a}_0, r_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{T-1}, \boldsymbol{a}_{T-1}, r_{T-1}, \boldsymbol{s}_T\}$, where T is total time step. Based on the PPO algorithm, the objective for agent i is defined as

$$J_i^{CLIP}(\theta^i) = \mathbb{E}_{\boldsymbol{s}_t,\boldsymbol{a}_t}[\min(\rho_t^i(\theta^i)\hat{A}(\boldsymbol{s}_t,\boldsymbol{a}_t), g(\epsilon, \hat{A}(\boldsymbol{s}_t,\boldsymbol{a}_t)))], \quad (14)$$

$$\rho_t^i(\theta^i) = \frac{\pi_{\theta^i}(a_t^i|s_t^i)}{\pi_{\theta^i_{old}}(a_t^i|s_t^i)}, \quad (15)$$

$$g(\epsilon, \hat{A}) = \begin{cases} (1+\epsilon)\hat{A}, & \hat{A} \geq 0 \\ (1-\epsilon)\hat{A}, & \hat{A} < 0 \end{cases} \quad (16)$$

where θ^i is the policy parameter that agent i uses to sample a trajectory τ and ϵ is a hyperparameter. Given a set of trajectories $\mathcal{D}$, the policy parameter of agent i is updated by

$$\theta^i_{new} = \arg\max_{\theta^i} \frac{1}{|\mathcal{D}|T} \sum_{\tau\in\mathcal{D}} \sum_{t=0}^{T-1} J_i^{CLIP}(\theta^i). \quad (17)$$

In practice, θ^i can be updated iteratively via stochastic gradient ascent:

$$\theta^i \leftarrow \theta^i + \alpha \nabla_{\theta^i} J_i^{CLIP}(\theta^i). \quad (18)$$

where α is the learning rate of the policy.

We use the centralized value function by evaluating the states and actions of all agents. As the state value function $V_\phi(\boldsymbol{s}) = \mathbb{E}_\pi[\hat{G}_t|\boldsymbol{s_t} = \boldsymbol{s}]$, the global critic is updated by regression on the mean squared error:

$$\begin{aligned} \phi_{new} &= \arg\min_{\phi} L(\phi) \\ &= \arg\min \phi \frac{1}{|\mathcal{D}|T} \sum_{\tau\in\mathcal{D}} \sum_{t=0}^{T-1} (V_\phi(\boldsymbol{s}) - \hat{G}_t)^2. \end{aligned} \quad (19)$$

Similar to θ^i, ϕ can be updated iteratively:

$$\phi \leftarrow \phi + \beta \nabla_{\phi_i} L(\phi), \quad (20)$$

where β is the learning rate of the critic.

The training process is shown in Algorithm 3.

Algorithm 3 Training Algorithm of Grace

Input: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, source of flows $\boldsymbol{c}_t$, destination of flows $\boldsymbol{d}_t$.

Output: forwarding paths set $\boldsymbol{a}_t$.

```
Initialize policy parameter θ_0^i for each agent i, initialize global value function parameter φ_0;
for step ∈ {1, 2, ..., T} do
    Obtain node representation z_t = DeepWalk(G) using Algorithm 1;
    Integrate current state s_t = {c_t, d_t, z_t};
    Select and execute action a_t with policy π_k = π(θ_k);
    Calculate reward r_t = r_{b_t} + r_{c_t} according to Equation (7);
    Observe next state s_{t+1} after performing a_t;
    if T%l == 0 then
        for k ∈ {1, 2, ..., K} do
            Collect set of trajectories D_k = {s_0, a_0, r_0, ..., s_{T-1}, a_{T-1}, r_{T-1}, s_T} using π_{θ_k};
            Compute advantage estimates Â_t based on value function V_{φ_k};
            Update the policy parameter θ_{k+1}^i of each agent i by Equation (18);
            Fit the global value function by regression on mean-squared error;
            Update the critic parameter φ_{k+1} by Equation (20) for each state s_t;
        end for
    end if
end for
```

G. Incremental Training Mode

When the given network environments or transmission tasks are complex, it is difficult for a routing algorithm to converge to the desired target with limited training overhead (such as the training time and dataset). Inspired by the pattern of human learning [22], we can train agents starting from a simple task and iteratively increase the complexity of the training scenarios.

Aiming to address the low efficiency and weak generalization abilities of DRL agents in a heavy-load network scenario with varying volumes of traffic, in this paper, an incremental training mode [23] is implemented for making routing decisions. During the training procedure, the traffic requirements and application scenes become successively more complex based on the previous phase. As shown in Fig. 6, in the beginning, we deploy a single-flow transmission task under a light-load environment for pretraining. After performing sufficient training, the decision model (called the basic model) attains the primary decision-making capacity for routing traffic. Subsequently, we transfer the model into environments with different complexities and perform incremental training with fewer steps. This causes the model (called the improved model) to have better decision efficiency.

In such a case, the basic model is first trained in an environment with multiple concurrent flows, and we can obtain a model with better robustness. By utilizing another

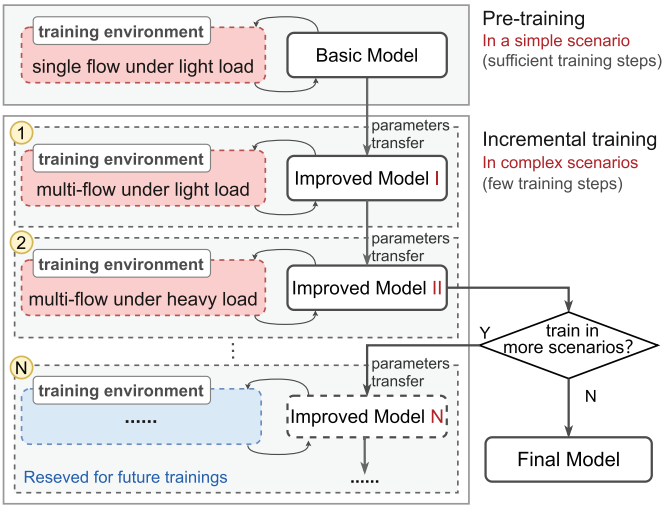


Fig. 6. Incremental training mode.

round of parameter transfer, we perform training again with a heavier traffic load and multiple active flows to enhance the adaptability of the DRL agents. We further reserve the interface for future incremental training; thus, only a few fine-tuning training steps are required when deploying the model in more complicated environments. It is worth noting that we only need a small time cost to perform incremental training after obtaining a sufficiently trained basic model. This requires much less time than directly training the model in an intrinsic scenario. By building each variation upon the previous variation, the DRL agents can efficiently adapt to task complexity in an incremental manner [24] with less overhead.

IV. Evaluation

A. Experimental Setup

We implemented *Grace* in a discrete-time network simulation platform (`ns.py` [13]) and evaluated its performance in comparison with that of several baseline methods on a PC with a 6-core i7 CPU @3.20 GHz and 16 GB of RAM. Our simulation runs were performed on two well-known network topologies from The Internet Topology Zoo [25], i.e., the 14-node NSFnet and 25-node AT&T. Each node in the topology has a limited buffer to temporarily store the received packets. A packet is transmitted over the communication link when it becomes the first packet in line [26]. If the specific buffer size at one node is full, the newly arriving packets are dropped. The simulation time was measured in milliseconds, and the transmission delay of each link was set to 1.0 ms.

The flow requests arrive at the network with randomly selected source-destination pairs, and the sending rate is fixed to 300 Kbps. However, Internet traffic exhibits multifaceted burstiness on short time scales, which has an impact on the QoS [27]. To fully imitate traffic changes in the network, we implemented static and bursty environments in the simulation. The static environment refers to a stable scenario in which the number of concurrent flows and the sending rates never change. We further induced a short-time traffic burst in the environment (referred to as bursty) with a doubled number of flows, a doubled sending rate, and a doubled packet size in the middle of testing. This burst lasts for 50 slots.

During the learning process, we set $\omega_d = 0.5$, $\omega_t = 0.4$, and $\omega_q = 0.1$ for the basic reward $\mathcal{R}_b$ to balance the delay, throughput and packet loss. The number of training steps used by *Grace* was set to 30000 in the pretraining stage and to 5000 in each incremental training environment. We also determined good settings for the graph embedding module: the embedding size $d = V$, the walk length $l = 2 \times V$, and the number of walks per vertex $w = V$.

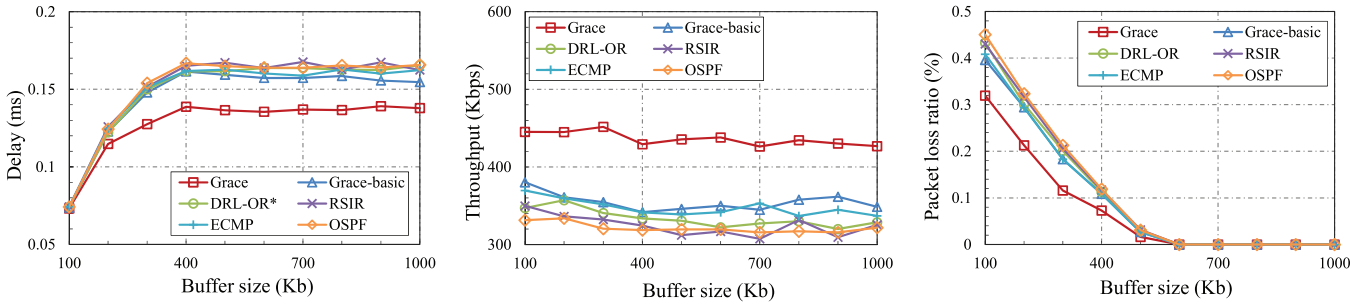
We compared *Grace* with the following three baselines.

- Open shortest path first routing (OSPF): Every packet is delivered by the shortest path algorithm, which selects the path with the fewest number of hops.
- Equal-cost multipath routing (ECMP): ECMP distributes data across multiple equal-cost paths with equal probabilities. It is worth noting that multipath routing algorithms have to handle the potential issue of packet reordering, or out-of-order of packets. Given that our paper's primary focus is on network routing, we have chosen to exclude the evaluation of packet reordering in our assessment of ECMP's performance.
- RL and software-defined intelligent routing (RSIR): RSIR [8] trains an RL-based routing agent with Q-learning, which takes several link states into account when making routing decisions.
- DRL-based online routing (DRL-OR*): DRL-OR* is derived from an online DRL-based routing algorithm called DRL-OR [9], which learns appropriate policies for different types of requirements. Without loss of generality, DRL-OR* does not use the link bandwidth in the state space during the evaluation since it can be modeled by its upstream port in `ns.py` [13].
- *Grace* without incremental training (Grace-basic): Grace-basic makes the agents learn complex transmission tasks directly, which is implemented to demonstrate the efficiency of the proposed incremental training process.

To comprehensively demonstrate the performance of *Grace*, we determined several performance metrics, including the delay, throughput, and packet loss ratio. During the evaluation process, we first focused on the effect of the buffer size and determined the most proper value of this parameter for running the following tests. Then, we separately evaluated the performance achieved under three different traffic load levels in the static and bursty environments mentioned above. Finally, we evaluated the performance by considering different numbers of flows and bursty environments.

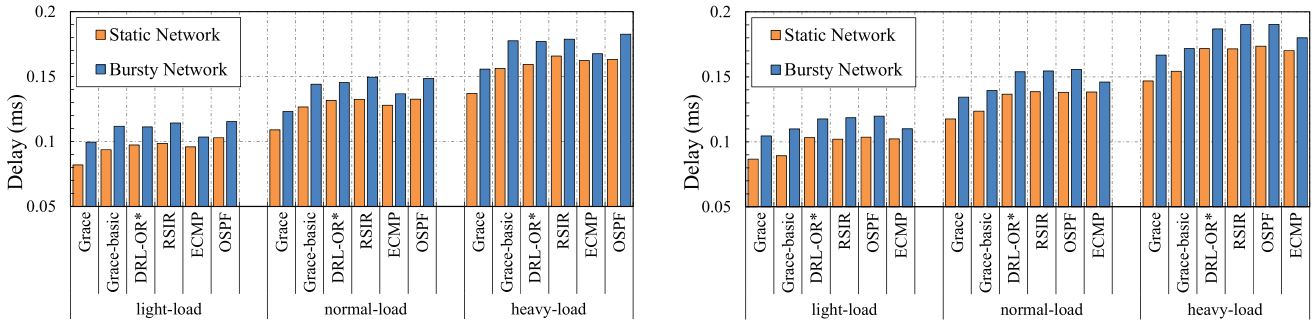
B. Impact of the Buffer Size on Performance

In a real network scenario, a traffic burst occurs when a large amount of data must be sent in a short time, resulting in packet loss caused by router congestion. Router buffers are used to mitigate packet loss by absorbing transient traffic bursts when the routers cannot forward them at those moments. When the buffer size is large, a significant increase in latency (buffer bloat) is generated. Conversely, a very small buffer size increases the packet loss during the congestion period [28]. Therefore, the buffer size is a critical parameter.



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

Fig. 7. Impact of the buffer size.



(a) NSFnet. (b) AT&T.

Fig. 8. Delay comparison over the traffic load.

To choose a suitable buffer size for running the subsequent tests, we explored the impact of the buffer size on the routing performance. For this test, we varied the buffer size from 100 Kb to 1000 Kb under the static NSFnet (the impact of the buffer size is similar to that under AT&T). During this test, the number of concurrent flows was set to 3, and the traffic load was set to a heavy load.

From Fig. 7, we find that the performance gap between *Grace* and the baselines increases with the expansion of the buffer. When the buffer size is small, all these methods exhibit similar delays, and a large number of packets are discarded by the nodes. When the buffer size exceeds 500 Kb, the delay tends to remain constant, and the throughput begins to decrease due to the larger number of stored packets. To avoid an extremely poor queuing delay or packet loss in the following evaluation, we fixed the buffer size to 500 Kb, according to the presented results.

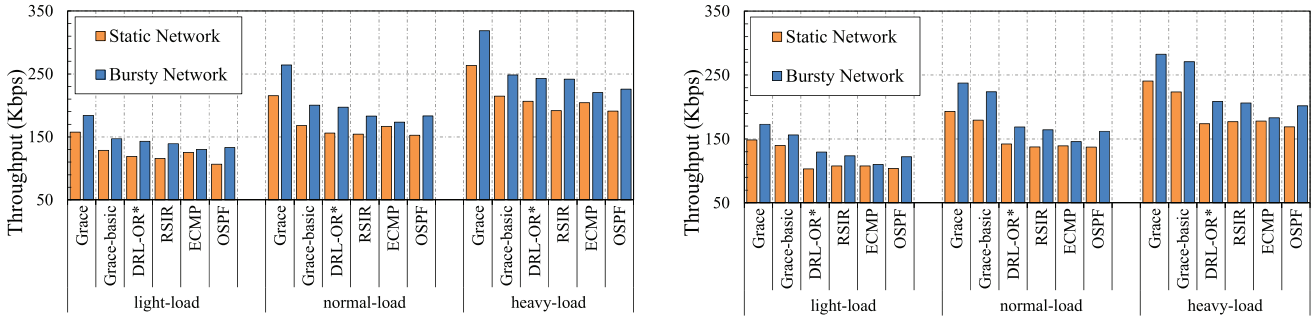
C. Performance Under Varying Traffic Loads

Similar to DRL-OR [9], we evaluated the performance of the tested methods under three traffic load levels: a light-load scenario in which the flow duration is 1.5 s; a normal-load scenario in which the flow duration is 2 s; and a heavy-load scenario in which the flow duration is 2.5 s. During the traffic load tests, the buffer size is fixed at 500 Kb, and the number of concurrent flows is 3. To further evaluate the robustness and adaptability of these routing algorithms, we ran the tests in both static and bursty environments.

1) Delay: Fig. 8 shows the delay comparison results obtained in the static and bursty environments. As can be seen, *Grace* outperforms all baseline methods. It is not surprising to see that all these methods lead to different kinds of performance degradation in the bursty environment, but *Grace* undeniably delivers superior performance. Moreover, the delay measured under the AT&T topology is larger due to the longer propagation delay caused by longer forwarding paths. Overall, compared to Grace-basic, DRL-OR*, RSIR, ECMP, and OSPF, *Grace* achieves average delay reductions of 12.7%, 14.1%, 15.9%, 13.1%, and 16.6% under NSFnet and 4.0%, 13.2%, 13.6%, 10.6%, and 14.2% under AT&T, respectively.

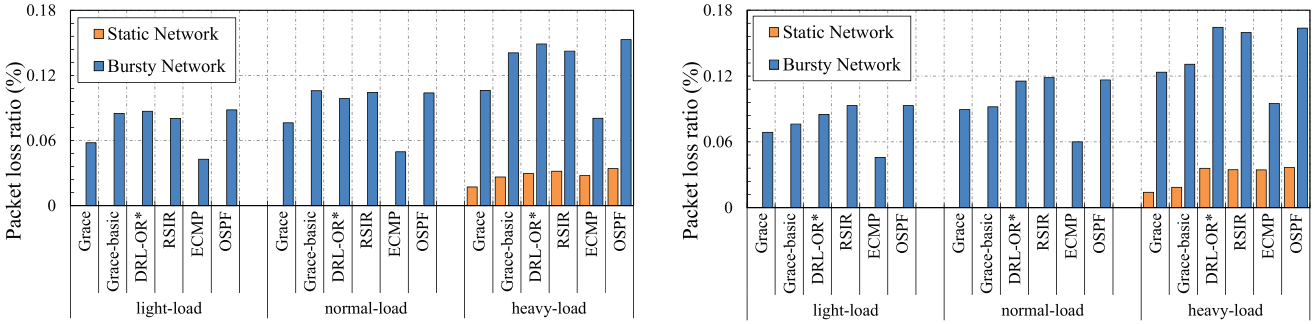
2) Throughput: Grace performs more satisfactorily on network throughput, and the performance gap in the bursty environment is further widened as desired, which can be seen in Fig. 9. For instance, under the bursty NSFnet, *Grace* outperforms Grace-basic, DRL-OR*, RSIR, ECMP and OSPF by 28.4%, 31.2%, 36.1%, 32.9% and 41.2%, respectively, indicating that it has the best robustness due to its ability to perceive graph-structured information. However, we can observe that the gap between *Grace* and Grace-basic is reduced when the topology is larger (shown in Fig. 9(b)), which illustrates that the incremental training mode exhibits more efficiency under a simple network. In general, we find that the network becomes saturated when the traffic load is larger, which leads to poorer throughput due to congestion.

3) Packet Loss Ratio: From Fig. 10, we discover that even though the objective of *Grace* does not directly consider the packet loss, it still performs routing reasonably and decreases the induced loss. In the static environment with a light load, no packet loss occurs due to light congestion. When traffic becomes bursty, the sudden change in traffic volume causes



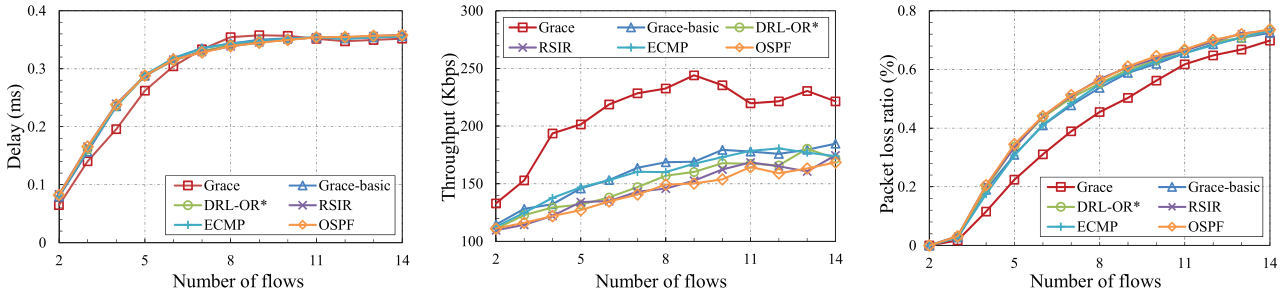
(a) NSFnet. (b) AT&T.

Fig. 9. Throughput comparison over the traffic load.



(a) NSFnet. (b) AT&T.

Fig. 10. Loss comparison over the traffic load.



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

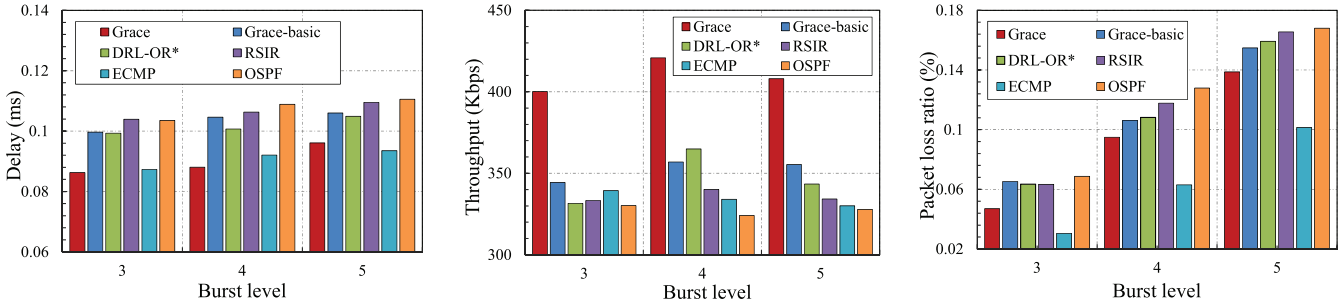
Fig. 11. Performance comparison over the number of flows under NSFnet. In this simulation, we changed the flow number from 2 to 14 to make a comprehensive observation. When the flow number is less than 5, *Grace* outperforms, and the packet loss ratios of these methods are all in a reasonable range. When the flow number exceeds 5, these methods show similar delays and the gap in throughput is still large. However, the packet loss ratio is too large for normal networks.

longer queues in the buffer and even packet loss. *Grace* yields a small change rate for the packet loss ratio, which implies that it has satisfactory adaptability when encountering traffic bursts. On average, compared with Grace-basic, DRL-OR*, RSIR, and OSPF, *Grace* decreases the packet loss ratio by 19.8%, 21.1%, 20.8% and 23.5% under NSFnet and 7.1%, 21.3%, 22.2% and 22.6% under AT&T. It worth noting that ECMP outperforms *Grace* under the bursty environment, since it allocates the flow in different paths, which releases the congestion in a single path.

D. Further Evaluation Under a Heavy Load

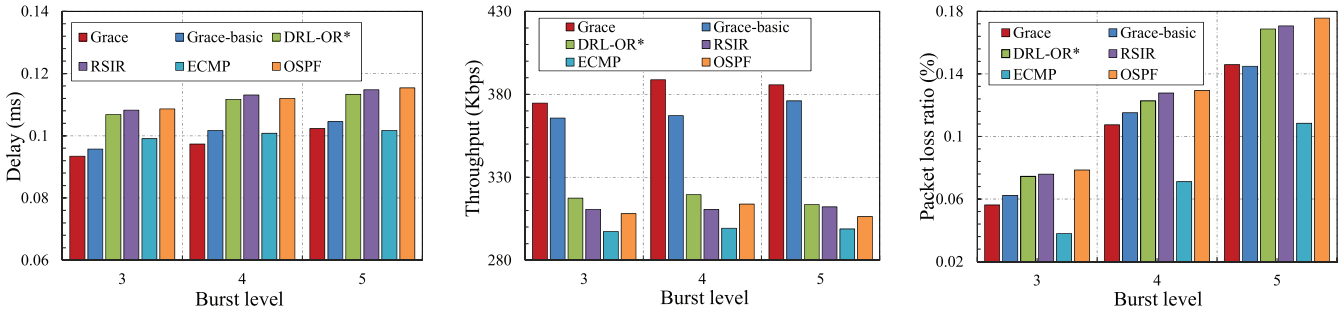
It can be seen from the results shown in subsection IV-C that *Grace* leads to relatively poor performance under a light load due to similar routing capacities in the case with a low traffic injection rate and less queuing time. However, owing to the difficulty of path decision making when the environment changes irregularly, the performance of routing strategies may fluctuate or even decrease with heavier traffic loads. To further explore the potential of *Grace* in dealing with network congestion and traffic bursts under heavy loads, in this subsection, we evaluate the following two conditions.

1) Performance Comparison Under Different Numbers of Concurrent Flows: The influence of the number of concurrent flows is shown in Fig. 11. In this evaluation, the buffer size is set to 500 Kb, and the traffic load is set to a heavy load. Intuitively, DRL-OR*, RSIR, ECMP and OSPF have degraded the packet loss ratio more drastically under higher traffic



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

Fig. 12. Performance comparison over burst flows under NSFnet. Based on different numbers of flows, we set the burst flows while raising the sending rate and packet size to imitate the emergence of traffic bursts. It is worth noting that we changed the number of burst flows from 3 to 5, in order to keep the packet loss ratio at a reasonable range, according to Fig. 11.



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

Fig. 13. Performance comparison over burst flows under the AT&T topology.

TABLE I
AVERAGE TIME OF DECISION-MAKING (S)

Topology	Grace	DRL-OR*	RSIR	ECMP	OSPF
14-node NSFNet topology	7.7×10^{-3}	8.5×10^{-3}	5.9×10^{-1}	1.9×10^{-5}	5.3×10^{-5}
25-node AT&T topology	1.6×10^{-2}	1.7×10^{-2}	2.3	2.7×10^{-5}	2.8×10^{-4}

concurrency. This is because the higher the concurrency is, the higher the queuing delay and congestion. Although ECMP can release congestion by allocating flow to multipaths, it still cannot handle the situation with large concurrent flows. Moreover, Grace-basic achieves unsatisfactory performance due to the neglect of iterative learning during training, which reduces its ability to adapt to complex tasks. Due to the consideration of the relations between network entities and the incremental learning process, *Grace* adaptively handles this situation and thus provides a consistently higher network throughput. More specifically, it increases the average throughput by up to 31.9%, 40.1%, 44.8%, 33.6%, and 47.0% over Grace-basic, DRL-OR*, RSIR, ECMP and OSPF, respectively.

2) Performance Comparison Under Bursty Traffic Injection: Based on different numbers of concurrent flows, we conducted additional simulations under a bursty environment. We distinguished the burst level using the number of concurrent flows F, where F changes from 3 to 5. As shown in Fig. 12, due to the neglect of concurrent flows during the model design stage, RSIR and DRL-OR* exhibit some performance degradation. On average, *Grace* achieves 16.3%, 18.3%, 22.0%, 22.4%, and 25.2% throughput improvements over Grace-basic, DRL-OR*, RSIR, ECMP and OSPF under NSFnet, respectively. ECMP still has better performance in terms of the packet loss ratio compared with *Grace*, however, the delay of *Grace* outperforms it, and the throughput of *Grace* and ECMP has a bigger gap. It is also shown in Fig. 13 that the performance decreases under the larger topology due to the presence of longer forwarding paths, but the gap between the baselines and *Grace* is still large, which demonstrates the rational design of our framework.

TABLE II
PERFORMANCE UNDER LARGE SCALE NETWORKS

	Grace	Grace-basic	DRL-OR*	RISR	ECMP	OSPF
delay	0.158	0.160	0.161	0.167	**0.157**	0.160
throughput	**140.99**	138.74	135.86	73.04	136.10	136.76
loss	0.105	0.103	0.105	0.112	**0.052**	0.107

Overall, combining performance comparisons conducted across diverse simulation scenarios, *Grace* inarguably achieves

TABLE III
PERFORMANCE WITH NETWORK FAILURES

Topology	Condition	Performance	Grace	Grace-basic	DRL-OR*	RISR	ECMP	OSPF
NSFNet	failure-free	delay	**0.1231**	0.1441	0.1454	0.1495	0.1361	0.1485
		throughput	**240.268**	182.0919	179.2825	166.4175	174.9171	166.8097
		loss	0.0691	0.0959	0.0895	0.0946	**0.0467**	0.0941
	failure	delay	**0.1282**	0.1479	0.1483	0.1551	0.1384	0.1480
		throughput	**227.3984**	172.8544	168.8936	149.7291	173.0507	169.2644
		loss	0.0786	0.0962	0.0979	0.1010	**0.0441**	0.1004
AT&T	failure-free	delay	**0.1344**	0.1394	0.1539	0.1545	0.1452	0.1557
		throughput	**215.7164**	203.4583	153.3420	149.3712	149.6974	147.2744
		loss	0.0810	0.0834	0.1047	0.1073	**0.0501**	0.1055
	failure	delay	**0.1340**	0.1405	0.1532	0.1574	0.1454	0.1551
		throughput	**214.7488**	200.1715	151.9905	146.3633	146.2315	149.7291
		loss	0.0845	0.0872	0.1060	0.1096	**0.0531**	0.1010

the best performance, and ECMP is also a satisfying method by allocating traffic into multipaths. However, multipath routing algorithm will incur out-of-order packets in a flow, which require other handling methods and we do not consider this circumstance. In particular, due to its perception of network interconnections and effective routing decision-making ability, *Grace* clearly manifests satisfying robustness under networks with time-varying flows and bursty traffic injection. This makes it more profitable when implemented in reality.

E. Time Efficiency

Usually, the time spent on the decision process after the agents have been trained is related to the computational complexity of DRL-based methods, and the average running time per execution can be used as the time cost to evaluate the time efficiency accordingly [29]. In addition, the computational complexity of DRL-based algorithms always has a connection with the neural network structure. Although the time complexity in the training phase is high due to the repeated updating of parameters, the network does not need to be updated in the testing phase [30]. Therefore, the time complexity during the test is low, and the running time is short, which fulfills the requirements of the real-time network conditions online.

In this paper, we evaluate the average decision-making time (i.e., the time of calculating the next hop) of these methods as the time efficiency metric in the 14-node NSFNet and 25-node AT&T topology. Table. I indicates that the decision time is related to the algorithm's complexity and the network's scale. Consequently, it is observed that decision-making time is notably longer in the AT&T topology compared to that of the 14-node NSFNet. Specifically, RSIR has the poorest performance, while ECMP and OSPF demonstrate fast computation times. As for the two DRL algorithms, we can observe that *Grace* performs better due to its well-structured action-state space setting and neural network design. This suggests that an optimized design can significantly enhance the efficiency of decision-making processes in dynamic routing scenarios.

F. Deep Dive

1) Performance Evaluation Under Large Scale Networks: In the simulation, to evaluate the adaptability of these methods in large-scale networks, we have added evaluations on the GTS topology obtained from Internet Topology Zoo. GTS topology comprises 148 nodes and 196 edges, providing a substantial framework for our analysis. Additionally, we have set the burst level to 4 and configured the traffic load as the normal load. The simulation results are shown in Table. II. We can observe from the table that when the network scale gets larger, the performances of these methods decrease in different degrees. Notably, *Grace* still performs effectiveness and robustness even under these scaled-up conditions. Specifically, RSIR exhibits the worst performance, which is attributed to a larger Q-table caused by the substantial increase in state-action space. Consequently, the convergence of routing calculation is adversely affected, leading to a deterioration in performance. It is noteworthy that ECMP emerges as a suitable alternative under these conditions. It is because the diversity of available paths increase as the network topology expands, effectively reducing network congestion.

2) Performance Evaluation With Network Failure: In the real-world networks, the dynamic nature of link failures poses a significant challenge, stemming from a myriad of internal and external factors. Such factors include, but are not limited to, traffic congestion and hardware breakdown. It is widely recognized that routing decisions lacking adaptability to the changing network topology can lead to a detrimental decline in network performance. In an effort to explore the specific impact of link failure on routing performance, this subsection presents a simulation to evaluate the robustness of *Grace* and its comparative baselines. Similar to related work [31] and [32], we assume that network links are subject to random failure with a 15% probability. These failures are caused by disruptive network congestion or hardware breakdown. Additionally, the burst level is designated as 4, and the traffic load is configured to normal load. As shown in Table. III, the performance metrics of these evaluated methods exhibit a decline during instances of network failures. The primary

TABLE IV
PERFORMANCE WITH DIFFERENT BURST STEPS

Topology	Burst step	Performance	Grace	Grace-basic	DRL-OR*	RISR	ECMP	OSPF
NSFNet	100	delay	**0.1444**	0.1625	0.1668	0.1654	0.1460	0.1650
		throughput	**256.2389**	187.6901	176.5788	179.0207	179.5903	180.7401
		loss	0.1386	0.2218	0.1972	0.2041	**0.0924**	0.2029
	150	delay	0.1596	0.1803	0.1825	0.1828	**0.1548**	0.1822
		throughput	**279.3919**	216.8235	197.3329	197.7436	187.0679	191.2301
		loss	0.2126	0.2343	0.2941	0.2862	**0.1363**	0.3026
	200	delay	0.1785	0.1975	0.2005	0.1956	**0.1638**	0.2007
		throughput	**289.7404**	226.1615	210.3283	217.1156	192.4981	203.4112
		loss	0.3055	0.3975	0.3884	0.3814	**0.1920**	0.3966
AT&T	100	delay	0.1546	0.1584	0.1721	0.1711	**0.1529**	0.1709
		throughput	**225.2888**	211.9083	163.0121	168.1580	156.0432	164.2404
		loss	0.1605	0.1836	0.2100	0.1990	**0.1052**	0.2053
	150	delay	0.1737	0.1785	0.1866	0.1883	**0.1603**	0.1880
		throughput	**243.1999**	230.1695	183.1299	174.7411	167.9253	178.3220
		loss	0.2490	0.2780	0.3046	0.3075	**0.1491**	0.3042
	200	delay	0.1890	0.1976	0.2055	0.2050	**0.1709**	0.2052
		throughput	**274.1193**	247.6901	186.8221	187.1464	166.4658	180.6203
		loss	0.3060	0.3400	0.4149	0.4160	0.2053	0.4294

effects of these failures are observed in the reduction of throughput and the increase in packet loss ratio. Despite these adverse conditions, *Grace* still achieves considerable performance, including the best delay and superior throughput compared with other baselines. This resilience underscores the robustness of *Grace* in the face of network disruptions.

3) Performance Evaluation With Different Burst Steps: In this subsection, we implement different burst steps to explore the impact of the network burst degree. In particular, the burst step is set to 100, 150, and 200, and the simulation results are listed in Table. IV. We can find from the table that ECMP has better delay and loss compared to *Grace*. However, the throughput of ECMP is poor, which indicates that this method cannot make a good tradeoff between network performances. Although *Grace* does not make the best, its three performances are all in a proper range, and it shows feasible adaptability under different burst circumstances.

V. CONCLUSION

In this paper, we introduce a new framework called *Grace* to achieve more powerful DRL-based distributed routing optimization in dynamic network environments. Specifically, we incorporate graph embedding in our DRL-based routing algorithm, which is able to represent the explicit knowledge of the network topology. By applying an ICM in this framework, we represent curiosity-driven explorations to help *Grace* tackle large environments with sparse rewards. To train such a distributed multiagent system framework, an incremental training method for *Grace* is introduced to make the model adapt to changing scenarios in a gradual manner. Extensive experiments conducted on two real-world topologies demonstrate the rationality and effectiveness of *Grace* in dynamic network environments involving both varying traffic loads and bursty traffic.

REFERENCES

[1] P. A. Humblet, "Another adaptive distributed shortest path algorithm," *IEEE Trans. Commun.*, vol. 39, no. 6, pp. 995–1003, Jun. 1991.

[2] M. Dzida, M. Zagozdzon, M. Pioro, and A. Tomaszewski, "Optimization of the shortest-path routing with equal-cost multi-path load balancing," in *Proc. Int. Conf. Transparent Opt. Netw.*, vol. 3, Jun. 2006, pp. 9–12.

[3] N. He et al., "Leveraging deep reinforcement learning with attention mechanism for virtual network function placement and routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 4, pp. 1186–1201, Apr. 2023.

[4] Y. Shao, A. Rezaee, S. C. Liew, and V. W. S. Chan, "Significant sampling for shortest path routing: A deep reinforcement learning solution," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2234–2248, Oct. 2020.

[5] J. Hu et al., "Load balancing with multi-level signals for lossless datacenter networks," *IEEE/ACM Trans. Netw.*, vol. 32, no. 3, pp. 2736–2748, Jun. 2024.

[6] 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.

[7] M. Ye, J. Zhang, Z. Guo, and H. J. Chao, "LARRI: Learning-based adaptive range routing for highly dynamic traffic in WANs," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, May 2023, pp. 1–10.

[8] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da Fonseca, "Intelligent routing based on reinforcement learning for software-defined networking," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 870–881, Mar. 2021.

[9] C. Liu, M. Xu, Y. Yang, and N. Geng, "DRL-OR: Deep reinforcement learning-based online routing for multi-type service requirements," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.

[10] W. Wei et al., "GRL-PS: Graph embedding-based DRL approach for adaptive path selection," *IEEE Trans. Netw. Service Manage.*, vol. 20, no. 3, pp. 2639–2651, Mar. 2023.

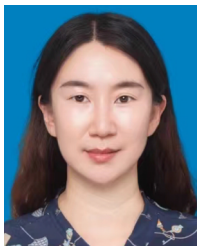
[11] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowl.-Based Syst.*, vol. 151, pp. 78–94, Jul. 2018.

[12] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 2778–2787.

[13] B. Li and L. Chan. *A Pythonic Discrete-Event Network Simulator*. Pancypeng and Lunarss. Accessed: Mar. 10, 2023. [Online]. Available: https://github.com/TL-System/ns.py

[14] N. C. Luong et al., "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.

[15] P. Xuan, V. Lesser, and S. Zilberstein, "Communication decisions in multi-agent cooperation: Model and experiments," in *Proc. 5th Int. Conf. Auton. Agents*, 2001, pp. 616–623.
[16] L. Chen, B. Hu, Z.-H. Guan, L. Zhao, and X. Shen, "Multiagent meta-reinforcement learning for adaptive multipath routing optimization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 10, pp. 5374–5386, Oct. 2022.
[17] X. Liao et al., "Distributed intelligence: A verification for multi-agent DRL-based multibeam satellite resource allocation," *IEEE Commun. Lett.*, vol. 24, no. 12, pp. 2785–2789, Dec. 2020.
[18] J. Xie et al., "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 393–430, 1st Quart., 2019.
[19] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio, "RouteNet: Leveraging graph neural networks for network modeling and optimization in SDN," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2260–2270, Oct. 2020.
[20] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2014, pp. 701–710.
[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[22] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behav. Brain Sci.*, vol. 40, p. e253, Jan. 2017.
[23] J. Gao, W. Ye, J. Guo, and Z. Li, "Deep reinforcement learning for indoor mobile robot path planning," *Sensors*, vol. 20, no. 19, p. 5493, Sep. 2020.
[24] J. van Oijen, G. Poppinga, O. Brouwer, A. Aliko, and J. J. Roessingh, "Towards modeling the learning process of aviators using deep reinforcement learning," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2017, pp. 3439–3444.
[25] 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.
[26] P. Pinyoanuntapong, M. Lee, and P. Wang, "Delay-optimal traffic engineering through multi-agent reinforcement learning," in *Proc. IEEE INFOCOM Workshops*, Apr. 2019, pp. 435–442.
[27] M. Torabzadeh and W. Ajib, "A short-time burst degradation classifier for real-time traffic with application in MPLS ingress nodes," *IEEE J. Commun.*, vol. 6, no. 3, pp. 215–224, May 2011.
[28] L. Sequeira, J. Fernández-Navajas, L. Casadesus, J. Saldana, I. Quintana, and J. Ruiz-Mas, "The ifluence of the buffer size in packet loss for competing multimedia and bursty traffic," in *Proc. IEEE Int. Symp. Perform. Eval. Comput. Telecommun. Syst. (SPECTS)*, Jul. 2013, pp. 134–141.
[29] X. Wang, Y. Zhang, R. Shen, Y. Xu, and F.-C. Zheng, "DRL-based energy-efficient resource allocation frameworks for uplink NOMA systems," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7279–7294, Aug. 2020.
[30] Y. Ren, X. Chen, S. Guo, S. Guo, and A. Xiong, "Blockchain-based VEC network trust management: A DRL algorithm for vehicular service offloading and migration," *IEEE Trans. Veh. Technol.*, vol. 70, no. 8, pp. 8148–8160, Aug. 2021.
[31] 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," *J. Netw. Comput. Appl.*, vol. 177, Mar. 2021, Art. no. 102865.
[32] N. Geng, T. Lan, V. Aggarwal, Y. Yang, and M. Xu, "A multi-agent reinforcement learning perspective on distributed traffic engineering," in *Proc. IEEE 28th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2020, pp. 1–11.



Wenting Wei (Member, IEEE) received the M.E. and Ph.D. degrees in telecommunication and information systems from Xidian University in 2014 and 2019, respectively. She is currently an Associate Professor with the State Key Laboratory of ISN, School of Telecommunications Engineering, Xidian University. Her main research interests include data center networking, satellite networks, network virtualization, and intelligent networking.

Huaxi Gu is a Professor with the State Key Laboratory of ISN, Xidian University. He is the Leader of the Youth Innovation Team, Shaanxi Universities. He is leading a project as the Principal Investigator, funded by the National Key Research and Development Program of China. He is also the Principal Investigator for one key, two general, and one youth project funded by the National Natural Science Foundation. He has published more than 200 journal and conference papers, with his research interests being in the areas of networking technologies, network on chip, and optical interconnect.

Liying Fu received the B.E. degree from the School of Telecommunications Engineering, Xidian University, in 2021, where she is currently pursuing the master's degree with the State Key Laboratory of ISN. Her research interests include routing algorithm and intelligent networking.

Baochun Li (Fellow, IEEE) received the B.E. degree from Tsinghua University in 1995 and the M.S. and Ph.D. degrees from the University of Illinois at Urbana-Champaign in 1997 and 2000, respectively. He is a Professor with the Department of Electrical and Computer Engineering at the University of Toronto. His research interests include cloud computing, large-scale data processing, datacenter networking, and coding for distributed storage systems. He is a fellow of the Canadian Academy of Engineering.