Abstract—Software-defined networking (SDN) allows flexible and centralized control in cloud data centers. An elastic set of distributed SDN controllers is often required to provide sufficient yet cost-effective processing capacity. However, this introduces a new challenge: Request Dispatching among the controllers by SDN switches. It is essential to design a dispatching policy for each switch to guide the request distribution. Existing policies are designed under certain assumptions, including a single centralized agent, global network knowledge, and a fixed number of controllers, which often cannot be satisfied in practice. This article proposes MADRina, Multiagent Deep Reinforcement Learning for request dispatching, to design policies with high dispatching adaptability and performance. First, we design a multiagent system to address the limitation of using a centralized agent with global network knowledge. Second, we propose a Deep Neural Network-based adaptive policy to enable request dispatching over an elastic set of controllers. Third, we develop a new algorithm to train the adaptive policies in a multiagent context. We prototype MADRina and build a simulation tool to evaluate its performance using real-world network data and topology. The results show that MADRina can significantly reduce response time by up to 30% compared to existing approaches.

Index Terms—Distributed software-defined networking (SDN) controllers, multiagent deep reinforcement learning (MA-DRL), request dispatching, resource scheduling, SDN.

I. INTRODUCTION

CLOUD computing has attracted increasing attention by providing on-demand resources and services through multiple geographically distributed data centers.

Manuscript received 12 September 2022; revised 13 January 2023 and 17 March 2023; accepted 4 April 2023. Date of publication 3 May 2023; date of current version 17 April 2024. This article was recommended by Associate Editor S. J. Yoo. (Corresponding author: Victoria Huang.)

Victoria Huang is with the HPC and Data Science Department, National Institute of Water and Atmospheric Research, Wellington 6021, New Zealand (e-mail: victoria.huang@niwa.co.nz).

Gang Chen is with the School of Engineering and Computer Science and the Centre for Data Science and Artificial Intelligence, Victoria University of Wellington, Wellington 6011, New Zealand (e-mail: aaron.chen@ecs.vuw.ac.nz).

Xingquan Zuo is with the School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: zuoxq@bupt.edu.cn).

Albert Y. Zomaya is with the School of Computer Science, The University of Sydney, Sydney, NSW 2008, Australia (e-mail: albert.zomaya@sydney.edu.au).

Nasrin Sohrabi, Zahir Tari, and Qiang Fu are with the School of Computing Technologies, RMIT University, Melbourne, VIC 3000, Australia (e-mail: nasrin.sohrabi@rmit.edu.au; zahir.tari@rmit.edu.au; qiang.fu@rmit.edu.au).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCYB.2023.3266448.

Digital Object Identifier 10.1109/TCYB.2023.3266448

To support flexible and efficient communication, production data centers often adopt software-defined networking (SDN) to manage enterprise networks [1] and wide area networks [2].

Compared to traditional computer networks, SDN empowers network operators with flexible network management and rapid network policy deployment [2]. To provide sufficient processing capacity for the increasing communication activities in a network, especially in production data centers, distributed controller architectures featuring the joint use of multiple controllers are quickly gaining popularity. To make the best use of controller capacity and achieve high network performance, previous studies [3], [4] suggested that every SDN switch should use its own request dispatching policy based on various factors, such as switch-controller propagation latency and request arrival rate. The dispatching policy selects suitable controllers for new requests based on current network information. Clearly, the careful design of such a policy is of paramount importance in multicontroller SDNs [3], [5].

Considering the complexity of network environments (e.g., heterogeneous controllers and traffic dynamics), it is difficult to accurately model the environment and manually design effective dispatching policies [6]. However, deep reinforcement learning (DRL)-based approaches can fully automate the policy design process by interacting with unknown network environments, making them a desirable choice. Recently, DRL has been successfully utilized to design routing policies in SDNs [7], [8] and dispatching policies for resource management [9], [10]. Nevertheless, when directly applying existing DRL algorithms [11], [12] to our policy design problem, there are two main challenges.

Single Agent With Global Knowledge: Many existing approaches [5], [9], [10], [13], [14] rely on a single agent to learn a policy by using global network information (i.e., a fully observable environment). For example, Park and Park [10] trained a centralized agent to dynamically dispatch jobs to machines in semiconductor packaging facilities. Similarly, a centralized agent was trained in [9] to allocate power demand among a collection of generation units in a smart grid. Such centralized approaches are susceptible to scalability issues due to their inability to handle large volumes of jobs. In fact, a single agent can easily become the performance bottleneck of the whole network. Additionally, obtaining timely global information over the entire network can result in significant communication overhead. These issues are aggravated by the presence of geographically dispersed data centers [15]. One
solution is to employ one agent for each switch to independently learn its policy using an existing single-agent DRL (SA-DRL) algorithm while treating other agents as part of the environment. However, from each agent’s perspective, the environment becomes nonstationary since other agents keep updating their policies over time. This nonstationarity violates the underlying Markov assumption commonly required by SA-DRL algorithms [11], [12], resulting in unreliable learning behaviors. This is a well-acknowledged challenge in existing multiagent DRL (MA-DRL) works [16], [17] and has also been evidenced by our evaluation in Section VII-B1.

Static Set of SDN Controllers: While many DRL-based methods have been proposed for resource management [9], [13], [18], [19], adopting DRL in the SDN dispatching policy design is not yet well researched. Typically, a policy in DRL is represented using a deep neural network (DNN) [11], [12], with each output node corresponding to a distinct SDN controller (i.e., “Nonadaptive policy” in Fig. 1). Specifically, every switch executes a forward pass of the DNN to probabilistically select a controller to process each new incoming request. Unfortunately, this policy design becomes inapplicable when the number and location of controllers change over time, driven by the varying network traffic demand [20]. This issue was first identified in [5] which designed a DNN policy that calculates the priority of using each controller individually (i.e., “Adaptive policy” in Fig. 1). However, this policy must be used repeatedly whenever a new request arrives, incurring non-negligible policy processing overhead. Moreover, the policy was trained using SA-DRL. On the other hand, in DRL, the trainable parameters in a policy are updated along the direction of their policy gradients [21]. Following [11], [12], it is straightforward to estimate policy gradients of a traditional DNN policy that requires the collection of controllers to be fixed in a SDN. However, existing policy gradient techniques cannot be utilized to directly calculate the gradients of a policy that can adapt to a changing number of controllers in an MA-DRL setting.

In summary, existing approaches work well assuming a single agent with global knowledge and a static set of SDN controllers. However, this assumption limits their applicability in practice. To address this limitation, we develop Multiagent DRL for request dispatching (MADRina), a multiagent system driven by adaptive policies. MADRina only requires local knowledge and dispatches requests across an elastic set of SDN controllers, where the number, location, and capacity of the controllers may change. Our contributions are listed as follows.

1) To effectively carry out request dispatching in SDN without relying on global knowledge, we propose a multiagent system based on the mathematical framework of fully cooperative and partially observable multiagent Markov decision process (MA-MDP). Each agent in the system is co-located with a switch and only needs to use the information locally accessible to the switch. All agents are trained simultaneously by a newly developed MA-DRL algorithm to jointly optimize the global network performance measured by the average request response time. We adopt the principle of centralized training and decentralized execution, realized by using a centrally maintained and highly scalable value function in our algorithm [22], such that agents can perform independently and contribute positively to global performance.

2) To facilitate efficient and scalable request dispatching across an elastic set of SDN controllers, we propose a novel DNN-based adaptive policy design. The common strategy in the literature is to activate the designed policy for every single request [5]. In contrast, our policy assigns a real-valued priority to each controller based on its current workload for a fixed time period, during which controllers are selected for request processing based on a probability proportional to their priorities. This is the first time that a policy is designed to support a continuous action space in the form of controller priorities, effectively preventing frequent activation of the policy network and thereby ensuring policy efficiency and scalability. Meanwhile, to stabilize policy training and ensure dispatching performance, overloaded and distant controllers are filtered out by a simple yet effective controller filtering mechanism, for both training and actual dispatching.

3) We have developed a novel MA-DRL algorithm called multiagent proximal policy optimization (MA-PPO) to effectively train our adaptive policies. MA-PPO extends the widely used proximal policy optimization (PPO) algorithm [11] to address the challenges and requirements that are essential to multiagent learning. Large-scale SDN networks with distributed controllers inevitably impose a large multiagent joint action space. As the action space expands in scale, it becomes ineffective to use a centralized $Q$-function to drive the policy training process, which is widely exercised by many state-of-the-art DRL algorithms [22], [23]. To address this issue, MA-PPO is designed to learn the state value

In agriculture, madrina means lead mare, similar to our agents which act as leaders of the switches to guide the request dispatching. In Spanish, madrina means godmother, who offers mentorship to the child.

$Q$-function measures the expected long-term cumulative rewards if the agents initiate from the given environment state and take the given actions.
function instead of the $Q$-function. As a result, policy training no longer relies on the multiagent action space, leading to significantly improved algorithm effectiveness and scalability, as evidenced in Section VII. MA-PPO also features an efficient policy gradient estimation technique derived mathematically in this article.

4) We prototype MADRina and build a simulation tool for an extensive set of analyses using real-world network data and topologies. Compared to existing approaches, MADRina can reduce response time by up to 30%.

The remainder of this article is organized as follows. Section II discusses existing solutions to the request dispatching problem in SDN. We present our problem formulation in Section III. The adaptive policy design and our new multiagent training algorithm are presented in Sections IV and V, respectively. Section VI elaborates on system implementation and network settings, followed by a discussion of experimental results in Section VII. Section VIII concludes this article.

II. RELATED WORK

Generally speaking, the request dispatching problem in SDN can be addressed either at a switch or request level. At the switch level, solving the request dispatching problem means finding the right switch-controller mapping. At the request level, requests from one switch can be distributed and processed among multiple controllers.

A. Switch-Level Request Dispatching

Different approaches have been proposed to find the switch-controller mapping, for example, exact algorithms [24] and approximation algorithms [25], [26]. Due to the NP-hardness of the switch-controller mapping problem [25], it is computationally expensive to find an optimal solution for large networks. Thus, heuristic methods have been widely explored in [27], [28], and [29]. Cui et al. [28] introduced a greedy heuristic approach, which prioritizes selecting the controller with the highest response time and the most heavily loaded switch. The chosen switch is then migrated to the controller with the lightest workload to reduce controller response time. Similar heuristic approaches have also been used in [27] and [29].

Note that all algorithms in this category assume that requests generated from one switch can only be handled by its mapped controller. Whenever the switch is remapped from one controller to another, the switch’s workload will be transferred entirely to the new controller, making the new controller susceptible to being overloaded. Moreover, the request dispatching can only be performed at a coarse level (i.e., switch level), restricting the opportunity of properly distributing workload across all controllers to achieve high network performance, as demonstrated in [3] and [30].

Compared to the switch-level dispatching, request-level dispatching is performed at a more fine-grained level. Therefore, this article will focus on request-level dispatching.

B. Request-Level Request Dispatching

Some approaches allow requests from one switch to be distributed and processed among multiple controllers. For example, BLAC [3] introduced a scheduling layer, where multiple schedulers are deployed to distribute requests from switches to different controllers. Similarly, to reduce overheads caused by switch-controller remapping and to balance the controller’s workload, Al-Tam and Correia [31] suggested partially transferring the workload from overloads controllers to underloaded ones. Similar work can also be found in [4] and [20].

To capture the correlation between the policy and its performance, model-based methods have been proposed [4], [20], [32], [20]. For example, the Lyapunov optimization was used in [32] to minimize power consumption in wireless networks. In [20], a queuing model was adopted, and a gradient descent (GD)-based algorithm was designed to compute the request distribution to minimize the average response time.

However, these solutions have certain limitations. For example, the introduction of a centralized super controller limits the scalability of the control plane [30]. In addition, existing work mostly used heuristics which cannot guarantee the quality of the solution (see Section VII-C1). The methods using Lyapunov optimization [32] were designed for solving a different problem. The network performance achieved by the GD-based algorithm in [20] is sensitive to the accuracy of the used queuing models. An inaccurate model will hinder the GD-based algorithm from achieving its “best” (“optimal”) performance (see Section VII-C2).

Compared to existing request dispatching methods, MADRina uses multiple decentralized agents to individually make the dispatching decisions based on their local network knowledge to achieve high scalability. Moreover, the adoption of a model-free DRL enables the proposed agents to automatically learn the optimal dispatching policy with little or no prior knowledge of the network environment.

C. DRL-Based Resource Management

Recently, model-free DRL approaches have been successfully utilized to tackle many resource management problems [8], [9], [10], [13], [14], [18], [33, 34], [35]. For example, a self-organizing neural scheduler was proposed in [13] for flexible job shop scheduling. The scheduler utilized a self-organizing population to search for scheduling solutions. Meanwhile, DRL was applied to determine the mutation operation and its location in the candidate solutions. Similarly, Zhao et al. [34] proposed a framework for real-time dynamic scheduling of the blast furnace gas system. Specifically, a heuristic dynamic programming framework was proposed for dynamic and real-time scheduling, driven by a DRL-based semi-supervised granulation process to predict the gas tank levels. Other DRL-based approaches can also be found in [9] and [10].

DRL fully automates the policy design process and noticeably improves the performance of designed policies [6]. However, many existing approaches are designed under the assumption of a single agent and a fully observable
environment that does not always apply to the request dispatching problem in SDNs, as mentioned in Section I.

To address these issues, MA-DRL techniques have been developed in [9], [33], and [36]. Cooperative double Q-learning is proposed to find the optimal signal timing strategy for large-scale traffic signal control problems [33]. Similarly, a distributed Q-learning-based algorithm is proposed for industrial process optimization [9]. However, the value function search methods in [9], [33], [37] may not be suitable for our problem. This is due to the need to enumerate the entire action space to find the optimal action for every dispatching decision, limiting its efficiency in traffic-intensive networks.

Although several multiagent policy search algorithms, for example, multiagent deep deterministic policy gradient (MADDPG) [22], have been exploited for resource allocation [18], [19], they need to learn a centralized Q-network with the high-dimensional multiagent joint action space as its input. The high-dimensional input increases the complexity of the Q-function network, thus reducing the training effectiveness. Whenever the joint action space changes, the Q-function must be retrained from scratch. To avoid the problems introduced by the Q-function, multiagent policy proximal optimization with feature pruned (MAPPO-FP) [38] was proposed which trained the value function instead of the Q-function. However, MAPPO-FP requires the global state input from the environment, which may not always be practical in an SDN network. Moreover, as pointed out by [39], the state input in MAPPO-FP needs to be manually designed/pruned to avoid overlapping between agent-specific features and global state. In comparison, our proposed MA-PPO uses the concatenation of the agents’ local observations as input. MAPPO-FP also assumed homogeneous agents and all agents share the same policy, while our proposed MA-PPO can support both homogeneous and heterogeneous agents. Furthermore, most of the policies trained via existing MA-DRL algorithms cannot adaptively support a varying number of actions and thus may not scale well to large networks [40].

To address these issues, we have designed a new adaptive policy representation to support request dispatching among a changing number of controllers. In line with the new policy representation, a new policy search algorithm called MA-PPO is proposed. Compared to MADDPG and COMA, MA-PPO learns a centrally maintained and highly scalable value function, which avoids the multiagent joint action space and keeps the value function network small and easy to train.

III. PROBLEM FORMULATION

In this section, we describe the policy design problem in an SDN network. Instead of relying on the global network knowledge, we formulate our policy design problem as an MA-MDP, ensuring that all agents cooperate with each other to minimize the average request response time using their local network knowledge.

A. Policy Design Problem in SDN

As shown in Fig. 2, we consider an SDN network with switches $\{Sw_1, \ldots, Sw_N\}$ and controllers $\{C_1, \ldots, C_M\}$. Each switch is equipped with a co-located agent. Whenever a new packet arrives at a switch, the switch will generate a request and forward it to a controller chosen by the agent. New packets can arrive at any time. Therefore, the request generation at every switch is stochastic.

We consider that the controllers may have different capacities and the number of the controllers may change to meet the varying network demand. Each controller processes its requests in an FIFO manner [3] and sends corresponding responses back to the switches. The time interval measured by the switch from sending a request to receiving a response is defined as request response time $\tau$.

To reduce communication overheads, we assume each agent only has limited access to network information. For example, the agent located at $Sw_n$ can only observe the request generation history of $Sw_n$ and the one-way propagation latency between $Sw_n$ and $M$ controllers. Each agent collaborates with each other to achieve the same goal of optimizing the network-wide performance in terms of average request response time.

B. MA-MDP Formulation

We formulate the policy design problem as a fully cooperative and partially observable MA-MDP with $N$ agents. Each agent collaborates with the others and learns a request dispatching policy through continuously interacting with the environment using local network knowledge. During each time step $t$, every agent $A_{gt_n}$ receives a local observation $z^t_n$ from the environment and takes an action $a^t_n \in A^t$ based on its policy $\pi^t_{\theta^t_n}$. Then, each agent receives a reward $r^t_n$ and the next local observation $z^{t+1}_n$ from the environment. The goal of MA-MDP is hence to identify the optimal policies $\{\pi^t_{\theta^t_n}\}_{t=1}^N$ so as to maximize the expected joint cumulative rewards

$$J(\{\pi^t_{\theta^t_n}\}_{t=1}^N) = \mathbb{E}_{\{a^t_n \sim \pi^t_{\theta^t_n}\}_{t=1}^T} \left[ \sum_{t=0}^{T} \gamma^t \sum_{n=1}^{N} r^t_n(z^t_n, a^t_n) \right]$$

(1)

where $\gamma \in [0, 1)$ is a discount factor. Evaluation with different $\gamma$ values will be reported in Section VII-A3.

In the SDN policy design problem, each agent is a request scheduler which controls one of the $N$ SDN switches for

---

\$^{4}$Value function search learns the optimal value function. The value function is used to extract the optimal policy by greedily selecting the action that maximizes the long-term rewards. More details about value function search can be found in [21].
request dispatching, as shown in Fig. 2. The environment is
the SDN network described in Section III-A. From its local
observation $z_i^n$, agent $A_{t_n}$ can extract information related
to each controller using an observation function $O^n(z_i^n, m)$. For
example, $A_{t_n}$’s local observation $\zeta_i^{n,m}$ of controller $C_m$ is
$\zeta_i^{n,m} = O^n(z_i^n, m)$ (see Section VI-B for detailed local
observation). The action $a_i^n = (d_i^{n,m})_{m=1}^M$ specifies the priority $d_i^{n,m}$ of each controller $C_m$, which will be mapped into dispatching
probabilities for an agent $A_{t_n}$ to dispatch its new requests
to any controller $C_m$ during a fixed time interval (e.g., from
time step $t$ to $t+1$). Using the dispatching probabilities over a
fixed time interval can avoid frequent activation of the policy
network, ensuring its efficiency and scalability.

Generally, in DRL, the reward received by $A_{t_n}$ is asso-
ciated with the objectives, that is, request response time.
Therefore, by maximizing the reward, agents learn policies $[\pi_{\theta_n}]_{n=1}^N$ that optimize the objectives. In this article, $A_{t_n}$’s
reward $r_t^n$ is based on the responses it received from all
controllers during time $t$ and $t+1$

$$r_t^n = \xi X_t^n - \sum_{s=1}^{X_t^n} t_s^n$$  \hspace{1cm} (2)

where $X_t^n$ is the total number of responses received between
t and $t+1$ by $A_{t_n}$ and $t_s^n$ is the response time from one of
the $X_t^n$ responses. $\xi$ is a weight factor that controls the
importance of the throughput $X_t^n$ relative to the response
time. Clearly, all agents prefer to receive more responses with lower
response time according to (2). For this purpose, each agent
$A_{t_n}$ learns a policy $\pi_{\theta_n}$ parameterized by $\theta^n$ that maps observation $z_i^n = (\zeta_i^{n,m})_{m=1}^M$ to its action $a_i^n$. More details on the
adaptive policy design will be presented in Section IV.

IV. ADAPTIVE POLICY DESIGN

In line with the MA-MDP problem formulation, an adaptive
DNN-based policy is proposed in Section IV-A to enable
request dispatching over an arbitrary number of controllers.
Equipped with the policy, a dispatching system is proposed in
Section IV-B with a controller filtering mechanism to stabilize
policy training as well as prevent requests from being sent to
unsuitable controllers.

A. DNN-Based Adaptive Policy

The policy is expected to adapt easily to a changing number
of controllers. However, this requirement is seldom supported
by existing policy representations which only allow a fixed
collection of actions, as mentioned in Section I. One possible
strategy to resolve this issue is to train multiple policies
with each policy targeting a particular number of controllers.
However, each policy needs to be individually evaluated or
trained in advance before being deployed, which leads to high
sampling costs. Thus, instead of training multiple policies, we
design and train an adaptive policy that can support different
numbers of controllers.

To address this issue, we propose a new policy design, as
depicted in Fig. 3. The policy $\pi_{\theta^n}$ takes the local observations $z_i^n$ from agent $A_{t_n}$ as inputs and outputs an action $a_i^n$.
In [5], an action corresponds to a chosen controller for request
processing. This design requires repeated processing of the
policy with respect to every new request, preventing efficient
use of the policy in traffic-intensive networks. This issue is
addressed by defining $a_i^n = (d_i^{n,m})_{m=1}^M$ as the controller
priorities. Instead of executing $\pi_{\theta^n}$ with respect to every new
request, $a_i^n$ is computed for every given time interval as discussed
earlier. Request distribution within the time interval will follow $a_i^n$ consistently.

Initial Priority Mapping: Instead of generating the action
$a_i^n = (d_i^{n,m})_{m=1}^M$ through one run of the policy network, an
agent $A_{t_n}$ feeds its local observation $z_i^n$ of controller $C_m$ to
the DNN in Fig. 3 one-by-one for all controllers. For each
local observation $z_i^n$, the DNN assigns an initial priority value $d_i^{n,m}$ to $C_m$. Instead of directly using the initial priorities $\{d_i^{n,m}\}_{m=1}^M$ as action $\{d_i^{n,m}\}_{m=1}^M$, $\{d_i^{n,m}\}_{m=1}^M$ needs to be
normalized and added noise for exploration (to be discussed
in the next paragraph). For simplicity, we denote the DNN as a
priority function $f_{\theta^n}$ with trainable parameters $\theta^n$, to
distinguish it from the policy $\pi_{\theta^n}$ with additional components for
normalization and exploration,\(^5\) as explained below.

Normalization and Exploration: The Softmax function is
used to normalize all controllers’ initial priorities $\{d_i^{n,m}\}_{m=1}^M$ into
a probability distribution $\{d_i^{n,m}\}_{m=1}^M$, as indicated in Fig. 3.
Rather than using $\{d_i^{n,m}\}_{m=1}^M$ in a deterministic manner, the
agent must continue to explore different request dispatching
distinctions and determine their impact on network
performance during policy training. This is achieved by adding
small Gaussian noises

$$\epsilon_i^{n,m} \sim N(0, \sigma^2), \hspace{1cm} m = 1, \ldots, M$$

\(^5\)The exploration component in a policy is only activated during policy
training for stochastic exploration of different request dispatching
distributions. This component is deactivated during testing.
to $\tilde{a}_{t}^{n,m}$, as defined as follows:

$$a_{t}^{n} = \tilde{a}_{t}^{n} + \epsilon_{t}^{n}$$

where $\tilde{a}_{t}^{n} = \{\tilde{a}_{t}^{n,m}\}_{m=1}^{M}$ and $\epsilon_{t}^{n} = \{\epsilon_{t}^{n,m}\}_{m=1}^{M}$.

In association with the discussion above, the whole action-generation process based on our new policy design can be formulated as follows:

$$a_{t}^{n} = \begin{bmatrix} a_{t}^{1,1} \\ \vdots \\ a_{t}^{n,M} \end{bmatrix} = \pi_{\theta^{n}} \left( \begin{bmatrix} z_{t}^{1,1} \\ \vdots \\ z_{t}^{n,M} \end{bmatrix} \right)$$

$$= \begin{bmatrix} \text{Softmax} \left( f_{\theta^{n}} \left( z_{t}^{1,1} \right) \right) + \epsilon_{t}^{1,1} \\ \vdots \\ \text{Softmax} \left( f_{\theta^{n}} \left( z_{t}^{n,M} \right) \right) + \epsilon_{t}^{n,M} \end{bmatrix}.$$  \hfill (4)

Because of $\epsilon_{t}^{n}$, (4) produces $a_{t}^{n}$ as the continuous action output in a stochastic manner.

**B. Dispatching System**

When performing request dispatching, the switch should avoid sending requests to unsuitable controllers, for example, overloaded or remotely located controllers. To this end, a controller filtering mechanism is designed and used before mapping $a_{t}^{n}$ to dispatching probabilities. In particular, the agent keeps track of the operating status of all controllers and maintains a candidate controller list $L_{t}^{n} = \{L_{t}^{n,m}\}_{m=1}^{M}$. Preference is given to controllers with relatively small propagation latency from the agent as well as controllers under moderate or low workload. Accordingly, up to $\chi$ controllers can be considered as candidates by the agent.

Armed with the adaptive policy and the controller filtering mechanism, a dispatching system is designed for each switch, as depicted in Fig. 4. The dispatching system takes the local observations as inputs and outputs the request dispatching probabilities for the switch. Specifically, given local observations, an agent’s policy generates the action, that is, controller priorities according to (4). After that, overloaded or remotely located controllers are filtered by assigning 0 to its corresponding priority value, the filtered action $\{\tilde{a}_{t}^{n,m}\}_{m=1}^{M}$ is then mapped to a dispatching probability $\{p_{t}^{n,m}\}_{m=1}^{M}$ through function $T$. The newly arriving request will be sent to a controller selected randomly based on the probability proportional to its priority. Meanwhile, to avoid any requests being assigned to an unreachable controller, a time-out mechanism is adopted where a controller is considered unreachable if a corresponding response is received after a specified period of time. Whenever a controller becomes unreachable, the switch will remove the unavailable controller and recompute the probabilities which includes the long-distance controllers if the capacity of available nearby controllers cannot handle the request demand. Evaluation with different numbers of controllers becoming unavailable will be reported in Section VII-A2.

**V. ADAPTIVE POLICY TRAINING**

In this section, a new training algorithm named MA-PPO is developed to train the adaptive policy with a new mathematical method to estimate the policy gradient. By removing the high-dimensional multiagent joint action space, MA-PPO substantially reduces the DNN input dimension, making policy training effective and scalable in large SDN networks.

**A. Multiagent Proximal Policy Optimization**

Aiming at training a policy for each SDN switch, one straightforward approach is to directly adopt the SA-DRL algorithm. That is, one DRL agent is placed at a specific SDN switch to continuously and independently learn its policy while the other agents are treated as part of the environment. Despite its simplicity, the training process is vulnerable to the nonstationary environment problem [22]. The reward received by each agent and the global state transition do not depend solely on one agent’s individual actions. Instead, they are affected by the joint actions from all agents. When the agents continuously train their individual policies, the environment observed by each agent becomes nonstationary (i.e., violating the Markov property), preventing DRL algorithms to converge reliably [22]. Evaluation of the single-agent learning approach (denoted by SA-PPO-MA) in a multiagent environment will be reported in Section VII-B1.

Without pursuing a learning system using SA-DRL, we propose MA-PPO, a multiagent extension of PPO [11]. MA-PPO fulfills the general principle of centralized training...
and decentralized execution [22], which is essential for reliable MA-DRL. A common strategy in the literature (e.g., MADDPG [22] and COMA [23]) is to use a centralized Q-function approximated by a DNN to drive the multiagent learning. However, the Q-function requires a joint multiagent action space as input, which becomes large in large-scale SDN networks with distributed controllers, making policy training ineffective. MA-PPO avoids the multiagent joint action space in the Q-function by learning a centrally maintained parametric value function $V_\omega$ using all agents’ joint local observation $\{s^n_{t}\}_{n=1}^{N}$ as input. The value function $V_\omega$ is then shared among the agents. MA-PPO and MADDPG are compared in Section V-B.

In MA-PPO, $V_\omega$ is approximated by a DNN. Following PPO, $V_\omega$ is learned in an on-policy fashion based on a collection of network state-transition samples obtained from the current policies $\{\pi_\theta^n\}_{n=1}^{N}$. Each state-transition sample $u$ records agents’ local observations, actions, and rewards

$$u = \{s^n_{t}, a^n_{t}, r^n_{t}, s^n_{t+1}\}.$$  (5)

Several mini-batches of samples denoted by $B$ can be retrieved from the collection to repeatedly train $V_\omega$ to minimize the Bellman loss in the following:

$$H(V_\omega) = \frac{1}{|B|} \sum_{B} \left( V_\omega(s_t) - \sum_{n=1}^{N} \gamma^n r^n_t - \gamma V_\omega(s_{t+1}) \right)^2.$$  (6)

For notation simplicity, we replace multiagent joint observation $\{s^n_{t}\}_{n=1}^{N}$ with $s_t$.

Guided by the trained $V_\omega$, each agent in MA-PPO continues to use the sampled mini-batches to update its policy $\pi_\theta^n$ along the direction of the estimated policy gradient $\nabla_{\theta^n} J(\pi_\theta^n)$ in Section V-B.

B. Policy Gradient

Following PPO, in order to estimate $\nabla_{\theta^n} J(\pi_\theta^n)$, MA-PPO must find a way to estimate the following gradient:

$$\nabla_{\theta^n} \mathcal{L}(\pi_\theta^n) = \nabla_{\theta^n} \mathbb{E}_{T_{\text{old}}} \left[ \frac{\pi_\theta^n(a^n_t|s^n_t)}{\pi_\theta^n_{\text{old}}(a^n_t|s^n_t)} A_T(s_t, \{a^n_{t}\}_{n=1}^{N}) \right] \approx \frac{1}{|B|} \sum_{B} A_T(s_t, \{a^n_{t}\}_{n=1}^{N}) \nabla_{\theta^n} \log \pi_\theta^n(a^n_t|s^n_t),$$  (7)

where $\theta^n$ and $\theta^n_{\text{old}}$ refer to the policy parameters after and before a policy update in a training iteration (TI), respectively. $A_T(s_t, \{a^n_{t}\}_{n=1}^{N})$ is the advantage function obtained through $V_\omega$ by using the generalized advantage estimation (GAE) technique developed in [41].

According to (4)

$$\epsilon_{t,n,m} = \text{Softmax}(f_\theta^n(z_{t,n,m})), \quad \mu_{t,n,m} = \text{Softmax}(f_\theta^n(z_{t,n,m})).$$

Therefore, each element $\epsilon_{t,n,m}$ in $a^n_t$ follows a Gaussian distribution:

$$a_{t,n,m} = \text{Softmax}(f_\theta^n(z_{t,n,m})) \sim \mathcal{N}(\text{Softmax}(f_\theta^n(z_{t,n,m})), \sigma^2).$$

Note that, the Gaussian noise $\epsilon_{t,n,m}$ for each $a_{t,n,m}$ is independently sampled. Therefore

$$\pi_{\theta^n}^{\epsilon}(a^n_t|z^n_t) = \prod_{m=1}^{M} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2} (\frac{a_{t,n,m} - \mu_{t,n,m}}{\sigma})^2}$$

where $\mu_{t,n,m} = \text{Softmax}(f_\theta^n(z_{t,n,m})).$

For each sample $u_t \in B$, $\nabla_{\theta^n} \pi_\theta^n(a^n_t|z^n_t)$ can be calculated by using $a^n_t$ and $z^n_t$ recorded in sample $u_t$ as shown in the following:

$$\nabla_{\theta^n} \pi_\theta^n(a^n_t|z^n_t) = \pi_{\theta^n}^{\epsilon}(a^n_t|z^n_t) \nabla_{\theta^n} \log \pi_{\theta^n}^{\epsilon}(a^n_t|z^n_t)$$

$$= \pi_{\theta^n}^{\epsilon}(a^n_t|z^n_t) \nabla_{\theta^n} \log \left( \prod_{m=1}^{M} \frac{1}{\sigma^2} e^{-\frac{1}{2} (\frac{a_{t,n,m} - \mu_{t,n,m}}{\sigma})^2} \right)$$

$$= \pi_{\theta^n}^{\epsilon}(a^n_t|z^n_t) \nabla_{\theta^n} \left( \sum_{m=1}^{M} \log \left( e^{-\frac{1}{2} (\frac{a_{t,n,m} - \mu_{t,n,m}}{\sigma})^2} \right) \right)$$

$$= \pi_{\theta^n}^{\epsilon}(a^n_t|z^n_t) \left( \sum_{m=1}^{M} \nabla_{\theta^n} \log \left( e^{-\frac{1}{2} (\frac{a_{t,n,m} - \mu_{t,n,m}}{\sigma})^2} \right) \right)$$

$$= \pi_{\theta^n}^{\epsilon}(a^n_t|z^n_t) \left( \sum_{m=1}^{M} \left( \frac{1}{\sigma^2} (a_{t,n,m} - \mu_{t,n,m}) \nabla_{\theta^n} \mu_{t,n,m} \right) \right).$$  (8)

Given

$$\mu_{t,n,m} = \text{Softmax}(f_\theta^n(z_{t,n,m})), \quad \epsilon_{t,n,m} = \frac{\epsilon_{t,n,m}}{\sum_{i=1}^{M} e^{\epsilon_{t,n,m}}}$$

we have

$$\nabla_{\theta^n} \mu_{t,n,m} = \frac{\epsilon_{t,n,m}}{\sum_{i=1}^{M} e^{\epsilon_{t,n,m}}} \sum_{i=1}^{M} e^{\epsilon_{t,n,m}} \left( \nabla_{\theta^n} f_\theta^n(z_{t,n,m}) - \nabla_{\theta^n} f_\theta^n(z_{t,n,m}) \right)$$  (9)

where $\nabla_{\theta^n} f_\theta^n(z_{t,n,m})$ is the gradient of the priority function (i.e., the DNN) in Fig. 3.

Summarizing the above discussions, with respect to a mini-batch $B$, $\nabla_{\theta^n} \mathcal{L}(\pi_\theta^n)$ is estimated using the following:

$$\nabla_{\theta^n} \mathcal{L}(\pi_\theta^n) \approx \frac{1}{|B|} \sum_{B} A_T(s_t, \{a^n_{t}\}_{n=1}^{N}) \pi_{\theta^n}^{\epsilon}(a^n_t|z^n_t) \left( \sum_{m=1}^{M} (a_{t,n,m} - \mu_{t,n,m}) \right) \times \left( \sum_{i=1}^{M} e^{\epsilon_{t,n,m}} \right)^{-2} \times \left( \nabla_{\theta^n} f_\theta^n(z_{t,n,m}) - \nabla_{\theta^n} f_\theta^n(z_{t,n,m}) \right).$$  (10)

provided that $\pi_{\theta^n}/\pi_{\theta^n_{\text{old}}}$ falls in the range $(-\infty, 1 + \epsilon)$ if $A_T(s_t, \{a^n_{t}\}_{n=1}^{N}) > 0$, or $(1 - \epsilon, +\infty)$ if $A_T(s_t, \{a^n_{t}\}_{n=1}^{N}) < 0$, $\epsilon$ is a hyper-parameter that is set to 0.2 following PPO.
with respect to any \( \{a^n_t\}_{t=1}^{N} \) and \( s_t \). Otherwise, \( \nabla \theta \cdot \mathcal{L}(\pi_{\theta^n}) = 0 \). Note that to update the policy \( \pi_{\theta^n} \), \( \nabla \theta \cdot \mathcal{L}(\pi_{\theta^n}) \) in (10) requires the multiagent joint observation \( s_{t} = (z_{1}^{t}, \ldots, z_{N}^{t}) \) to calculate \( A_{t}(s_{t}, \{a_{t}^{m,n}\}_{m=1}^{M}) \). Meanwhile, calculating \( \nabla \theta \cdot \mathcal{L}(\pi_{\theta^n}(a^n_t|s^n_t)) \) in (8) only requires an agent’s local observation.

According to PPO, the policy \( \pi_{\theta^n} \) can be improved by repeatedly updating the policy parameters \( \theta^n \) along the direction of \( \nabla \theta \cdot \mathcal{L}(\pi_{\theta^n}) \). With the help of TensorFlow, the gradient calculation can also be fully automated in our training system, regardless of how many controllers are involved. The computational complexity \( \mathcal{O}(M \times \mathcal{O}_{\theta^n}) \) is linear with respect to the number of controllers where \( \mathcal{O}_{\theta^n} \) is the complexity of the DNN \( f_{\theta^n} \).

VI. IMPLEMENTATION AND SIMULATION SETTING

To evaluate MADRina, we developed a prototype and conducted extensive performance evaluations with real-world data. Purely from a research point of view, we were only concerned with request dispatching within the SDN framework. To clearly demonstrate how MADRina works as a multiagent system, we built a simulator for performance evaluation eliminating the impact of SDN’s complex internals. This also makes it easy for future extensions. To ensure that our simulation results are accurate and reliable, a network simulator from [20] is used to accurately simulate the network behaviors. Real-world network topologies and traffic traces have been utilized in our simulation runs (see Section VI-B). We are hence confident that our evaluation should genuinely reflect the real-world performance of the system.

A. MADRina Implementation

We implemented MADRina based on OpenAI baselines\(^9\) and deep learning tool TensorFlow. A fully connected multilayer feedforward neural network (NN) with two hidden layers of 64 ReLU units is adopted for both \( f_{\theta^n} \) and \( V_{\omega} \), the same NN architecture recommended in PPO [11]. Temperature scaling [42] has been applied for NN calibration.

Meanwhile, we follow closely the hyper-parameter settings of PPO on Mujoco benchmarks in [11]. However, there are a few exceptions. Specifically, the Gaussian noises \( \epsilon_t \) in (3) have their standard deviation set to 0.01. During every algorithm run, the policy is trained for 900 TIs which consist of 1800 episodes and each episode contains 60-time steps. Both \( \theta^n \) and \( \omega \) are trained using data sampled from the current TI. The NN parameters \( \theta^n \) and \( \omega \) are trained using Adam optimizer with \( 3 \times 10^{-4} \) learning rate, 40 minibatch size, and 8 epochs.

The agent receives the requests from the co-located switch. The request is associated with a timestamp indicating its arrival time at the switch. The controller has a processing queue and each controller processes its requests in an FIFO manner. Since distributed SDN controllers are designed to provide a sufficient amount of processing capacity and redundancy [43], we set the controller queue to be unlimited to reflect the capacity of distributed controllers and to avoid the unnecessary impact of packet losses during the simulation. The requests are dispatched to the processing queue of the selected controllers. After processing, the responses are directed back to the switches. The whole procedure is executed in real time.

For the policy to work properly, each agent must provide its local observations \( \{z_{t}^{m,n}\}_{m=1}^{M} \) to the priority function \( f_{\theta^n} \) in Fig. 3. Given the importance of controller capacity, their distance, and current availability, as well as the workload experienced by the agent, the local observation \( z_{t}^{m,n} \) with respect to \( C_{m} \) consists of the following network statistics: 1) request arrival rate history of the switch \( S_{w,n} \); 2) the processing capacity of \( C_{m} \); 3) the propagation latency between \( S_{w,n} \) and \( C_{m} \); 4) the queue length of \( C_{m} \); 5) the number of requests sent from \( A_{gt_{n}} \) to \( C_{m} \) during the previous time step; and 6) the total number of requests received by \( C_{m} \) during the previous time step.

In practice, the request arrival history is made up of a list of request arrival rates measured in the past few time steps by the agent. Intuitively, the longer the list, the easier it is for the agent to detect traffic change patterns and adjusts its request dispatching. Moreover, an observation with a longer historical list can better fulfill the Markov property. The impact of the history length will be investigated in Section VII-A3.

B. Network Setting

Simulations are conducted using real-world network topologies (including network latency, links, and nodes) provided by Sprint IP backbone network databases\(^10\). South America and Asia Sprint networks, respectively. A set of heterogeneous controllers with capacities ranging from 6000 pkts/s to 9000 pkts/s are deployed into the network using an existing controller placement algorithm [20].

Each episode is initialized with 0% utilization for all controllers and 0 packets in the network. During the simulation, the requests arriving at each agent follow the Poisson distribution. Weighted Round-Robin is used for request dispatching during the warm-up period. The warm-up period lasts for 30 simulated seconds which is assumed to be sufficiently long for the network to enter and stay in a stationary condition. Each simulation episode runs for 30 simulated minutes which is divided into a series of time steps. Preliminary experiments have been conducted to measure the impact of different intervals for one time step ranging from 10 to 120 simulated seconds. The results from Fig. 5 showed very similar average response time regardless the interval length, indicating that with the interval set to the granularity of tens of seconds, scalability is not an issue. Network operators can also adjust the time interval to reflect the current traffic dynamics, for example, reducing the time interval if the traffic is highly dynamic. In this article, we set every time step to last for 30 consecutive simulated seconds. At the beginning of each time step, each agent executes its policy individually to calculate the priority which is mapped into the probability of dispatching any new requests to each controller for the next time step, that is, the next 30 simulated seconds.

To enable the agent to learn how to dispatch requests under different workloads, two episodes with two request arrival

---

\(^9\)https://github.com/openai/baselines

\(^10\)Sprint Network https://www.sprint.net/performance/.
TABLE I
NETWORK PERFORMANCE CHANGES WHEN CONTROLLERS FAIL. THE DIFFERENCE IN RESPONSE TIME BEFORE AND AFTER CONTROLLER FAILURE IS MOSTLY NEGLIGIBLE (<0.01 s)

| Arrival rate (pkts/s) | Before failures (8 controllers) | After 1 controller fail | After 2 controllers fail | After 3 controllers fail | After 4 controllers fail |
|-----------------------|--------------------------------|------------------------|-------------------------|-------------------------|-------------------------|
| 9800                  | 0.050503 ± 0.000679            | 0.050538 ± 0.000664    | 0.05054 ± 0.000649      | 0.050574 ± 0.000655    | 0.05059 ± 0.000645     |
| 12600                 | 0.050493 ± 0.000134            | 0.050435 ± 0.000156    | 0.050411 ± 0.000186     | 0.050497 ± 0.000153    | 0.050432 ± 0.000177    |
| 15400                 | 0.050532 ± 0.000233            | 0.050564 ± 0.000256    | 0.050533 ± 0.000257     | 0.050565 ± 0.000266    | 0.050615 ± 0.000273    |
| 18200                 | 0.050572 ± 0.000348            | 0.050529 ± 0.000284    | 0.050515 ± 0.000288     | 0.050563 ± 0.000253    | 0.05059 ± 0.000255     |
| 21000                 | 0.050557 ± 0.000387            | 0.050578 ± 0.000381    | 0.050589 ± 0.0003875    | 0.050843 ± 0.003873    | 0.050903 ± 0.003902    |
| 23800                 |                                |                        |                         |                         | 0.051095 ± 0.003959    |

Fig. 5. Barplot comparison between different time intervals in two different networks where the error bars show the standard deviation. The average response time under different time intervals in the same network is very similar. (a) South America. (b) Asia.

Fig. 6. Comparison between adaptive and nonadaptive policies. (a) South America. (b) Asia.

rates are simulated in each TI. For the low workload setting, the combined request arrival rate from all switches is set to be 50% of the total control plane capacity while the arrival rate under high workload is 80%.

VII. NUMERICAL RESULTS

Results are presented in this section to demonstrate the effectiveness of our adaptive policy design and training.

A. Adaptive Policy

1) Adaptive Versus Nonadaptive: We first compare the performance of our adaptive policy with a nonadaptive policy as shown in Fig. 1. To demonstrate the effectiveness of the new policy design, the evaluation is done under an SA-DRL framework instead of MA-DRL. This is because the performance of MA-DRL depends not only on the policy design but also other factors, such as interagent cooperation and nonstationary environment handling. Therefore, to exclude the impact of such multiagent factors in the experiment, the comparison is conducted in the single-agent setting. Fig. 6 shows that both policies achieve similar performance. Note that, since the proposed adaptive policy has a smaller number of inputs and outputs for the DNN compared to the nonadaptive policy, our new policy design can reduce the DNN complexity without hurting the performance.

To demonstrate the adaptiveness of the proposed policy with respect to different numbers of controllers, the trained policy is evaluated in a network with a changing number of controllers (e.g., new controllers are added to match the request demand). As shown in Fig. 7, with the proposed new design, our policy network can flexibly support an elastic set of SDN controllers without retraining. Moreover, the use of a dispatching probability over all controllers effectively avoids the frequent activation of the policy network for every request, which ensures the policy efficiency and scalability.

2) Impact of Unreachable Controllers: As we discussed in Section IV-B, whenever a controller is identified as unreachable using the timeout mechanism, the switch will recompute the probabilities. If the capacity of nearby available controllers cannot handle the request demand, long-distance controllers will be included in the probability recalculation. Specifically, Table I reports the observed changes in response time when a certain number of controllers become unreachable in a network with eight controllers originally deployed. We can see that the difference in response time before and after controller failure is mostly negligible (<0.01 s), confirming that our method can effectively handle controller failures.

3) Impact of History Length and γ: Similar to [44], we investigate the influence of historical information and the discount factor γ on the performance, respectively.

The list of historical request arrival rates is contained in the agent’s observation, used as the policy input. Its length needs

11The nonadaptive policy uses the traditional policy design where the policy is directly represented as a DNN.
to be set properly. With a larger history length (e.g., 4), more information of the past is included in the agent’s observation, which provides a better approximation of a Markov state but requires more learning samples for the network to improve its performance. On the other hand, when the length is too small (e.g., 1), the response time stops reducing after 400 TI. Fig. 8 shows that the most suitable length is 3 for a good tradeoff between sampling costs and performance.

Fig. 9 shows the evaluation of the trained policies with different $\gamma$ under a broad range of request arrival rates over the two topologies. From Fig. 9, one can see that the policy with $\gamma = 0.9$ consistently achieves the lowest response time compared to policies with $\gamma = 0.5$ and $\gamma = 0.7$ in both topologies. This confirms our theory that the agent needs to consider the impact of its actions to prevent any controller from being overloaded, due to accumulated requests over a long run. Thus, for the remaining simulation studies, $\gamma$ is fixed to 0.9. We also observe that as the request arrival rate exceeds a certain value, the response time of all policies increases sharply regardless of the values of $\gamma$. This is because the control plane is highly loaded.

B. Policy Training

1) Single-Agent Versus Multiagent Training: To demonstrate the necessity of using MA-DRL for policy training in a multiagent environment, the policy trained by MA-PPO is compared with the single-agent training approach (denoted by SA-PPO-MA) as we discussed in Section V. SA-PPO-MA trains a policy and a value function on each agent independently using single-agent PPO. Both its training and testing performance is shown in Fig. 10. During the training process, we can observe a high variance in response time at the later TIs from Fig. 10(a), which implies that the learning fails to converge. This observation confirms the nonstationary environment issue when SA-DRL algorithms are used in a multiagent environment as we discussed in Section V. Correspondingly, during the testing process, we can also see from Fig. 10(b) that SA-PPO-MA can keep the response time at a low level when the request arrival rate is less than 14,000 pkts/s (e.g., the left dotted line LWL). However, it fails to avoid overloading controllers at a high request arrival rate (e.g., the right dotted line HWL). This is because SA-PPO-MA does not consider the impact of the other agents during the training. As the request arrival rate increases, the importance of agent cooperation becomes significant and the deficiency of SA-PPO-MA becomes evident.

Similarly, we also investigated the training and testing performance of MA-PPO. The learning curves in Fig. 11(a) shows that the response time can rapidly converge during the MA-PPO training process. Fig. 11(b) and (c) show that the policies obtained at the later TIs achieve lower response time compared to those obtained at the earlier TIs, which implies that MA-PPO can effectively improve the performance with the continued training of the policy. For example, the response time of the initialized policies (i.e., TI = 0) jumps from 90 ms to 2 s when the arrival rate reaches 19k pkts/s. This is because when the policy is randomly initialized, its behaviors
are similar to a randomized policy, which equally distributes requests among all controllers. Therefore, as the request arrival rate increases, controllers with low capacities are easily overloaded, resulting in high response time. In comparison, the policies obtained after 320 TIs can keep the response time below 1 s under the same request arrival rate. Apart from avoiding overloading controllers at high request arrival rates, the training also consistently reduces the response time when the request arrival rate is low.

We also compared the performance of the policies trained by MA-PPO and SA-PPO-MA, respectively, over the two network topologies. Fig. 12 confirms that policies trained by MA-PPO can effectively cope with the increasing number of requests through agent cooperation.

2) Comparison With Existing MA-DRL Algorithms: In the context of multiagent training, different multiagent algorithms [22], [23], [37] have been proposed. Among them, we compare MA-PPO with MADDPG [22] because MADDPG is closely related to MA-PPO and widely used. However, the deterministic policy gradient used in MADDPG cannot be calculated with respect to our adaptive policy network design. Nevertheless, we performed experiments using MADDPG to train policies over the network topologies where the number of controllers remains unchanged for both training and testing. As shown in Fig. 13, MA-PPO significantly outperforms MADDPG in both topologies. Specifically, MADDPG struggles to optimize each agent’s policy under a heavy workload. For example, the response time of the MADDPG-trained policies is at least 30 times longer than the MA-PPO trained policies in the South America topology when the request arrival rate is 192,000 pkts/s. This is mainly due to the ineffective $Q$-function training, which affects the quality of the trained policy. Given a network with $N$ agents and each agent takes an $M$-dimension action, MADDPG uses a centralized $Q$-function that requires the high-dimensional multiagent joint action with dimension $MN$ as its input. The high-dimensional input increases the $Q$-function complexity, often making the training ineffective.

C. Performance Comparison

1) MADRina Versus Heuristic-Based Approach: We compared MADRina with a widely used heuristic called weighted round robin (denoted by Weighted-RR) [3]. In Weighted-RR, request dispatching probabilities are proportional to controller capacities. The results are shown in Fig. 14. We can see that in both topologies [Fig. 14(b) and (d)], the response time of Weighted-RR remains stable because the number of requests dispatched to each controller is proportional to its capacity, which effectively prevents overloading any controller at an early stage. However, solely sending requests based on the controller capacity may not achieve the optimal network performance. Especially, when the control plane is not overloaded, dispatching more requests to a closer controller is a better option. Although the performance of Weighted-RR can be improved by fine-tuning its weights (i.e., dispatching probabilities), this process can be time consuming and requires domain knowledge. Moreover, the fine-tuned weights are usually problem specific and need to be manually reconfigured when the network setting changes. In MADRina, the relationship between network performance and request dispatching probabilities is learned during the interaction between the agents and the environment. Meanwhile, each agent in MADRina optimizes its scheduling policy to achieve the same goal of minimizing request response time. Therefore, we can see from both Fig. 14(b) and (d) that MADRina achieves a lower response time compared to Weighted-RR.
to optimize dispatching performance through a fully cooperative and partially observable MA-MDP based on the principle of centralized training and decentralized execution. To enable request dispatching across an elastic set of controllers, MADRina adopts a DNN-based adaptive policy, which adapts to an elastic set of controllers and establishes dispatching probabilities among the controllers. A dispatching system is built accordingly, filtering out overloaded and distant controllers for both training and dispatching. To facilitate the training of the policy, we proposed MA-PPO with a new method to estimate policy gradient. Through an extensive set of analysis, we demonstrate that MADRina can significantly reduce response time with improved practicability.

Note that recent research has been reported to use DRL in the SDN request dispatching domain [47], [48]. In line with the reported works, we have highlighted several potential research directions. For example, to ensure the performance of the production network and reduce the training cost, we adopted offline training in this work. However, due to the fact that it is impractical to train a policy to accommodate all situations the agent may see in the real world, any unexpected perturbations or unseen situations can potentially deteriorate the performance of the trained policy. To address this issue,
adapting the pretrained policy to the changes in the real world using online training techniques demands for more in-depth future studies. Meanwhile, this article mainly focused on the reactive flow setup mode. On the other hand, controllers in the separate proactive mode need to populate flow rules in switches in advance. To support this operation mode, it is important to investigate new hybrid strategies such that certain flow rules can be proactively installed in the controllers while maintaining the controllers flexibility of reactively handling incoming traffic. Furthermore, in an extreme situation when the current control plane capacity does not match the request demand, adaptive/dynamic controller deployment needs to be considered, which will be investigated in our future work. To encourage the widespread use of our system, it is important for our future research to further evaluate the performance of our multiagent system on a real-world testbed.

REFERENCES

[1] Z. Liu, Y. Cao, X. Zhang, C. Zhu, and F. Zhang, “Managing recurrent virtual network updates in multi-tenant datacenters: A system perspective,” IEEE Trans. Parallel Distrib. Syst., vol. 30, no. 8, pp. 1816–1825, Aug. 2019.

[2] C.-Y. Hong et al., “B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google’s software-defined WAN,” in Proc. Conf. ACM Spec. Interest Group Data Commun., 2018, pp. 74–87.

[3] V. Huang, Q. Fu, G. Chen, E. Wen, and J. Hart, “BLAC: A bindingless architecture for distributed SDN controllers,” in Proc. IEEE ICN, 2017, pp. 146–154.

[4] V. Sridharan, M. Gurusamy, and T. Truong-Huu, “Multi-controller traffic engineering in software defined networks,” in Proc. IEEE ICN, 2017, pp. 137–145.

[5] V. Huang, G. Chen, and Q. Fu, “Effective scheduling function design in SDN through deep reinforcement learning,” in Proc. IEEE ICC, 2019, pp. 1–7.

[6] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, “Learning scheduling algorithms for data processing clusters,” Proc. ACM Spec. Interest Group Data Commun., 2019, pp. 270–288.

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

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

[9] F. Li, J. Qin, and W. X. Zheng, “Distributed Q-learning-based online optimization algorithm for unit commitment and dispatch in smart grid,” IEEE Trans. Cybern., vol. 50, no. 9, pp. 4146–4156, Sep. 2020.

[10] I.-B. Park and J. Park, “Scalable scheduling of semiconductor packaging facilities using deep reinforcement learning,” IEEE Trans. Cybern., early access, Dec. 3, 2021, doi: 10.1109/TCYB.2021.3128075.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017, pp. 1–16.

[12] H. Huang, Z. Hu, Z. Lu, and X. Wen, “Network-scale traffic signal control via multiagent reinforcement learning with deep spatiotemporal attentive network,” IEEE Trans. Cybern., vol. 53, no. 1, pp. 262–274, Jan. 2023.

[13] V. Huang et al., “A scalable approach to SDN control plane management: High utilization comes with low latency,” IEEE Trans. Netw. Service Manage., vol. 17, no. 2, pp. 682–695, Jun. 2020.

[14] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, Cambridge, MA, USA: MIT Press, 2018.

[15] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” in Proc. NeurIPS, 2017, pp. 6379–6390.

[16] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, “A survey and critique of multiagent deep reinforcement learning,” Auton. Agents Multi-Agent Syst., vol. 33, no. 6, pp. 750–797, 2019.

[17] T. T. Nguyen, N. D. Nguyen, and N. Phan, “Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications,” IEEE Trans. Cybern., vol. 50, no. 9, pp. 3826–3839, Sept. 2020.

[18] Y. Zhang, Q. Yang, D. An, D. Li, and Z. Wu, “Multistep multiagent reinforcement learning for optimal energy schedule strategy of charging stations in smart grid,” IEEE Trans. Cybern., early access, Apr. 27, 2022, doi: 10.1109/TCYB.2022.3160974.

[19] H. Huang et al., “A scalable approach to SDN control plane management: High utilization comes with low latency,” IEEE Trans. Netw. Service Manage., vol. 17, no. 2, pp. 682–695, Jun. 2020.

[20] V. Huang et al., “A scalable approach to SDN control plane management: High utilization comes with low latency,” IEEE Trans. Netw. Service Manage., vol. 17, no. 2, pp. 682–695, Jun. 2020.

[21] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, Cambridge, MA, USA: MIT Press, 2018.

[22] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” in Proc. NeurIPS, 2017, pp. 6379–6390.

[23] J. Foerster, G. Farquhar, T. Afrousar, N. Nardelli, and S. Whiteson, “Counterfactual multi-agent policy gradients,” in Proc. AAAI, vol. 32, 2018, pp. 2974–2982.

[24] S. Li, Z. Li, and W. Zhang, “A dynamic association strategy for controller load balancing in software-defined networks,” in Proc. 23rd Asia-Pacific Netw. Oper. Manag. Symp. (APNOMS), 2022, pp. 1–6.

[25] L. Chen, F. Tang, and X. Li, “Mobility- and load-adaptive controller placement and assignment in LEO satellite networks,” in Proc. IEEE INFOCOM Conf. Comput. Commun., 2021, pp. 1–10.

[26] X. Ye, G. Cheng, and X. Luo, “Maximizing SDN control resource utilization via switch migration,” Comput. Netw., vol. 126, pp. 69–80, Oct. 2017.

[27] Y. Zhou, K. Zheng, W. Ni, and R. P. Liu, “Elastic switch migration for control plane load balancing in SDN,” IEEE Access, vol. 6, pp. 3990–3919, 2018.

[28] J. Cui, Q. Lu, H. Zhong, M. Tian, and L. Liu, “A load-balancing mechanism for distributed SDN control plane using response time,” IEEE Trans. Netw. Service Manage., vol. 15, no. 4, pp. 1197–1206, Dec. 2018.

[29] A. A. Z. Ibrahim, F. Hashim, N. K. Noordin, A. Sali, K. Navaie, and S. M. E. Fadul, “Heuristic resource allocation algorithm for controller placement in multi-control 5G based on SDN/NFV architecture,” IEEE Access, vol. 9, pp. 2602–2617, 2021.

[30] V. Sridharan, M. Gurusamy, and T. Truong-Huu, “On multiple controller mapping in software defined networks with resilience constraints,” IEEE Commun. Lett., vol. 21, no. 8, pp. 1763–1766, Aug. 2017.

[31] F. Al-Tam and N. Correia, “Fractional switch migration in multi-controller software-defined networking,” Comput. Netw., vol. 157, pp. 1–10, Jul. 2019.

[32] N. Moghadam, H. Li, H. Zeng, and L. Liu, “Lyapunov scheduling and optimization in network coded wireless multicast network,” IEEE Trans. Veh. Technol., vol. 67, no. 6, pp. 5135–5145, Jun. 2018.

[33] X. Wang, L. Ke, Z. Qiao, and X. Chai, “Large-scale traffic signal control using a novel multiagent reinforcement learning,” IEEE Trans. Cybern., vol. 51, no. 1, pp. 174–187, Jan. 2021.

[34] J. Zhao, T. Wang, W. Pedrycz, and W. Wang, “Granular prediction and dynamic scheduling based on adaptive dynamic programming for the blast furnace gas system,” IEEE Trans. Cybern., vol. 51, no. 4, pp. 2201–2214, Apr. 2021.

[35] V. Huang, C. Wang, H. Xu, G. Chen, and K. Christopher, “Cost-aware dynamic multi-workflow scheduling in cloud data Center using evolutionary reinforcement learning,” in Proc. 20th Int. Conf. Service Oriented Comput. (iSCOC), 2022, pp. 449–464.

[36] Z. Zhang, Y.-S. Ong, D. Wang, and B. Xue, “A collaborative multiagent reinforcement learning method based on policy gradient potential,” IEEE Trans. Cybern., vol. 51, no. 2, pp. 1015–1027, Feb. 2021.

[37] K. Son, D. Kim, W. J. Kang, D. E. Hostaller, and Y. Yi, “QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 5887–5896.

[38] C. Yu et al., “The surprising effectiveness of PPO in cooperative, multi-agent games,” 2021, arXiv:2103.01955.

[39] S. Hu and J. Hu, “Multiagent Policy Learning,” in Proc. Conf. ACM Spec. Interest Group Data Commun., 2021, pp. 2106.14334.
S. Abdallah and V. Lesser, “Multiagent reinforcement learning and self-organization in a network of agents,” in Proc. IJCAI, 2007, p. 39.

J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” in Proc. ICLR, 2016, pp. 1–14.

C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On calibration of modern neural networks,” in Proc. ICML, 2017, pp. 1321–1330.

K.-K. Yap et al., “Taking the edge off with espresso: Scale, reliability and programmability for global Internet peering,” in Proc. Conf. ACM Spec. Interest Group Data Commun., 2017, pp. 432–445.

N. Jay, N. H. Rotman, B. Godfrey, M. Schapira, and A. Tamar, “A deep reinforcement learning perspective on Internet congestion control,” in Proc. ICML, 2019, pp. 3050–3059.

F. Wilcoxon, “Individual comparisons by ranking methods,” in Breakthroughs in Statistics. New York, NY, USA: Springer, 1992, pp. 196–202.

N. Nachar, “The Mann-Whitney U: A test for assessing whether two independent samples come from the same distribution,” in Tut. Quantitative Methods Psychol., vol. 4, no. 1, pp. 13–20, 2008.

P. Sun, Z. Guo, G. Wang, J. Lan, and Y. Hu, “MARVEL: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning,” Comput. Netw., vol. 177, Aug. 2020, Art. no. 107230.

S. Yeo, Y. Naing, T. Kim, and S. Oh, “Achieving balanced load distribution with reinforcement learning-based switch migration in distributed SDN controllers,” Electronics, vol. 10, no. 2, p. 162, 2021.

Victoria Huang received the Ph.D. degree from the Victoria University of Wellington, Wellington, New Zealand, in 2021. She is currently a Data Scientist with the HPC and Data Science Department, National Institute of Water and Atmospheric Research, Wellington. Her research interests include reinforcement learning, evolutionary computation algorithms, resource scheduling, software-defined networking, and cloud computing.

Gang Chen (Senior Member, IEEE) received the Ph.D. degree from Nanyang Technological University, Singapore, in 2007. He is currently a Senior Lecturer with the School of Engineering and Computer Science and Centre for Data Science and Artificial Intelligence, Victoria University of Wellington, Wellington, New Zealand. He has more than 150 publications, including leading journals and conferences in machine learning, evolutionary computation, and distributed computing areas, such as IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, JAAMAS, ACM TRANSACTIONS ON AUTOMATED AND ADAPTIVE SYSTEMS, IEEE ICWS, and IEEE SCC. His research interests include evolutionary computation, reinforcement learning, multiagent systems, and cloud and service computing.

Dr. Chen is serving as the PC member of many prestigious conferences, including ICLR, ICML, NeurIPS, IJCAI, and AAAI, and the Co-Chair for Australian AI 2018 and CEC 2019.

Xingquan Zuo (Senior Member, IEEE) received the Ph.D. degree in Control Theory and Control Engineering from the Harbin Institute of Technology, Harbin, China, in 2004. He is currently a Professor with the School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China. He has published over 120 research papers in journals and conferences, and some papers were published in IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, and IEEE TRANSACTIONS ON CYBERNETICS, IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, and IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS. His research interests are in optimization and scheduling, artificial intelligence, data mining, and intelligent transportation systems.

Prof. Zuo currently serves as an Associate Editor for IEEE TRANSACTIONS ON CYBERNETICS AND ENGINEERING.

Albert Y. Zomaya (Fellow, IEEE) received the Ph.D. degree from Sheffield University, Sheffield, U.K., in 1990.

He is the Peter Nicol Russell Chair Professor of Computer Science with the School of Computer Science, The University of Sydney, Camperdown, NSW, Australia, and serves as the Deputy Dean of the School of Computer and Information Sciences. His research is focused on content delivery networks, traffic analysis and security, scheduling, software-defined networking, and cloud computing. Prof. Zomaya is the Editor-in-Chief of the ACM Computing Surveys and serves as an associate editor for several leading journals. He is a Decorated Scholar with numerous accolades including Fellowship of the IEEE, AAS, and the IET. Also, he is a Fellow of the Australian Academy of Science, the Fellow of the Royal Society of New South Wales, the Foreign Member of Academia Europaea, and a member of the European Academy of Sciences and Arts.

Dr. Zomaya is the Editor-in-Chief of the ACM Computing Surveys and serves as an associate editor for several leading journals. He is a Decorated Scholar with numerous accolades including Fellowship of the IEEE, AAS, and the IET. Also, he is a Fellow of the Australian Academy of Science, the Fellow of the Royal Society of New South Wales, the Foreign Member of Academia Europaea, and a member of the European Academy of Sciences and Arts.

Nasrin Sohrabi received the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 2022.

She is currently a Postdoctoral Research Fellow of Cloud, Systems, and Security Discipline with the School of Computing Technologies, RMIT University. Her expertise is in the areas of scalability and performance of distributed systems, consensus algorithms, and systems’ security.

Zahir Tari received the Ph.D. degree from the University of Grenoble, Grenoble, France, in 1990.

He is a Full Professor of Distributed Systems with RMIT University, Melbourne, VIC, Australia, and the Research Director of the Centre of Cyber Security Research and Innovation. His main expertise is in the areas of system’s performance (e.g., F2P, cloud, and IoT) as well as system’s security (e.g., SCADA, SmartGrid, Cloud, and IoT). Dr. Tari is or has been an Associate Editor of ACM Computing Surveys, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and IEEE Cloud Computing.

Qiangu Fu received the Ph.D. degree from The University of Queensland, Brisbane, QLD, Australia, in 2005.

He is currently a Senior Lecturer of Cloud, Systems, and Security with the School of Computing Technologies, RMIT University, Melbourne, VIC, Australia. His research interests are broadly in the areas of Internet and cloud-based systems, including wireless and mobile systems. In particular, he has a focus on content delivery networks, traffic analysis over data centers, virtualization as well as network programmability.