Cooperative Multi-Agent Deep Reinforcement Learning for Dynamic Virtual Network Allocation With Traffic Fluctuations

Akito Suzuki, *Member, IEEE*, Ryoichi Kawahara, *Member, IEEE*, and Shigeaki Harada, *Member, IEEE*

***Abstract*—Network traffic and computing demand have been changing dramatically due to the growth of various types of network services, e.g., high-quality video delivery and operating system (OS) updates. To maximize the utilization efficiency of limited network resources, network resource control technology is required for smooth and quick operation when network demands change. Therefore, we propose a dynamic virtual network (VN) allocation method based on cooperative multi-agent deep reinforcement learning (Coop-MADRL). This method can quickly optimize network resources even while network demands are drastically changing by learning the relationship between network demand patterns and optimal allocation by using deep reinforcement learning (DRL) in advance. The key idea is to use a multi-agent technique for a reinforcement learning (RL) based dynamic VN allocation method, which can reduce the number of candidate actions per agent and can improve the performance for VN allocation. Moreover, a cooperation technique improves the efficiency of VN allocation. From results of a simulation evaluation, Coop-MADRL can calculate effective allocation within 1 s, which reduces the maximum server and link utilization and drastically reduces the constraint violations compared with that of the static VN allocation method. Furthermore, we revealed that the learning with various mixed traffic models could achieve a high generalization performance for all traffic patterns.**

***Index Terms*—Network functions virtualization, Reinforcement Learning, Network Control, Dynamic Resource Allocation.**

I. Introduction

NETWORK functions virtualization (NFV) [2] is one of the key technologies of future networks. NFV has emerged as an innovative network paradigm that abstracts the network functions from hardware. NFV is closely related to other emerging technologies, such as Software Defined Networking (SDN) [3]. SDN is a networking technology that decouples the control plane from the underlying data plane and allows programmatic and centralized resource management of network functions. Combining SDN and NFV will enable to provide the complex network services in next-generation networks through centralized network management by SDN and specific abstraction and isolation mechanisms by NFV.

NFV enables multiple virtual network (VN) requests to be shared on the same physical network. A VN is represented by a set of virtual nodes and virtual links. A virtual node indicates the server resource requirements such as the required number of central processing units (CPUs) and amount of random access memory (RAM) and is often treated as a unit of a virtual machine (VM). A virtual link indicates the network resource requirements such as the required bandwidth and delay between virtual nodes. To maximize the resource utilization efficiency of limited physical resources, the mapping of VN requirements to a physical network needs to be optimized. The optimal mapping refers to the allocation that maximizes the objective function (e.g., resource utilization) while satisfying constraints (e.g., resource capacity). The performance of VN allocation algorithms determines the efficiency of the overall network resources, making it a critical part of NFV technology.

The problem of finding an optimal VN allocation is known as the virtual network embedding (VNE) problem. Most existing approaches [4]–[6] only focus on static VN allocation, where the amount of VN demand for resources is unchanged over time. When a VN is embedded at once in the physical network, the VN requests will hold fixed resources until the end of their lifetime in the static embedding process. However, since network traffic and computing demand have been changing dramatically due to the various types of network services, e.g., high-quality video delivery and operating system (OS) updates, the VN demands are dynamically changing and fluctuating. In the above situation, the static VN allocation leads to resources being inefficiently utilized and/or networks becoming congested, so the dynamic VN allocation for time-varying demand becomes more important. To more efficiently utilize resources, this paper attempts to solve the dynamic VN allocation problem. Although the dynamic VN allocation problem has been studied for more than a decade, the following difficulty remains unresolved.

The difficulty in optimizing dynamic VN allocation is the need to simultaneously allocate VNs efficiently and immediately, even though efficiency and immediacy are in a trade-off relationship. Dynamic VN allocation needs to increase the computation time to increase the efficiency of VN allocation.

Manuscript received 2 June 2021; revised 29 November 2021 and 18 January 2022; accepted 31 January 2022. Date of publication 7 February 2022; date of current version 12 October 2022. This journal paper is an extension of the conference version, which has been accepted by ICCCN 2021 [DOI: 10.1109/ICCCN52240.2021.9522308]. The associate editor coordinating the review of this article and approving it for publication was T. Zinner. *(Corresponding author: Akito Suzuki.)*

Akito Suzuki and Shigeaki Harada are with the NTT Network Service Systems Laboratories, NTT Corporation, Tokyo 180-8585, Japan (e-mail: akito.suzuki.tw@hco.ntt.co.jp; shigeaki.harada.fy@hco.ntt.co.jp).

Ryoichi Kawahara is with the Faculty of Information Networking for Innovation and Design, Toyo University, Tokyo 115-8650, Japan (e-mail: ryoichi.kawahara@iniad.org).

Digital Object Identifier 10.1109/TNSM.2022.3149243

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

The increase in computation time directly causes an increase in the control period. Alternatively, dynamic VN allocation requires a short control period to keep up with changes in demand. That is, increasing the efficiency leads to decreasing the immediacy. Similarly, since allocation performance decreases when the control period is limited, increasing immediacy leads to decreasing efficiency. Therefore, many methods have failed to simultaneously achieve efficiency and immediacy.

Several studies have addressed this problem [7]–[14]. Specifically, reinforcement learning (RL) [15], [16] has been focused on as a solution [7]–[12]. RL solves the decision problem of what **action** an agent should take by observing the current **state** within a certain environment. An agent receives a **reward** from the environment depending on the selected action and learns a policy (i.e., strategy) that maximizes the received reward through a series of selected actions. RL is expected to be able to immediately output a close-to-optimal VN allocation even as the network resource demand drastically changes by learning the relationship between resource demand patterns and optimal VN allocation in advance.

The main challenge in applying RL to dynamic VN allocation is related to the number of candidate actions when solving the combinatorial optimization problem by RL. The RL approach relies on exploring all available actions sufficiently to compute a policy close to the optimal. Since the number of ways a VN can be embedded is combinatorial, the candidate actions of VN allocation exponentially increase as the number of nodes and the number of links increase. Therefore, the RL approach potentially requires a huge number of actions to derive an appropriate solution, which could lead to a prohibitively long convergence time for the learning process [17]. It was also reported that the performance of RL drastically worsens as the number of candidate actions increases [18]. Thus, the action space of the VNE problem needs to be shrunk.

We propose a dynamic VN allocation algorithm based on cooperative multi-agent deep reinforcement learning (**Coop-MADRL**). The key idea is to use a multi-agent technique for an RL-based dynamic VN allocation method. We prepare each agent for each VN allocation control, which can reduce the action space. However, when training decentralized independent agents to optimize for the team reward, each agent is faced with a non-stationary learning problem, i.e., the dynamics of its environment change as other agents change their behaviors through learning [19]. An example of such a non-stationary problem is that when each agent independently acts at the same time, all VMs will be allocated on the smallest load server, resulting in server overload. Therefore, we introduce a cooperative element in which several agents jointly optimize a single reward through *centralized training and decentralized execution*, which can improve the efficiency of VN allocation. Previous methods [10]–[12] avoid the problem of exponentially increasing action space by restricting the agents' actions at each step or compressing the number of actions by handcrafted features (see Section II for details). The proposed method directly treats all candidate actions at each step without restricting agent actions or modeling and compressing handcrafted features for states and actions. This cooperation can avoid conflicts of control between agents and improve the performance of VN allocation. Moreover, it can take the hassle out of problem-specific feature design.

The main contributions can be summarized as follows.

- We formulate the dynamic VN allocation problem to find an optimal VN allocation for time-varying VN demands. This problem aims to find the optimal allocation consisting of VM allocations and the routing between users and VMs. Here, we define the optimal allocation as a solution that maximizes server and link resource utilization efficiency and minimizes the number of migrated VMs while satisfying server and link capacity constraints.
- We propose a dynamic VN allocation algorithm based on Coop-MADRL. We introduce a cooperative multi-agent technique for a deep RL (DRL)-based dynamic VN allocation method. This technique can avoid control conflicts between agents and improve the performance of VN allocation without restricting agent actions or modeling and compressing handcrafted features for states and actions.
- We evaluated the effectiveness of the proposed method through simulations in terms of performance, computation time, and scalability for the number of VNs and network topology size. Simulations revealed that our method can reduce the maximum server and link utilization and drastically reduce the constraint violations compared with a static VN allocation method under practical-network conditions.
- We also evaluated the generalization performance for unknown traffic patterns. The results showed that the agent training with mixed various traffic models could achieve a high generalization performance for all traffic models. To the best of our knowledge, this is the first paper to evaluate the generalization performance of traffic patterns for the RL-based network control method.

This journal paper is an extension of the conference version [1]. The main extension in this journal version is a comprehensive evaluation of the proposed method. In particular, we evaluate the generalization performance for various traffic demand patterns. We also extend the penalty function of VM migration in such a way that the function depends on the number of VMs migrated each time, which allows for more realistic allocation.

The rest of the paper is organized as follows. Section II describes related work. Section III defines the dynamic VN allocation problem. Section IV briefly reviews RL, and Section V describes the Coop-MADRL-based dynamic VN allocation method and its modeling and formulation. Section VI evaluates its performance, and Section VII concludes the paper.

II. Related Work

There have been several studies on dynamic VN allocation (RL-based [7]–[12] and heuristic [13], [14]).

A. RL-Based Dynamic VN Allocation

Mijumbi *et al.* [7]–[9] proposed a multi-agent RL-based dynamic bandwidth control method under the decentralized resource management system, which prepares the agents for

each physical node and physical link. They [7] applied a Q-learning based RL agent. They [8] also used an artificial network to make resource reallocation decisions and train the network with a Q-table. They [9] also proposed an RL-based neuro-fuzzy algorithm. However, it controls only the buffer size of virtual nodes and the bandwidth of virtual links and does not reallocate virtual nodes. Therefore, it cannot cope with demand changes for server resources. Moreover, to prevent the number of actions from exponentially increasing, all the demands passing through each node and link are controlled by uniform parameters. Therefore, the bandwidth for all users is limited regardless of the demand of each user.

With the breakthrough of DRL in human-level control applications [16], the DRL-based VNE algorithm has been increasingly studied. Studies have mentioned the problem of the RL-based approach: the candidate actions of VN allocation exponentially increase as the number of nodes and the number of links increase [10]–[12]. Dolati *et al.* [10] attempted to shrink the action space of the VNE problem to provide sufficient flexibility for exploring different VN mappings while retaining the efficiency of the learning process. They adopted a convolutional neural network (CNN) for a DRL approach in solving VNE problems. Dolati *et al.* assumed that networks have grid-like typologies to make the image representation easier to obtain, but this is not true in many other situations. Yan *et al.* [11] decompose a VNE process into a sequence of virtual node embedding to shrink the action space, and the learning agent only focuses on one virtual node of the current VN request at every single step. We previously proposed a MADRL-based dynamic VN allocation method [12]. We divided the VN demands into groups, and each agent is prepared for each group, which can shrink the action space per agent. We also restricted the agents that could act at each time to avoid conflicts among agents. However, the shrinking action space and the restriction on the agent's action decrease the performance of VNE. Moreover, this restriction increases the number of steps required for VN reallocation and may delay the response to dynamic demand changes. Conversely, this paper adds a cooperative element to our previous method, which can avoid conflicts of control between agents and improve the performance of VN allocation.

In conclusion, we summarize the shortcoming of existing methods and the strengths of our method. As mentioned above, the problem of RL-based approaches is that the number of candidate actions for VN allocation increases exponentially as the number of nodes and links increases. Existing methods introduce the multi-agent technique to prevent the exponential increase of actions. However, existing methods restrict the agent's actions at each step or compress the action space using handcrafted features, which degrades the performance of VN allocation. On the other hand, we additively introduce a cooperative technique for a multi-agent DRL-based dynamic VN allocation method. This technique can improve the performance of VN allocation without restricting agent actions or compressing actions. Furthermore, we aim to optimize the routing between users and VMs for time-varying traffic demands. We also evaluate the generalization performance of traffic patterns. These perspectives are not considered in existing methods.

B. Heuristic Dynamic VN Allocation

Zheng *et al.* [13] proposed a dynamic VNE algorithm based on a radial basis function (RBF) neural network to learn and predict the dynamic changes of resources, and users dynamically adjust and allocate resources by the predicted results. Although these proactive control methods based on predicting the dynamic changes are effective for near-stationary resource demand, they sometimes fail since drastic demand changes cannot be predicted. On the other hand, our approach uses the almost real-time demand for control by drastically reducing the computation time and control interval, not minimizing the prediction error.

Dehury and Sahoo [14] proposed a dynamic VN allocation method that combines online and offline allocation algorithms. The VN demand is accepted and allocated immediately by the online algorithm, then VN demand is periodically reallocated by the offline algorithm. By combining the two algorithms, they simultaneously achieved immediate demand acceptance and optimal demand allocation. However, they assumed only adding and deleting virtual links and not the change in virtual link bandwidths. Therefore, their method cannot be applied for time-varying traffic demand.

III. Dynamic Virtual Network Allocation

A. Problem Definition

This paper addresses the dynamic VN allocation for time-varying demand. Each VN consists of server demands as virtual nodes and traffic demands as virtual links. In this paper, dynamic VN is defined as the time-varying resource requirement of virtual nodes and virtual links, and dynamic VN allocation is defined as the resource allocation for time-varying VN demands. In static allocation, each user estimates the maximum amount of demand during their lifetime in advance and pays a fixed fee based on the estimated amount. The service provider allocates the resources for the demands to maximize the VNE acceptance rate. Once accepted, the allocation is fixed until the end of the service. This provision may not be optimal when demand fluctuates. In contrast, in dynamic allocation, each user pays a minimal fee corresponding to the resources consumed at each time. The service provider dynamically reallocates the resources in accordance with the demand at each time to maximize resource utilization efficiency, which can decrease the service cost per user and minimize the negative effect of network congestion and server overload. Moreover, the dynamic allocation is made robust to sudden changes in demands by maintaining high utilization efficiency each time.

Figure 1 describes an overview of dynamic VN allocation. For an example of a use case, we consider providing a cloud computing service in a physical network consisting of a wide-area network (WAN) and data centers (DCs). To provide such a service, each user requests a VN demand, which consists of server demands as VMs and traffic demands between the user and VMs, and each VN demand needs to be allocated to a physical network. For simplicity, Fig. 1 describes a case where

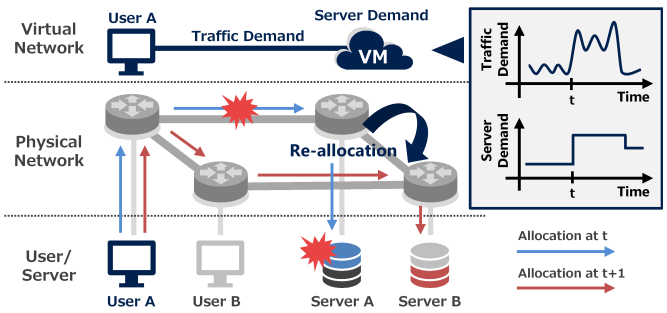


Fig. 1. Overview of Dynamic VN Allocation.

the number of VNs is 1, and a VN consists of one user, one VM, and the links between them. A physical network consists of 4 nodes including 2 user nodes and 2 server nodes. Here, the nodes connected to the user and server are called user nodes and server nodes, respectively. Some nodes can perform as both user and server nodes. We consider a discrete time-step t and assume the VN demands change at each t. As shown in the graph in Fig. 1, we assume that the two kinds of demands are time-varying, and an unexpected heavy demand suddenly emerged at time t, thereby causing the link congestion and the overload on server A. In dynamic allocation, after observing the current demands and calculating the next allocation, the VN is reallocated at $t + 1$. In this example, the VM is migrated to server B, and the route is switched from a blue one to a red one.

This paper focuses on immediately calculating a close-to-optimal VN allocation every t. We use the offline VN allocation, where all VN demands are given at the beginning of each t. We consider K VN demands and assume that K is constant during a series of steps. When the number of VNs are changed during a series of steps, the maximum number of VNs is regarded as K. Here, a series of steps refers to the steps until the number of VNs exceeds K. When the actual number of VNs exceeds K, the RL agent needs to be retrained. At the beginning of each t, the VN demands are observed. On the basis of the observation, our method calculates the next VN allocation for the next $t + 1$. Note that this next allocation is calculated on the basis of the current observation and does not directly predict the next $t + 1$ observation. Since our method responsively allocates the demands after observation, it must immediately calculate a close-to-optimal allocation to follow the change, e.g., within 1 s. This target computation time is very challenging and different from those of existing methods. Next, if necessary, controllers update the routing information and migrate each VM. After the update is complete, our method proceeds to the next $t + 1$.

Note that we assume the migration process is ideal, which means that the VM can be migrated in a short time without interrupting the running service. The way to accomplish the ideal migration is out of the scope of this paper and may be considered in future research. In related work, the live migration of VM has been widely studied [20]. Moreover, the development of lightweight VMs has notably progressed. Firecracker [21], an open-source virtualization technology, enables us to deploy workloads in lightweight VMs, called microVMs. It can boot application code within 125 ms with a memory overhead of less than 5 MB per container and has been commonly used in cloud computing. We assume that, as virtual network functions (VNFs) evolve into cloud-native network functions (CNFs) in the future, nearly ideal migration will be made possible by utilizing these techniques.

TABLE I
SYMBOL DESCRIPTIONS FOR PHYSICAL NETWORK

Symbols	Definitions
$G(\boldsymbol{N}, \boldsymbol{L})$	Network graph
$n \in \boldsymbol{N}$	Node
$s \in \boldsymbol{S}$	Server
link $(i, j) \in \boldsymbol{L}$	Link from node i to node j
c_i^S	Server capacity of server i
c_{ij}^L	Link capacity of link (i, j)
$\boldsymbol{Z} := \{z_{ij}\}$	User placement (user i, node j)

TABLE II
SYMBOL DESCRIPTIONS FOR VN DEMANDS

Symbols	Definitions
$t \in T$	Time-step (T: Total time steps)
K	Number of VNs
$\boldsymbol{B}_t := \{b_t^i\}$	Traffic demands of i^{th} VN at step t
$\boldsymbol{D}_t := \{d_t^i\}$	VM demands of i^{th} VN at step t

B. Problem Formulation

Table I summarizes the definitions of the physical network variables. We assumed that the physical network graph $G(\boldsymbol{N}, \boldsymbol{L})$ consists of a physical node set $\boldsymbol{N}$ and a physical link set $\boldsymbol{L}$. Each node connected to the user and server is called user nodes and server nodes, respectively. For example, the nodes directly connected to the data center are called server nodes, and the nodes directly connected to the access network are called user nodes. Some nodes can perform as both user and server nodes. Each server node has a server connected to it. We denote the server as $s \in \boldsymbol{S} \subseteq \boldsymbol{N}$. All servers and links have the capacities c_i^S and c_{ij}^L, which indicate the limit of computing resources and bandwidth resources. Each user connects to the nearest user node through the access network, which is not included in $G(\boldsymbol{N}, \boldsymbol{L})$ in this paper. User placement is defined as $\boldsymbol{Z} := \{z_{ij}\}$, in which z_{ij} is 1 if the i^{th} user is connected to the j^{th} node; otherwise, 0. Here, $\boldsymbol{Z}$ is assumed to be constant during the VN demand lifetime.

Table II summarizes the definitions of the VN demand variables. A VN demand consists of one origin (i.e., user) and one destination (i.e., VM), user placement $\boldsymbol{Z}$, traffic demand $\boldsymbol{B}_t := \{b_t^i\}$, and VM demand $\boldsymbol{D}_t := \{d_t^i\}$. The i^{th} traffic demand b_t^i indicates the bandwidth between the user and VM at t, and the i^{th} VM demand d_t^i indicates the processing power of the VM request at t such as the number of CPU cores. When each VN demand is accepted, the amount of link and server resources consumed depend on the traffic and VM demand. If an origin-destination (i.e., user-VM) pair is allocated in the same server, the traffic demand between the user and VM on the physical network is regarded as 0.

TABLE III
SYMBOL DESCRIPTIONS FOR CONTROL VARIABLES

Symbols	Definitions
$\boldsymbol{X}_t := \{x_{ij,t}^{pq}\}$	Proportion of passed τ_t^{pq} on link (i, j)
$\boldsymbol{Y}_t := \{y_{ij,t}\}$	VM allocation at step t (VM i, server j)

TABLE IV
SYMBOL DESCRIPTIONS FOR DYNAMIC VN ALLOCATION

Symbols	Definitions
$w \in \boldsymbol{W}$	User
$v \in \boldsymbol{V}$	VM
$\tau_t := \{\tau_t^{pq}\}$	Traffic matrix from node p to node q
P_t	Penalty Function of VM migration
$u_{ij,t}^L$	(i, j) link utilization at step t
$u_{i,t}^S$	i^{th} server utilization at step t
$U_t^L = \max_{ij}\left(u_{ij,t}^L\right)$	Maximum link utilization at step t
$U_t^S = \max_i\left(u_{i,t}^S\right)$	Maximum server utilization at step t
$\boldsymbol{R}_t^L := \left\{r_{ij,t}^L\right\}$	(i, j) residual link resources at step t
$\boldsymbol{R}_t^S := \left\{r_{i,t}^S\right\}$	i^{th} residual server resources at step t

We mainly adopted the VN demand model consisting of one origin and one destination. The problem formulation can be extended to the VN demand model with multiple origins and destinations. In this case, we newly define $\tilde{\boldsymbol{B}}_t := \{\tilde{b}_t^{ij}\}$ as the traffic demands between i^{th} user and j^{th} VM at step t, and defined $\tilde{\boldsymbol{D}}_t := \{\tilde{d}_t^i\}$ as the VM sizes of i^{th} VM at step t. The problem formulation and proposed method can be used directly by replacing $\boldsymbol{B}_t$ and $\boldsymbol{D}_t$ with $\tilde{\boldsymbol{B}}_t$ and $\tilde{\boldsymbol{D}}_t$.

Note that, though the VN model in this paper is assumed to consist of a single VM, it can be extended to more complex VN models consisting of a graph with multiple VMs and virtual link(s) by using an extendable NFV-integrated control architecture [22]. They [22] define 12 types of VN models and describe how to extend the formulation of the RL-based VN allocation algorithm when changing the VN model from one model to the other models. For example, they describe how to extend the VN model consisting of one VM to the VN model consisting of the chain of multiple VMs assumed in the use case of service function chaining (SFC) (see [22] for details). Though this formulation can be extended to more complex VN models in principle, its performance has not been evaluated yet, and its evaluation is one of the future works.

For the above physical network graph and VN demand, we formulate the dynamic VN allocation problem. Table III summarizes the definitions of the control variables. The goal of this problem is to find an optimal VN allocation consisting of $\boldsymbol{X}_t$ and $\boldsymbol{Y}_t$ every t. Here, $\boldsymbol{X}_t := \{x_{ij,t}^{pq}\}$ shows the proportion of traffic τ_t^{pq} from origin node p to destination node q passing through link (i, j), and $\boldsymbol{Y}_t := \{y_{ij,t}\}$ shows the VM allocation in which $y_{ij,t}$ is 1 if the i^{th} VM is assigned to the j^{th} server; otherwise, 0.

Table IV summarizes the definitions of the variables of the dynamic VN allocation problem. We introduce an objective function:

$$\min \ : \ \sum_{t\in T}\left(U_t^S + U_t^L + \alpha P_t\right), \tag{1}$$

where U_t^S and U_t^L show the maximum server utilization and maximum link utilization at t, and P_t shows the penalty of VM migration. Here, α is a positive value and determines the degree of VM migration. We set the penalty to depend on the number of migrated VMs, which is formulated as follows.

$$P_t = \frac{1}{2}\sum_{i\in V}\sum_{j\in S}\left\|y_{ij,t} - y_{ij,t-1}\right\| \tag{2}$$

Equation (2) calculates the sum of the absolute value of the difference between current $\boldsymbol{Y}_t$ and previous $\boldsymbol{Y}_{t-1}$. Since $y_{ij,t}$ and $y_{ij,t-1}$ are binary numbers, the sum of the absolute values of these differences indicates the number of migrated VMs at t. We also impose two constraints: link capacity and server capacity, i.e., $s.t. : U_t^S < 1$ and $U_t^L < 1$. To formulate this, a VM allocation variable $y_{ij,t}$ is formulated to minimize the server utilization U_t^S while satisfying the constraints as follows.

$$\text{s.t.} : \sum_{j\in \boldsymbol{S}} y_{ij,t} = 1 \ (\forall i \in \boldsymbol{V}) \tag{3}$$

$$\sum_{i\in \boldsymbol{V}} d_t^i y_{ij,t} \le c_j^S U_t^S \ (\forall j \in \boldsymbol{S}) \tag{4}$$

$$y_{ij,t} \in \{0, 1\} \tag{5}$$

$$0 \le U_t^S \le 1 \tag{6}$$

Equation (3) shows the VM conservation law. In other words, it shows that each VM must be allocated to any server. Equation (4) shows the constraint of server capacity, and Eqs. (5)–(6) show the range of variables. In addition, a routing variable $x_{ij,t}^{pq}$ is formulated to minimize the link utilization U_t^L while satisfying the constraints as follows.

$$\text{s.t.:} \sum_{j:(i,j)\in \boldsymbol{L}} x_{ij,t}^{pq} - \sum_{j:(j,i)\in \boldsymbol{L}} x_{ji,t}^{pq} = 0 \tag{7}$$

$$(\forall p, q \in \boldsymbol{N}, i \ne p, i \ne q)$$

$$\sum_{j:(i,j)\in \boldsymbol{L}} x_{ij,t}^{pq} - \sum_{j:(j,i)\in \boldsymbol{L}} x_{ji,t}^{pq} = 1 \tag{8}$$

$$(\forall p, q \in \boldsymbol{N}, i = p)$$

$$\sum_{p,q\in \boldsymbol{N}} \tau_t^{pq} x_{ij,t}^{pq} \le c_{ij}^L U_t^L \tag{9}$$

$$(\forall (i, j) \in \boldsymbol{L}, \forall p, q \in \boldsymbol{N})$$

$$0 \le x_{ij,t}^{pq} \le 1 \ (\forall (i, j) \in \boldsymbol{L}, \forall p, q \in \boldsymbol{N}) \tag{10}$$

$$0 \le U_t^L \le 1 \tag{11}$$

Equations (7)–(8) show the traffic flow conservation law. Equation (7) shows that the traffic flowing into a node equals the traffic flowing out of the node except the source node p and destination node q. Equation (8) shows that the net flow out of the source node p is 1. The traffic flow conservation law at the destination node q is guaranteed when Eqs. (7)–(8) are satisfied, which is proved in [23]. Equation (9) shows the constraint of link capacity, and Eqs. (10)–(11) show the range of variables. Since traffic demands between nodes τ_t^{pq} in Eq. (9) are determined by the traffic demands $\boldsymbol{B}_t$, user placements $\boldsymbol{Z}$, and VM allocation $\boldsymbol{Y}_t$, the relational equations between both constraints can be formulated

as follows.

$$\boldsymbol{\tau}_t = \boldsymbol{Z}^\top \boldsymbol{B}_t^\top \boldsymbol{Y}_t$$
$$\tau_t^{pq} = \sum_{i \in \boldsymbol{W}} \sum_{j \in \boldsymbol{V}} z_{ip}\, b_t^i\, y_{jq,t} \tag{12}$$

Here, we define a traffic matrix as $\boldsymbol{\tau}_t := \{\tau_t^{pq}\}$ and each τ_t^{pq} is a positive real number. Equation (12) converts the VN traffic demands into the traffic matrix of the physical network.

The dynamic VN allocation problem is formulated to minimize Eq. (1) and the constraints Eqs. (3)–(12). This problem is NP-hard [24] and categorized mixed-integer nonlinear problems. Since the optimal solution takes a long time to calculate, most previous studies have targeted small-scale or static VN demands. Conversely, in this paper, we use RL to solve the problem. Whereas the methods not based on RL take a long time to find the optimal solution each time, RL can instantly output the close-to-optimal solution by learning the relationship between resource demand patterns and optimal VN allocation in advance. However, RL needs to efficiently learn the optimal allocation for a wide variety of demand patterns to improve performance. Therefore, we develop a cooperative multi-agent technique for an RL-based method to prevent VN allocation decisions from exponentially increasing.

IV. Reinforcement Learning

A. Single-Agent Reinforcement Learning

Single-agent RL considers a sequential decision-making problem in which an agent interacts with an environment. The agent observes state $s \in \mathcal{S}$ in which $\mathcal{S}$ is the state space, takes action $a \in \mathcal{A}$ where $\mathcal{A}$ is the action space, and executes it in the environment to receive reward r and transfer to the new state $s' \in \mathcal{S}$. The goal of the agent is to determine a policy that maximizes the long-term reward. The policy is a map $\pi : \mathcal{S} \to P(\mathcal{A})$, where P is the transition probability among the states. Q-learning [15], a widely used RL algorithm, learns the relationship of $\langle s, a, r, s' \rangle$ to maximize the action value $Q(s, a)$, which is defined as the expectation of the sum of rewards obtained in the future when action a is selected in state s. The agent receives a reward r and updates the Q-function in accordance with the following equation.

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha \left[r + \gamma \max_{a' \in \mathcal{A}} Q\left(s', a'\right) \right], \tag{13}$$

where α is a learning rate and γ is a discount factor.

DRL [16], [18] dramatically improves the generalization and scalability of traditional RL algorithms and can handle continuous and high-dimensional state space by approximating the $Q(s, a)$ with a deep neural network (DNN). Deep Q-network (DQN) [16] uses a *replay memory* to store the transition tuple $\langle s, a, r, s' \rangle$. DNN parameters θ are learned by sampling batches b of transitions from the replay memory and minimizing the squared error:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(y_i^{\mathrm{DQN}} - Q(s,a;\theta) \right)^2 \right], \tag{14}$$
$$y^{\mathrm{DQN}} = r + \gamma \max_{a' \in \mathcal{A}} Q\left(s', a'; \theta^-\right), \tag{15}$$

where θ^- are the parameters of a *target network* that are periodically copied from θ and kept constant for several iterations. It was reported that the performance of RL algorithms drastically worsens as the number of candidate actions increases due to the decrease in the sampling efficiency of $\langle s, a, r, s' \rangle$ and increase in the error of the DNN [18]. Therefore, stable DRL becomes difficult as the number of actions increases.

B. Multi-Agent Reinforcement Learning

Multi-Agent RL (MARL) is a system of multiple agents interacting within a common environment. Each agent decides each time-step and works together with the other agent(s) to achieve a given goal. It is used for learning a complex environment by dividing a single task into multiple sub-tasks. The learning cost of each agent can be reduced by assigning each agent to each task. Due to the complexities of the environments and the combinatorial nature of the problem, most MARL problems are categorized as NP-hard problems [25].

A cooperative multi-agent environment can be described as a decentralized partially observable Markov decision process (Dec-POMDP) [26] consisting of $\langle K, \mathcal{S}, \mathcal{A}, R, P, \boldsymbol{O}, \gamma \rangle$, in which K is the number of agents, $\mathcal{S}$ is state space, $\mathcal{A} = \{\mathcal{A}^1, \ldots, \mathcal{A}^K\}$ is the set of actions for all agents, P is the transition probability among the states, R is the reward function, and $\boldsymbol{O} = \{\boldsymbol{O}^1, \ldots, \boldsymbol{O}^K\}$ is the set of observations for all agents. In a cooperative multi-agent problem in which the environment can be fully observed, the i^{th} agent at time-step t observes the global state s_t, takes action a_t^i ($\boldsymbol{a}_t = \{a_t^i\}$), and receives reward r_t. If the agents cannot observe the global state, each agent only accesses its own local observation o_t^i. The state s_t needs to contain all information required to uniquely represent the current environment's status. Whereas the observation o_t^i is a part of state s_t, and it needs to contain the information required to uniquely represent i^{th} agent's current status.

Each agent has an observation-action history $h^i \in \boldsymbol{h}$, where the history indicates a series of past observations and $\boldsymbol{h}$ is the observation-action history of all agents. The joint policy is a map $\pi : \boldsymbol{h} \to P(\mathcal{A})$, and the joint policy π has a joint action-value function:

$$Q^\pi(\boldsymbol{h}_t, \boldsymbol{a}_t) = \mathbb{E}\left[\sum_{j=0}^{\infty} \gamma^j r_{t+j} \mid \boldsymbol{h}_t, \boldsymbol{a}_t \right]. \tag{16}$$

Since the dynamic VN allocation deals with discrete actions, we describe a representative method of a value function factorization based MARL algorithm with discrete actions [27], and our Coop-MADRL algorithm is based on these algorithms.

1) Independent Q-Learning: The most naive approach to solve the multi-agent RL problem is to treat each agent independently. This idea is formalized in an independent Q-learning (IQL) algorithm [28], [29], which decomposes a multi-agent problem into a collection of simultaneous single-agent problems that share the same environment. Each agent runs Q-learning [15] or DQN [16]. IQL is scalable from the viewpoint of implementation while increasing the number of agents, and each agent only needs its local history of observations during the training. In this paper, IQL is trained to

minimize the loss:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \sum_{k=1}^{K} \left[\left(y_i^k - Q_k\left(h^k, a^k; \theta^k\right) \right)^2 \right], \tag{17}$$

where b is the batch size of transitions sampled from the replay memory, Q_k is the k^{th} agent's Q-value function, and y^k is the k^{th} agent's y^{DQN} as in Eq. (15).

2) Value Decomposition Networks: Value decomposition networks (VDNs) [30] aim to learn a joint action-value function $Q_{tot}(\boldsymbol{h}, \boldsymbol{a})$. It is assumed that the joint action-value function Q_{tot} can be additively decomposed into K Q-value functions for K agents, in which each Q-value function Q_i only relies on the local state-action history:

$$Q_{tot}(\boldsymbol{h}, \boldsymbol{a}) = \sum_{i=1}^{K} Q_i\left(h^i, a^i; \theta^i\right). \tag{18}$$

Therefore, each agent observes its local state, obtains the Q-values for its action, and selects an action, and then the sum of Q-values for the selected action of all agents provides the total Q-value of the problem. By using the shared reward and the total Q-value, the loss is calculated and then the gradients are back-propagated into the networks of all agents. Because each agent's DNN is updated on the basis of the total Q-value, each agent learns the best behavior for all agents, i.e., they learn cooperative behavior. The loss function for VDN is as follows:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(y_i^{tot} - Q_{tot}(\boldsymbol{h}, \boldsymbol{a}; \theta) \right)^2 \right], \tag{19}$$

where b is the batch size, $y^{tot} = r + \gamma \max_{\boldsymbol{a}'} Q_{tot}(\boldsymbol{h}', \boldsymbol{a}'; \theta^-)$ and θ^- are the parameters of a target network as in DQN.

3) QMIX: QMIX [31] extends VDN to address a broader class of environments. To represent a more complex factorization, a mixing network with trainable parameters is introduced to compute the total Q-value on the basis of each agent's state-action value function. As mentioned above, VDN adds restrictions to have the additivity of the Q-value and further shares the action-value function during the training. QMIX also shares the action-value function during the training. Besides, QMIX adds the below constraint to the problem:

$$\frac{\partial Q_{tot}}{\partial Q_i} \geq 0, \ \forall i, \tag{20}$$

which enforces positive weights on the mixer network, and as a result, QMIX can guarantee monotonic improvement. QMIX is trained to minimize the DQN loss, and the gradient is back-propagated to the individual Q-values, similarly to VDN.

V. Proposed Method

A. Concept of Coop-MADRL-Based VN Allocation

In the RL-based dynamic VN allocation method, an agent observes the current VN demands and physical network states and learns how to change the VN allocation to more efficiently use network resources. By applying DRL to agent learning, the agent can handle continuous traffic demand and high-dimensional network states.

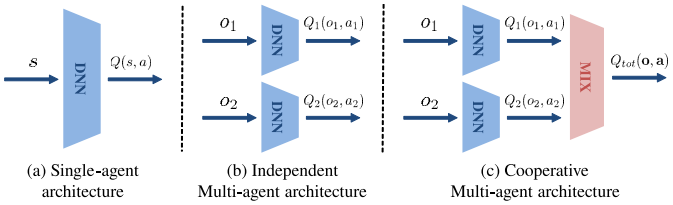


Fig. 2. Three types of DRL architectures: (a) single-agent, (b) independent multi-agent, and (c) cooperative multi-agent.

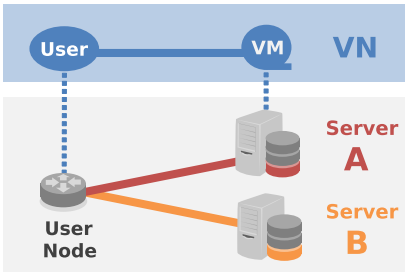


Fig. 3. Overview of VN allocation in a simple network topology.

Furthermore, to reduce the action space, we developed cooperative multi-agent techniques to be used with the proposed method.

Figure 2 shows three types of DRL architectures: single-agent, independent multi-agent, and cooperative multi-agent. To simplify the discussion, we assume the number of agents is set to 2 in this figure. In the single-agent case, an agent outputs $Q(s, a)$ with the global state s as input, where state s includes all information of a physical network and VNs. The dimension of output is equal to the action space $|\mathcal{A}|$ and is determined by the combination of all VN allocations. When assuming there are $|\boldsymbol{S}|$ servers, K VN demands, and the shortest path from the origin to destination is chosen, the number of candidate actions $|\mathcal{A}| = |\boldsymbol{S}|^K$. Moreover, as $|\boldsymbol{S}|$ and K increase, $|\mathcal{A}|$ increases exponentially. In the multi-agent case, each VN allocation control is assigned to each agent. The k^{th} agent outputs $Q_k(o^k, a^k)$ with its own local observation o^k as input, where observation o^k is part of the global state s and is related to a physical network and k^{th} VN information. The dimension of output is equal to the action space $|\mathcal{A}^k|$, which is determined only by k^{th} VN allocation, i.e., $|\mathcal{A}^k|$ drastically decreases to $|S|$.

We describe the multi-agent behavior in a simple example. Figure 3 shows the VN allocation for a simple network topology consisting of only two servers. Since the path is uniquely determined, each VN allocation is determined from two choices; the VM is assigned to server A or server B. In the single-agent case, if $K = 4$, the agent should find the optimal allocation from $|\mathcal{A}| = 2^4$ actions. However, if $K = 10$, the agent should find the optimal allocation from 2^{10} actions. The multi-agent technique reduces the number of candidate actions per agent. We prepared K agents for K VN demands, and each agent decides each VN allocation. In the multi-agent case, whether $K = 4$ or $K = 10$, k^{th} agent only finds the optimal allocation of k^{th} VN from the $|\mathcal{A}^k| = 2$ actions. In other words, each agent only decides whether it is better to assign each VM to server A or server B.

TABLE V
SYMBOL DESCRIPTIONS FOR COOP-MADRL

Symbols	Definitions
$e \in E$	Episode (E: Total episodes)
$\mathcal{G} := \{g_k\}$	Agent set ($1 \le k \le K$)
$s_t \in \mathcal{S}$	State at step t ($\mathcal{S}$: State space)
$\mathcal{O} := \{O^k\}$	Observation sets for all agents
$o_t^k \in O^k$	Observation for agent g_k at step t
$\boldsymbol{o}_t := \{o_t^k\}$	All observation at step t
$\mathcal{A} := \{\mathcal{A}^k\}$	Action sets for all agents ($\mathcal{A}^k$: Action space)
$a_t^k \in \mathcal{A}^k$	Action for agent g_k at step t
$\boldsymbol{a}_t := \{a_t^k\}$	All action at step t
r_t	Reward for agent g_k at step t
$Q_k\left(o_t^k, a_t^k\right)$	Action-value function for agent k
$Q_{tot}(\boldsymbol{o}_t, \boldsymbol{a}_t)$	Joint action-value function for all agent
$\mathcal{M}$	Replay memory
$h^i \in \boldsymbol{h}$	observation-action history
$\boldsymbol{h}$	observation-action history of all agents

When each agent is independent, the non-stationary learning problem described in the Introduction arises. In the cooperative multi-agent case, a mixed layer that calculates a joint action-value function Q_{tot} from each action-value function Q_k is added to cooperate with each agent. In the training phase, the loss of DNN is calculated by the shared reward and the joint action value Q_{tot}, thereby solving the non-stationary learning problem between agents and improving performance. It corresponds to *centralized training*. In the actual control phase, each agent can determine the best action on the basis of each observation o^k and each action value Q_k because the forward network of each agent DNN learns to output Q_k to minimize Q_{tot} during the training phase. In other words, agents can output globally optimal action by only calculating the forward network of their DNNs without a mixed network calculation. It corresponds to *decentralized execution* without other agent information.

B. Modeling

Table V summarizes the definitions of the variables of Coop-MADRL. We introduce K agents equal to the number of VNs. Each VN control is assigned to each agent, and the k^{th} agent learns how to optimize VN allocation for the k^{th} VN.

A state is defined as $s_t = [\boldsymbol{B}_t, \boldsymbol{D}_t, \boldsymbol{R}_t^L, \boldsymbol{R}_t^S]$, where $\boldsymbol{B}_t$ and $\boldsymbol{D}_t$ are the traffic and VM demands at t, and $\boldsymbol{R}_t^L$ and $\boldsymbol{R}_t^S$ are the residual resources of each link and server at t. Each $r_{ij,t}^L$ is calculated by $r_{ij,t}^L = 1 - c_{ij}^L u_{ij,t}^L$, and each $r_{i,t}^S$ is similar. An observation for agent g_k is defined as $o_t^k = [b_t^k, d_t^k, \boldsymbol{R}_t^L, \boldsymbol{R}_t^S]$. The state represents fully observed global information, and the observation represents agent-dependent information. Here, each VN demand such as b_t^k and d_t^k shows local information, and a physical resource such as $\boldsymbol{R}_t^L$ and $\boldsymbol{R}_t^S$ shows global information. By including the residual resources in agents' observation, each agent can take into account not only its demand information but also network-wide information. The action set $\mathcal{A}^k$ is defined as a set of allocation combinations included in agent g_k ($|\mathcal{A}^k| = |\boldsymbol{S}|$). To design rewards, we give a large negative value if the constraints are not satisfied; otherwise, a positive value depends on the objective function value. A certain negative value is also given when VN allocation is changed to avoid unnecessary VN reallocation.

Algorithm 1 Centralized Training of Coop-MADRL

1: **initialize:** agent parameters
2: **while** $t < T$ **do**
3: generate training-traffic sequences for all VNs
4: **initialize:** environment parameters
5: **for** $e = 0, E$ **do**
6: **for each** $g_k \in \mathcal{G}$ **do**
7: $o_t^k \leftarrow$ observation
8: $a_t^k \leftarrow$ select epsilon greedy action$\left(o_t^k\right)$
9: $\boldsymbol{o}_{t+1} \leftarrow$ update environment$(\boldsymbol{o}_t, \boldsymbol{a}_t)$ by Alg. 3
10: $r_t \leftarrow$ calculate reward$\left(\boldsymbol{o}_t, \boldsymbol{a}_t, \boldsymbol{o}_{t+1}\right)$ by Alg. 4
11: **if** $r_t \le -1$ **then**
12: terminate episode: $e \leftarrow E$
13: $t \leftarrow t + 1$
14: store episodic transition $\left(\boldsymbol{o}_j, \boldsymbol{a}_j, r_j\right), \forall j \in$ episode steps
15: train all agents $\mathcal{G}$ by random episodic transition

C. Formulation

The Coop-MADRL-based dynamic VN allocation method consists of two phases: centralized training and decentralized execution. The decentralized agents continually execute the dynamic VN allocation control after centralized training.

Algorithm 1 shows the centralized training of Coop-MADRL. Line 1 shows the initialization of agent parameters. A series of procedures (lines 2–15) is repeatedly executed until learning is complete. Lines 3–4 show the generation of the training-traffic sequences and the initialization of environment parameters and observation. A series of actions is called an episode, and each episode (lines 5–13) is repeatedly executed. In each episode, agents collect learning samples that are combinations of $\langle \boldsymbol{o}_t, \boldsymbol{a}_t, r_t, \boldsymbol{o}_{t+1} \rangle$. Each agent executes lines 6–8 in parallel. Line 7 shows the observation of o_t^k from the VN environment. Line 8 means the action selected on the basis of the strategy that a random action is selected with probability ε; otherwise, an action a_t^k that maximizes $Q_k(o_t^k, a')$ is selected (i.e., $\arg\max_{a' \in \mathcal{A}^k} Q_k(o_t^k, a')$) with probability $1 - \varepsilon$. This is to avoid convergence to a local optimum solution. In line 9, VN allocation is updated in accordance with $\boldsymbol{a}_t$ by Alg. 3 and returns the $\boldsymbol{o}_{t+1}$. Line 10 shows the reward calculation. Lines 11–12 mean the termination condition of agent learning. In this algorithm, $r_t \le -1$ is the termination condition, i.e., the state that does not satisfy at least one constraint. Line 14 shows stores in replay memory $\mathcal{M}$. The reason for storing the samples once in replay memory is to eliminate the time dependence of collecting training samples [16]. In line 15, all agents $\mathcal{G}$ are trained by the history of episodic transition, which is randomly taken from $\mathcal{M}$.

We describe the DNN architecture of Coop-MADRL. As shown in Fig. 2, DNN architecture consists of each agent's DNN layer and a mixed layer. For each agent's DNN layer, we introduced deep recurrent Q-networks (DRQN) [32] to handle time-series data as input, which incorporates recurrent neural networks (RNN) into DQN. We used a three-layer NN consisting of two fully connected layers and the gated recurrent

Algorithm 2 Dynamic VN Allocation Using Coop-MADRL

1: $Q_k\left(o^k, a^k\right) \leftarrow$ train all agents $\mathcal{G}$ by Alg. 1
2: **while** True **do**
3: **for each** $g_k \in \mathcal{G}$ **do**
4: $o_t^k \leftarrow$ observation
5: $a_t^k \leftarrow \arg\max_{a' \in \mathcal{A}^k} Q_k\left(o_t^k, a'\right)$
6: $\boldsymbol{o}_{t+1} \leftarrow$ update environment$(\boldsymbol{o}_t, \boldsymbol{a}_t)$ by Alg. 3
7: $t \leftarrow t+1$

Algorithm 3 Update Environment

1: $\boldsymbol{D}_{t+1}, \boldsymbol{B}_{t+1} \leftarrow$ update VN demand(t)
2: $\boldsymbol{Y}_{t+1} \leftarrow$ calculate next VM allocation$(\boldsymbol{a}_t, \boldsymbol{Y}_t)$
3: $\boldsymbol{\tau}_t \leftarrow$ convert traffic demands$\left(\boldsymbol{D}_{t+1}, \boldsymbol{Y}_{t+1}, \boldsymbol{Z}\right)$
4: $\boldsymbol{R}_{t+1}^L, U_{t+1}^L \leftarrow$ calculate next link utilization$(\boldsymbol{\tau}_t)$
5: $\boldsymbol{R}_{t+1}^S, U_{t+1}^S \leftarrow$ calculate next server utilization$(\boldsymbol{D}_t, \boldsymbol{Y}_t)$
6: $o_{t+1}^k = [d_{t+1}^k, b_{t+1}^k, \boldsymbol{R}_{t+1}^L, \boldsymbol{R}_{t+1}^S], \forall k$
7: **return** $\boldsymbol{o}_{t+1} = [o_{t+1}^1, \ldots, o_{t+1}^K]$

unit (GRU) layer [33]. We adopted Double-DQN [18] as the DRL algorithm. For the mixed layer, we use a mixed layer of VDN and QMIX as it is.

Algorithm 2 shows dynamic VN allocation using the Coop-MADRL algorithm. Lines 1 show the pre-training of $\mathcal{G}$ by using Alg. 1 and calculation of $Q_k(o^k, a^k)$. Next, this algorithm continually repeats lines 2–7 at every fixed t. In line 5, each agent selects an a_t^k that maximizes $Q_k(o^k, a^k)$.

D. Update Environment

In this formulation, the action sets that are changed by candidate VN allocation are determined only from the VM allocation, and the routes between users and VMs are uniquely determined in the environment update. We update the environment on the basis of the extendable NFV-integrated control architecture [22] that coordinates multiple control algorithms specified for individual metrics. We use the route-optimization algorithm as the specified optimization algorithm. By using this architecture, the routes between users and VMs are uniquely determined when the destination server is determined. The route-optimization calculates a routing variable $x_{ij,t}^{pq}$ to minimize the link utilization U_t^L while satisfying the constraints in Eqs. (7)–(11). Since the routing variable $x_{ij,t}^{pq}$ is a continuous value within 0–1 as shown in Eq. (11), this problem class is classified as a linear programming (LP) problem. If the routing variable $x_{ij,t}^{pq}$ is changed to a binary variable, it is classified as a mixed-integer programming (MIP) problem.

Algorithm 3 shows the procedure of the update environment. Line 1 shows the observation of the next demands. In line 2, the next VM allocation is calculated from $\boldsymbol{a}_t$ and current VM allocation. Line 3 means the calculation of the traffic matrix $\boldsymbol{\tau}_t$ from the origin node, destination server, and VN traffic demands $\boldsymbol{B}_t$. The origin node is determined by the user placement $\boldsymbol{Z}$, and the destination server is determined by the VM allocation $\boldsymbol{Y}_t$. Lines 4–5 show the calculation of next link and server utilization. Finally, Algorithm 3 returns $\boldsymbol{o}_{t+1}$.

Algorithm 4 Reward Calculation

1: $r_t \leftarrow \mathrm{Eff}\left(U_{t+1}^L\right) + \mathrm{Eff}\left(U_{t+1}^S\right)$
2: **if** $\boldsymbol{Y}_t \neq \boldsymbol{Y}_{t+1}$ **then**
3: $r_t \leftarrow r_t - \alpha P_t$
4: **return** $\max(-5, \min(1, r_t))$

E. Reward Calculation

We design the reward function on the basis of the objective function Eq. (1). Algorithm 4 shows the procedure of the reward calculation for $\mathcal{G}$. The $\mathcal{G}$ learns how to maximize the reward. The term Eff(x) in Alg. 4 shows the efficiency function and is defined as follows:

$$\mathrm{Eff}(x) = \begin{cases} 0.5 & (x \leq 0.4) \\ -x + 0.9 & (0.4 < x \leq 0.9) \\ -2x + 1.8 & (0.9 < x \leq 1) \\ -x - 0.5 & (1 < x). \end{cases} \tag{21}$$

This function returns a positive value depending on x if $x < 0.9$; otherwise it returns a negative value. We designed a handcrafted efficiency function so that the efficiency decreases as x increases. The decrease of efficiency doubles when x is more than 0.9, and the values of 0.5 indicate the upper limit of this function. We also designed the function to drastically decrease efficiency when $x > 1$, i.e., when the constraints are not satisfied. Note that this design is just one example that our method performed suitably, and there is still room for improvement in the design of the efficient function. Elucidating the relationship between reward function and VN allocation performance remains one of our future challenges.

Line 1 show the Eff (x) calculation. In lines 2–3, it adds $-\alpha P_t$ as a penalty shown in Eq. (2) if the $\boldsymbol{Y}_t$ is changed. Finally, it returns a clipped reward within $-5 \leq r_t \leq 1$. The reason for clipping the reward is to increase the stability of agent learning [16].

VI. Evaluation

We evaluated the effectiveness of the proposed method through simulations in terms of performance, computation time, and scalability for the number of VNs and network topology size. We also evaluated the generalization performance for unknown traffic patterns. We prepared two environments and five comparison methods. We implemented the DRL algorithm based PyTorch [34] and PyMARL [35] and route-optimization algorithm using the GNU Linear Programming Kit (GLPK) [36].

A. Evaluation Conditions

We set the number of VNs to $K = 20$ as the default settings. In evaluating scalability, we increased the K from 20 to 60 and increased link capacity and server capacity in proportion to K. We also set total time steps to $T = 3.0 \times 10^5$ and total episodes to $E = 200$, and the weight of the penalty function of VM migration to $\alpha = 0.01$ for all evaluations.

For the VN-demand conditions, the $\boldsymbol{Z}$ is randomly generated and fixed for all evaluations. The $\boldsymbol{D}_t$ is randomly generated integer values within 1–5 and is reset at the beginning of each

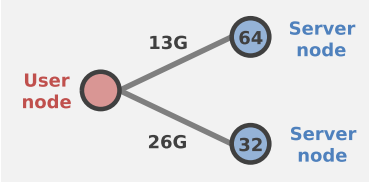


Fig. 4. Simple Network Topology.

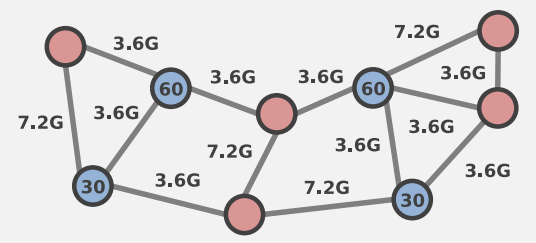


Fig. 5. Practical Network Topology.

episode. The $\boldsymbol{B}_t$ is 200-step time-series sequences randomly generated by various traffic models at the beginning of each episode. After generating traffic sequences, all the traffic data were normalized so that the average traffic volume was 1 Gbps and the minimum traffic volume was 0. We generated the $\boldsymbol{B}_t$ for training and evaluation. The details of each model are described in Section VI-B.

For the physical-network conditions, we prepared simple and practical networks. Figures 4 and 5 show the network topology for the simple and practical networks, respectively. We assumed that all users and VMs are placed in the user node and server node, respectively. The values in these figures show the server capacity and link capacity when $K = 20$. When K increases, server capacity and link capacity increase in proportion to K. All evaluations except for the scalability evaluation for the network topology size are on the basis of these two topologies. We prepared other topologies to evaluate scalability for the topology size (See Section VI-H in detail).

1) Simple-Network Conditions: We used the 3-node topology, which consists of 1 user node and 2 server nodes. When $K = 20$, the number of candidate VM allocations was $2^{20} \simeq 1.0 \times 10^6$, and that of the candidate actions of g_k was 2.

2) Practical-Network Conditions: We used the 9-node topology based on Internet2 [37], which consists of 5 user nodes and 4 server nodes as shown in Fig. 5, and all users and all VMs are allocated in user nodes and server nodes, respectively. When $K = 20$, the number of candidate VM allocations was $4^{20} \simeq 1.1 \times 10^{12}$, and that of the candidate actions of g_k was 4.

B. Traffic Models

Figure 6 shows the six types of generated traffic sequences used in this evaluation. It shows the generated samples of traffic sequences normalized within 0–2 Gbps after generating them. To evaluate the performance of the proposed method under various traffic patterns, we prepared five different models to generate time-series traffic sequences and one option to modify the generated traffic sequences. This option can apply to all traffic models, which assumes the case when unexpected heavy traffic demand suddenly emerges. We describe the models and option as follows.

1) ARMA Model: Figure 6(a) shows the traffic sequences generated by the autoregressive moving average (ARMA) model. This model assumes general traffic patterns, such as aggregated broadband traffic. The (p, q)–order ARMA model is formulated as follows.

$$f(t) = \varepsilon_t + \sum_{i=1}^{p} \phi_i f(t-i) + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}, \tag{22}$$

where $f(t)$ is the time-series data at time-step t, p is the order of AR(p), q is the order of MA(q), ϕ_i are AR(p) model parameters, θ_i are MA(q) model parameters, and ε_t is the distributed error at time-step t. We set $p = 2$, $q = 50$, and $\varepsilon_t \sim \mathcal{N}(0, 1)$. Here, $\mathcal{N}(0, 1)$ is the normal distribution with a mean of 0 and variance of 1. We also set hyper-parameters $\phi_1 = 0.9$, $\phi_2 = -0.1$, and $\theta_i = 0.95$ ($\forall i, 1 \leq i \leq 50$). These hyper-parameters were set to represent long-term fluctuations.

2) ARMA Model With Other Parameters: Figure 6(b) shows the traffic sequences generated by the ARMA model when changing the hyper-parameters from Fig. 6(a). We set $p = 2$, $q = 5$ for Eq. (22) in this model. We also set hyper-parameters $\phi_1 = 0.5$, $\phi_2 = -0.1$, and $\theta_i = 0.15$ ($\forall i, 1 \leq i \leq 5$). These hyper-parameters were set to represent short-term fluctuations. We changed q and θ_i from (1) ARMA model, which is equivalent to weakening the effect of MA factors.

3) SARIMA Model: Figure 6(c) shows the traffic sequences generated by the seasonal autoregressive integrated moving average (Seasonal ARIMA; SARIMA) model. This model assumes periodical traffic patterns, such as daily traffic trends. The $(p, d, q) \times (P, D, Q)_m$–order SARIMA model is formulated as follows.

$$\phi_p(B)\Phi_P(B)(1-B)^d(1-B^m)^D f(t) = \theta_q(B)\Theta_Q(B)\varepsilon_t, \tag{23}$$

where $f(t)$ is the time-series data at time-step t, B is the backward shift operator and is defined as $B^k f(t) := f(t-k)$. The $\phi_p(B)$ and $\Phi_P(B)$ are called AR operators, and the $\theta_q(B)$ and $\Theta_Q(B)$ are called MA operators. Each operator is defined as follows.

$$\phi_p(B) := 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p \tag{24}$$

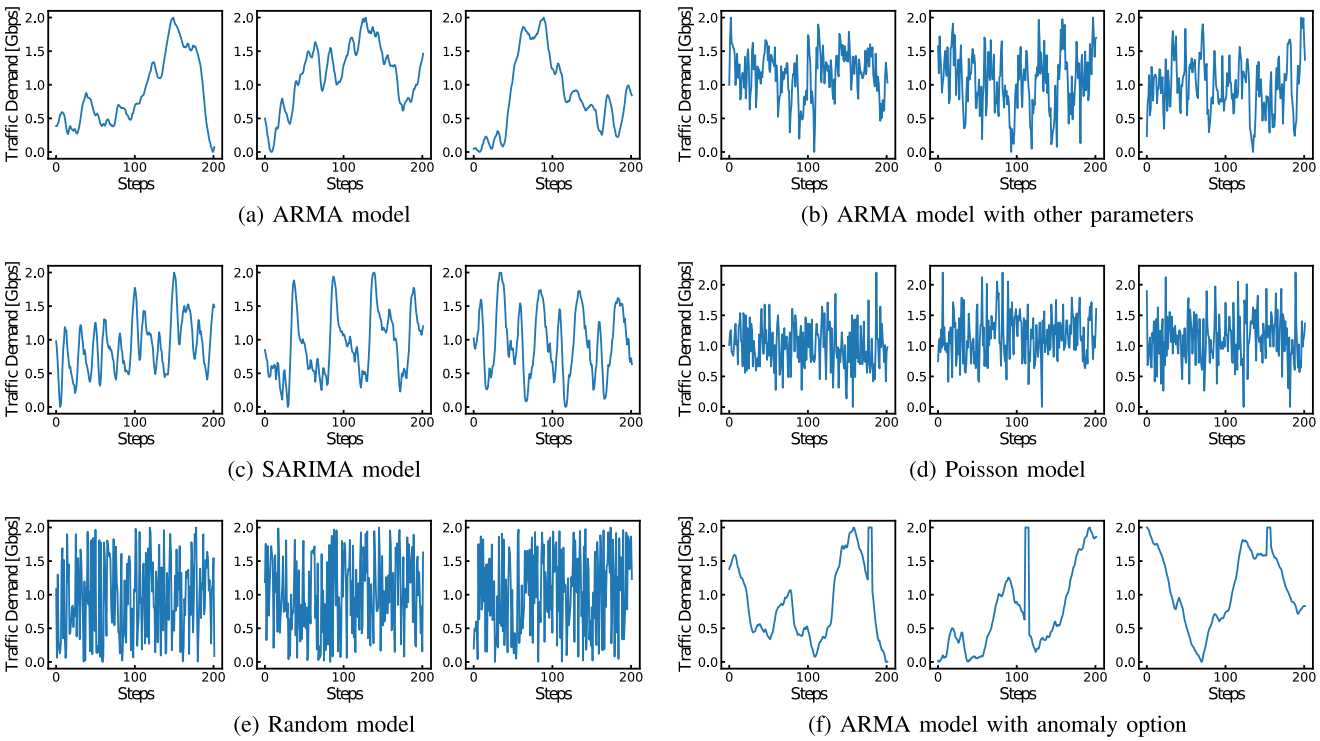
$$\theta_q(B) := 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q \tag{25}$$

$$\Phi_P(B) := 1 - \Phi_1 B^m - \Phi_2 B^{2m} - \cdots - \Phi_P B^{Pm} \tag{26}$$

$$\Theta_Q(B) := 1 - \Theta_1 B^m - \Theta_2 B^{2m} - \cdots - \Theta_Q B^{Qm} \tag{27}$$

Here, (p, d, q) and (P, D, Q) show the order of the ARIMA model, and m shows the period of time-serial sequences. We set (p, d, q) = (2, 0, 5), (P, D, Q) = (1, 0, 1), and $m = 50$. We also set hyper-parameters $\phi_1 = 0.9$, $\phi_2 = -0.1$, $\theta_i = 1 - 0.1 \times i$ ($\forall i, 1 \leq i \leq 5$) and $\Phi_1 = \Theta_1 = 0.9$. These hyper-parameters were set to represent periodical fluctuations based on (2) ARMA model with other parameters.

4) Poisson Model: Figure 6(d) shows the traffic sequences generated by the Poisson process. This model assumes traffic patterns for Internet of Things (IoT) applications. The traffic model for IoT applications is often described by periodic patterns from asynchronous sources. This superposition



(a) ARMA model (b) ARMA model with other parameters

(c) SARIMA model (d) Poisson model

(e) Random model (f) ARMA model with anomaly option

Fig. 6. Various traffic models.

of IoT traffic streams can be approximated by the Poisson process [38]. The Poisson model is formulated as follows.

$$f(t) = v\mathcal{P}_t,\ \mathcal{P}_t \sim \Pr(X{=}k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!},$$

where $\mathcal{P}_t$ is the Poisson distribution, the parameter λ is the number of IoT devices per unit time, and the v is the average traffic volume per IoT device.

5) Random Model: Figure 6(e) shows the traffic sequences generated by white noise. We set the traffic volume at each time-step to a random value in the range of 0–1.

6) With Anomaly Option: Figure 6(f) shows the traffic sequences with the anomaly option applied to the ARMA model. We randomly injected 5 steps between 0–200 steps as anomaly steps. We set the traffic volume of an anomaly step to 2 Gbps on the assumption that unexpected heavy traffic demand would emerge. After the anomaly traffic was injected, we normalized the average traffic to 1 Gbps. This normalization reduces the average traffic volume for the step that does not inject an anomaly since we set the average traffic volume that includes anomaly traffic to 1 Gbps. We apply this option to ARMA, SARIMA, Poisson, and Random models in this evaluation.

7) Mixed Model: We define a mixed model consisting of five traffic models and four traffic models with the anomaly option mentioned above. At the beginning of each episode, each user randomly selects one of the nine traffic patterns.

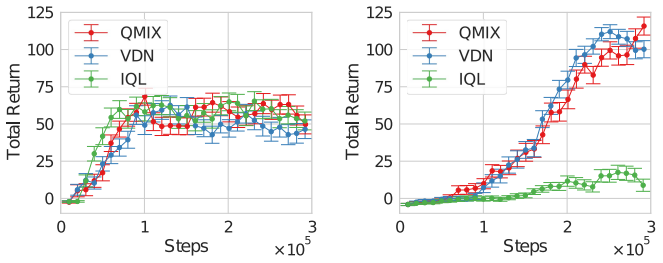
C. Comparative Methods

Table VI summarizes our methods and the comparison methods. QMIX and VDN indicate the Coop-MADRL-based dynamic VN allocation method shown in Alg. 2, which each use QMIX and VDN in a mixed network and use Eq. (19) for end-to-end training. IQL indicates the MADRL-based method without cooperation, i.e., each agent learns on the basis of their reward and Eq. (17) is used for end-to-end training. IQL does not impose any restrictions on an agent's action at each time. Note that our previous method [12] restricted one agent that could act at each time, and it is different from IQL. Also note that single-agent DRL was not evaluated because learning is clearly unsuccessful due to requirements for huge training iterations until the Q-values for all actions are sufficiently close to the optimal. When a single agent trains in a practical network, it is estimated that at least 10^{12} training steps will be required even if the agent learned each action once.

TABLE VI
SUMMARY OF OUR METHODS AND COMPARISON METHODS

Methods	Methodology	Dynamic	Cooperation
QMIX (Ours)	Coop-MADRL	✓	✓
VDN (Ours)	Coop-MADRL	✓	✓
IQL	MADRL	✓	-
Static Allocation (SA)	Heuristic	-	-
Exhaustive Search (ES)	Meta-Heuristic	✓	-

Static Allocation (SA) is the heuristic method where VM allocation is fixed to minimize the sum of U_0^L and U_0^S for an average amount of past demands, and routes were dynamically changed as with other methods. Since finding the initial VM allocation for SA is also NP-hard, in this evaluation, we sequentially decided on the best VM allocation that minimizes the sum of U_0^L and U_0^S for average VN demands consisting of 1 Gbps traffic demand and VM size of 3. By comparing SA and other methods, we expect to determine the effectiveness of dynamic allocation.



(a) Simple Network (b) Practical Network

Fig. 7. Training curves tracking the agent's total return (training by Mixed model).

Exhaustive Search (ES) is the meta-heuristic method where the action that maximizes the reward r_t is selected from all candidate actions $\mathcal{A}$ at t. ES finds the best action by exhaustively calculating each reward one by one for all candidate actions at t, which is equivalent to solving an optimization problem every t. The objective of ES is to maximize the immediate reward, which is the reward obtained in the present. Alternatively, the objective of RL is to maximize the delayed reward, which is the expectation of the total rewards to be obtained in the future. By comparing ES and other MADRL-based methods, we expect to evaluate the difference in performance between the two types of rewards. Note that the solution of ES is different from the global optimal solution because the global optimal allocation is defined as the solution that maximizes the sum of the rewards at each time, as shown in Eq. (1). To find the global optimal solution, it is necessary to perfectly predict all future demands and calculate the optimal allocation that maximizes the sum of the rewards at each time. Therefore, the global optimal solution is challenging to calculate within a realistic computation time, even for the simple-network condition.

D. Evaluation: Training Curve

Figure 7 shows the training curves tracking the agent's total return under simple- and practical-network conditions. The total return is defined as the sum of rewards at each time until the end of the episode. We carried out 5 evaluations for every 1×10^4 steps with random initial conditions. The width of each bar indicates the standard deviation ($\pm\sigma$). We adopted a mixed model for traffic patterns in training and evaluation.

Figure 7(a) shows that the average total return of the three MADRL-based methods increased as the training progressed. It also shows that the curves of the total return of the three methods are almost the same. Figure 7(b) shows that the average total return of the Coop-MADRL (QMIX and VDN) based methods increased as the training progresses, while that of the non-Coop-MADRL (IQL) does not increase. This means that IQL cannot learn the suitable allocation that maximizes the objective function while satisfying constraints. We discuss the performance details in Section VI-E.

E. Evaluation: Performance

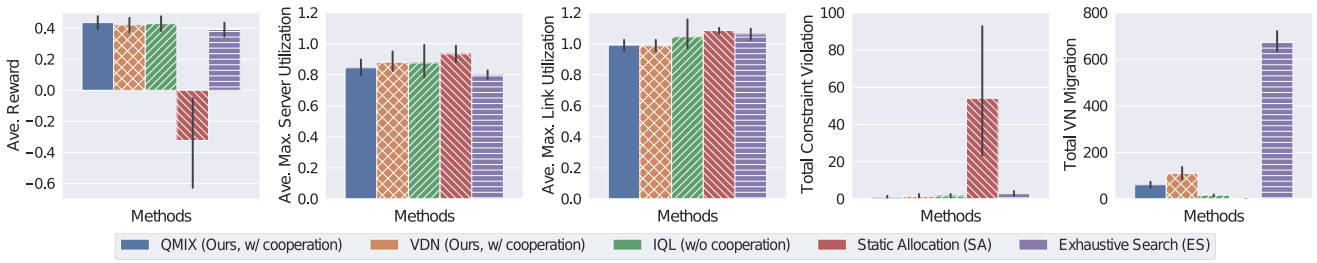
Figure 8 shows the average performance of each method under simple- and practical-network conditions. We carried out 20 calculations with random initial conditions and set the same random seeds for all evaluations. The width of each bar indicates the standard deviation ($\pm\sigma$). We used agents trained with the mixed model as described in Fig. 7. We also used the mixed model for the traffic model in this evaluation. The mixed model includes large demand fluctuations by the Random and Poisson models. The performance of each method can roughly be compared with the average reward. We also investigate the details of performance as shown in the 2–5$^{\text{th}}$ metrics in Figs. 8(a) and 8(b). That is, the performance metrics are the average maximum server utilization U_t^S, average maximum link utilization U_t^L, total constraint violation, and total VM migration. The server and link utilization usually take a value in the range of 0–1 if all constraints are satisfied. On the other hand, if constraint violations occur at t, the server and link utilization at t may take a value greater than 1 depending on the amount of exceeded resources. When k VMs migrate at a certain time, the total VM migration adds k.

In summary, Coop-MADRL reduced the maximum server and link utilization and drastically reduced the total constraint violations compared with SA. Therefore, our method is effective for simple- and practical-network conditions.

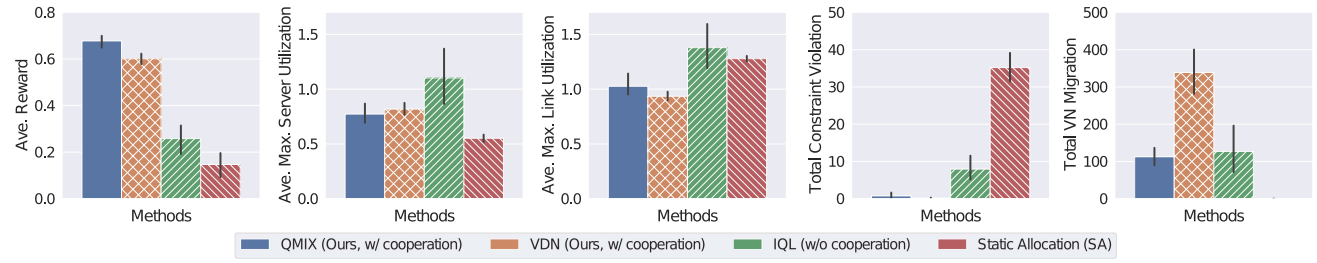
1) Performance on Simple Network: We first compare SA and the other four dynamic allocation methods. Figure 8(a) shows that the four dynamic methods performed better than SA. The reason is that the dynamic allocation methods can prevent constraint violations by changing the VN allocation in accordance with the demand change. This also indicates that three MADRL methods performed better than SA and that reducing the action space by using the multi-agent technique is effective. In particular, even IQL performed better than SA. MADRL can reduce the constraint violations without information of the next demand because the RL agent indirectly predicts the next demand and learns the effective VN allocation change through the relationship between the network state and network efficiency. However, since MADRL calculates rewards on the basis of the current demand and not the next-step demand, constraint violations cannot be absolutely eliminated.

We next compare ES and the other three MADRL-based methods. ES performed slightly lower than three MADRL-based methods. In addition, the number of total VM migrations is notable in ES, even though ES also considers a VM migration penalty described in Eq. (2) as well as other MADRL-based methods. The reason is that, since ES calculates the solution that maximizes the reward for the current demand, not for the next demand as described in Section VI-C, the optimal action for ES is frequently changed when the VN demand fluctuates. As a result, VM migration is increased, and its performance worsens. On the other hand, the MADRL-based methods learn a policy that maximizes the expected value of the cumulative reward obtained in the future. Thus, the MADRL-based methods can reduce VM migrations and improve the average reward. The result shows that the optimization for immediate rewards does not necessarily maximize the average reward, and the optimization of delayed reward by RL is effective when demand is fluctuating.

Finally, we compare three MADRL-based methods. Figure 8(a) shows that the performance of the three MADRL-based methods was almost the same. It assumes that the performance of the three methods is saturated because this



(a) Training and evaluation by Mixed model, for simple network



(b) Training and evaluation by Mixed model, for practical network

Fig. 8. Performance evaluation for each method.

condition is too simple. This can be seen from the training curves in Fig. 7. Although QMIX and VDN perform roughly the same, they differ in terms of total VM migration. We consider that QMIX can simultaneously reduce the constraint violations and unnecessary VM migrations since agents of QMIX acquired a higher level of cooperation knowledge than those of VDN.

2) Performance on Practical Network: Figure 8(b) shows the average performance of four of the methods under practical-network conditions. Note that SA changed routes calculated with the route-optimization algorithm at each t, and ES could not evaluate within a practical time because its computation time dramatically increased. Also, note that the average reward was higher for the practical network than for the simple network, which is due to the difference in evaluation conditions, and both reward values are not directly comparable.

Similar to the simple-network condition, three dynamic methods performed better than SA. Comparing its performance under simple- and practical-network conditions, both Coop-MADRL-based methods (QMIX and VDN) performed better than non-Coop-MADRL (IQL) based methods because the degree of freedom of allocation and cooperation improved as the numbers of nodes and links increased. In particular, both cooperative methods were able to reduce the number of total constraint violations drastically. Since each agent in IQL only considers its action without other agents' actions, the agents concentrate on lightly loaded resources in some cases, and this causes violations. On the other hand, QMIX and VDN have higher maximum server utilization than SA. The reason is that agents chose the action that maximized the average reward and minimized the constraint violations even if the maximum server utilization was increased.

F. Evaluation: Computation Time

Table VII shows the average computation time per t of each method, which includes the time to decide the action, update the next state, and evaluate the performance. We used Intel core i9-9980HK for the evaluation. In the evaluation phase, the computational complexity is the same for the three MADRL methods because the three methods only calculate the forward network of their DNNs without a mixed network calculation in a decentralized manner as described in Section V-A. Under the simple-network conditions, the computation time per t of four methods except ES was less than a few milliseconds, and that of ES drastically increased. MADRL took less than 1 ms for an agent to decide the next action. Though ES was the best method from the viewpoint of performance, it was not effective due to the huge computation time. Under the practical-network conditions, the computation time per t of all methods other than ES was less than 1 s. The computation times of the four methods except ES were almost equal because the time of the route-optimization calculation became dominant. The computation time of ES is estimated from the computational quantities of iteration since it was difficult to find in conventional time. The optimal solution may be found faster by improving the search algorithm, but it would be difficult to achieve the same speed as MADRL. Finally, we revealed that our method is also useful as a dynamic VM allocation method in terms of computation time.

TABLE VII
AVERAGE COMPUTATION TIME PER STEP FOR $K = 20$

Methods	Simple Network	Practical Network
QMIX (Ours)	2.3 ms	75 ms
VDN (Ours)	2.3 ms	75 ms
IQL	2.3 ms	75 ms
Static Allocation (SA)	1.8 ms	74 ms
Exhaustive Search (ES)	69 s	(estimate) 7.2×10^7 s

We also mention the computation time of training agents. For the simple network, QMIX took about 100 minutes and VDN and IQL took about 70 minutes to finish the training of the total time steps $T = 3.0 \times 10^5$. For the practical

TABLE VIII
SCALABILITY EVALUATION RESULTS FOR THE NUMBER OF VNS IN PRACTICAL NETWORK (QMIX, TRAINING BY ARMA MODEL)

Number of VNs	Ave. Reward
20	0.57 ± 0.10
40	0.47 ± 0.07
50	0.34 ± 0.08
55	0.20 ± 0.24
60	-5.0 ± 0.02

network, QMIX and VDN took about 7.1 hours and IQL took about 6.5 hours to finish the training of the total time steps $T = 3.0 \times 10^5$. This indicates that our method can learn the optimal VN allocation in less than half a day for networks of the scale used in the evaluation.

G. Evaluation: Scalability for Number of VNs

Table VIII shows the scalability of QMIX for the number of VNs K under practical-network conditions. We carried out 20 calculations with random initial conditions and calculated the mean value and the standard deviation of rewards. We used the ARMA model for the traffic model in training and evaluation. We increased the number of VNs K until the performance decreased. We also increased link capacity and server capacity in proportion to K, keeping the average traffic demands and average VM size constant.

As a result, the average reward rapidly decreased when the number of VNs exceeded 60 VNs, and our method was effective up to about 50 VNs. RL should heuristically discover actions that improve the reward and satisfy all constraints in the early learning steps when the agent acts randomly. It seems that scalability could be improved by supporting the initial learning process by providing correct training data by a person. Our method is as scalable as the previous methods [10]–[12] because they simultaneously handled less than 50 VNs at each time step. In this paper, a VN is assumed to be a resource-isolated network slice provided by each service provider and shared by many users or IoT devices. We believe that our method is sufficiently scalable considering the number of services handled by today's networks.

H. Evaluation: Network Topology Dependency

To verify the effectiveness of the proposed method for larger physical network topologies, we evaluate the computation time and performance of the proposed method for various network topologies. The network topology used in the evaluation refers to SNDLib [39], which is a library of test instances of survivable fixed telecommunication network design. In all evaluations, we used QMIX as the learning algorithm and the ARMA model as the traffic model, and we set the number of VNs to $K = 20$. We also randomly set user nodes and server nodes for all topologies. Other conditions are the same as in the above evaluation.

Table IX shows the computation time for various physical network topologies. The execution time shows the computation time required to select the agent's action, update the VN allocation, and calculate the reward in each execution step, which corresponds to lines 3–7 in Alg. 2. The LP time shows the computation time required to calculate the route-optimization algorithm in each execution step, which is a part of execution time and corresponds to line 4 in Alg. 3. The training time shows the computation time required for the agents to complete the training of the total time steps $T = 3 \times 10^5$. The result shows that all computation times drastically increase as the number of topology nodes increases. The result also shows that most of the execution and training time is spent solving the route-optimization problem. Due to the drastic increase in computation time, we estimated the training time for large topologies based on the execution time and the total training steps. Here, we assume that the total training steps $T = 3 \times 10^5$ are sufficient for all topologies, but larger ones may require more training steps.

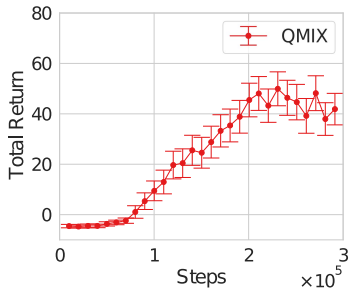


Fig. 9. Training curves tracking the agent's total return in *Atlanta* network.

We consider the practical limits of topology size in terms of computation time. The execution time determines the VN allocation control interval. For example, when the execution time takes 5 minutes, the control interval cannot be shorter than 5 minutes. If we aim to keep the control interval under 1 minute, the limit of the number of nodes can be estimated to be about 30 nodes. Similarly, if we aim for a control interval of 5 minutes or less, the limit can be estimated to be about 50 nodes. As for the training time, assuming that the reasonable training time is less than one week, the limit of the number of nodes can be estimated to be about 20 nodes. In conclusion, unless speeding up the computation time, we can estimate that the upper limit of the number of nodes to which the proposed method can be applied is about 20. Parallelization is one of the promising candidates for speedup the proposed method. Existing studies have reported that a distributed DRL can dramatically speed up the agent training process. For example, Espeholt *et al.* [40] developed a distributed DRL architecture that scales up to thousands of machines without sacrificing data efficiency or resource utilization. The proposed method may be applied to up to 30 nodes by parallelizing the agent training process.

To verify the effectiveness of the proposed method for network topologies within 20 nodes, we evaluated the performance of the proposed method on Atlanta network with 15 nodes and 22 links. Figure 9 shows the training curves tracking the agent's total return for Atlanta network. As in the practical-network condition, we set each server capacity to 30 and each link capacity to 3.6 Gbps. The result shows that the average total return of QMIX increased as the training progressed. Table X shows the average performance of

TABLE IX
COMPUTATION TIME FOR VARIOUS PHYSICAL NETWORK TOPOLOGIES

Topology	# of Nodes	# of Links	Execution Time	LP Time	Training time
Internet2	9	13	0.06 s	0.05 s	0.30 day
Abilene	12	15	0.17 s	0.15 s	0.83 day
Atlanta	15	22	0.43 s	0.39 s	1.6 day
Geant	22	36	3 s	2.7 s	10 day*
France	25	45	5.3 s	4.9 s	18 day*
India35	35	80	21 s	20 s	73 day*
Germany50	50	88	96 s	89 s	3.3×10^2 day*
Ta2	65	108	305 s	286 s	1.0×10^3 day*

*Estimated value

TABLE X
PERFORMANCE EVALUATION RESULTS IN **ATLANTA** NETWORK (TRAINING BY ARMA MODEL)

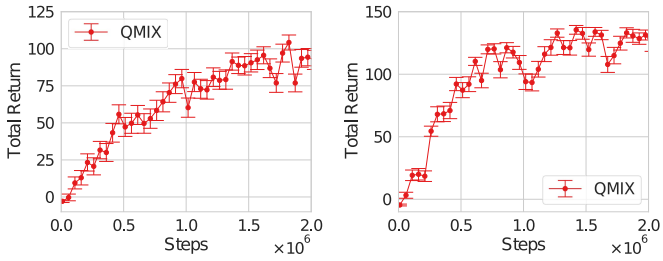
Methods	Ave. Reward	Ave. Max. Server Util.	Ave. Max. Link Util.	Constraint Violation	Total VM Migration
QMIX (Ours)	0.43 ± 0.14	0.97 ± 0.39	1.1 ± 0.21	9.5 ± 12	189 ± 62
Static Allocation (SA)	0.37 ± 0.11	0.50 ± 0.07	1.1 ± 0.07	14 ± 13	0.00 ± 0.0

TABLE XI
COMPUTATION TIME FOR VARIOUS PHYSICAL NETWORK TOPOLOGIES (WITH **SHORTEST PATH**)

Topology	# of Nodes	# of Links	Execution Time	Training time
Geant	22	36	1.6 ms	8.1 h
France	25	45	1.8 ms	16.9 h
India35	35	80	2.0 ms	11.9 h
Germany50	50	88	3.0 ms	10.3 h
Ta2	65	108	3.9 ms	19.8 h

QMIX and SA. Note that, as described in Section VI-C, SA also dynamically selects an optimal routes calculated with the route-optimization algorithm at each t. The result shows that QMIX performed better than SA because QMIX can prevent constraint violations by changing the VN allocation by the demand change. Although we can confirm the effectiveness of QMIX, we found that the relative performance improvement decreases compared to the result of Fig. 8(b). We consider two reasons for the decrease in the performance difference between the two methods. First, we suppose that the larger network topology makes dynamic routing more effective, and the dynamic routing improved the performance of VN allocation regardless of VM placement. In other words, the dynamic routing could compensate for the performance degradation of SA caused by the inefficient VM placement. Next, we also suppose that the larger network topology has increased the required training steps. If so, we can improve the performance of QMIX by increasing the training steps.

Next, based on the above results, we discuss applying the proposed method for larger network topologies by drastically reducing computation time. Since most of the computation time is spent solving the route-optimization problem as shown in Table IX, relaxing the route-optimization problem and replacing dynamic path allocation with static path allocation is an effective way of speeding up the method. Specifically, we replace the optimal path by LP with the shortest path by Dijkstra's algorithm, which corresponds to line 4 in Alg. 3. We then evaluate the computation time and performance for various network topologies. Finally, we show that the proposed method using the shortest path is adequate for practical topology size.



(a) **India35** (b) **Germany50**

Fig. 10. Training curves tracking the agent's total return with the proposed method using the shortest path.

Table XI shows the computation time in the proposed method using the shortest path for various physical network topologies. To ensure sufficient training, we increased the total time steps to $T = 2 \times 10^6$. The results show that all computation times are drastically faster than those estimated in Table IX. For all conditions, the execution time is less than one second, and the training time is less than one day. Moreover, since Ta2 network is the largest topology that mimics the core network in SNDLib, we conclude that the proposed method using the shortest path has no limitation on the number of nodes in the range of practical network topologies. The result also shows that the training time does not depend on the number of nodes. The reason is that other parameters, e.g., physical network and VN request parameters, are more dominant factors in determining the training time than the number of nodes.

Figure 10 shows the training curves tracking the agent's total return for India35 network and German50 network. We set

TABLE XII
PERFORMANCE EVALUATION RESULTS IN **INDIA35** NETWORK (TRAINING BY ARMA MODEL, WITH **SHORTEST PATH**)

Methods	**Ave. Reward**	**Ave. Max. Server Util.**	**Ave. Max. Link Util.**	**Constraint Violation**	**Total VM Migration**
QMIX (Ours)	0.56 ± 0.08	0.90 ± 0.16	0.94 ± 0.28	0.70 ± 1.9	263 ± 29
Static Allocation (SA)	−0.40 ± 0.33	0.45 ± 0.07	1.3 ± 0.08	102 ± 38	0.00 ± 0.0

TABLE XIII
PERFORMANCE EVALUATION RESULTS IN **GERMANY50** NETWORK (TRAINING BY ARMA MODEL, WITH **SHORTEST PATH**)

Methods	**Ave. Reward**	**Ave. Max. Server Util.**	**Ave. Max. Link Util.**	**Constraint Violation**	**Total VM Migration**
QMIX (Ours)	0.68 ± 0.06	0.84 ± 0.26	0.87 ± 0.27	0.10 ± 0.45	118 ± 32
Static Allocation (SA)	−0.78 ± 0.22	0.45 ± 0.07	1.5 ± 0.12	142 ± 27	0.00 ± 0.0

TABLE XIV
PERFORMANCE EVALUATION RESULTS IN **TA2** NETWORK (TRAINING BY ARMA MODEL, WITH **SHORTEST PATH**)

Methods	**Ave. Reward**	**Ave. Max. Server Util.**	**Ave. Max. Link Util.**	**Constraint Violation**	**Total VM Migration**
QMIX (Ours)	0.53 ± 0.26	0.87 ± 0.32	1.1 ± 0.67	2.8 ± 10	174 ± 44
Static Allocation (SA)	−1.3 ± 0.19	0.45 ± 0.07	1.9 ± 0.19	181 ± 16	0.00 ± 0.0

TABLE XV
EVALUATION OF AVERAGE REWARDS FOR VARIOUS TRAFFIC MODELS IN SIMPLE NETWORK (TRAINING BY **MIXED** MODEL)

Traffic Models	QMIX, Mix	VDN, Mix	IQL, Mix	Static Allocation (SA)
ARMA	0.46 ± 0.11	0.46 ± 0.12	0.15 ± 0.77	−0.22 ± 0.68
ARMA with other parameters	0.44 ± 0.12	0.43 ± 0.15	0.36 ± 0.41	−0.32 ± 0.71
SARIMA	0.43 ± 0.11	0.42 ± 0.11	0.35 ± 0.25	−0.30 ± 0.73
Poisson	0.36 ± 0.11	0.35 ± 0.12	0.35 ± 0.14	−0.41 ± 0.73
Random	0.23 ± 0.82	0.23 ± 0.74	0.31 ± 0.55	−0.26 ± 0.73
ARMA with anomaly	0.39 ± 0.13	0.38 ± 0.13	0.32 ± 0.43	−0.36 ± 0.72
SARIMA with anomaly	0.45 ± 0.09	0.44 ± 0.09	0.43 ± 0.11	−0.38 ± 0.75
Poisson with anomaly	0.46 ± 0.11	0.44 ± 0.15	0.34 ± 0.53	−0.10 ± 0.60
Random with anomaly	0.39 ± 0.13	0.38 ± 0.13	0.32 ± 0.43	−0.36 ± 0.72
Mix All Traffic	0.44 ± 0.10	0.42 ± 0.11	0.43 ± 0.11	−0.32 ± 0.70
Ave. All Traffic Models	0.41 ± 0.28	0.40 ± 0.26	0.34 ± 0.42	−0.30 ± 0.70

each server capacity to 30 and each link capacity to 5.0 Gbps. We increased link capacities because the shortest path concentrates the traffic load on some links. The result shows that the average total return of QMIX increased as the training progressed. Tables XII–XIV show the average performance of QMIX and SA. Both QMIX and SA used the shortest path for route calculation, since solving LP at each step is difficult for both methods due to the computation time. The results show that QMIX performed significantly better than SA. This performance difference is since QMIX dynamically changes the placement of VMs or not. In particular, the performance of SA dramatically decreased due to static routing. The reason is that, as mentioned before, the performance of SA was strongly dependent on dynamic routing. In conclusion, we revealed that the proposed method using the shortest path performs sufficiently in large network topologies.

In summary, we revealed that the proposed method could be applied to network topologies with less than 65 nodes. More precisely, the proposed method with route-optimization could be applied to network topologies with less than 20 nodes. The proposed method that uses the shortest path for route calculation could be applied regardless of the topology size. In addition, we revealed that the proposed method outperforms SA in terms of performance regardless of the topology size.

I. Evaluation: Generalization Performance

The generalization performance is a measure of how accurately an algorithm is able to perform outcomes for previously unseen data. To evaluate the generalization performance for traffic demands, we evaluated the average reward using various traffic models that were different from those during training. Tables XV and XVI show the average rewards of each method when evaluated with various traffic models under simple- and practical-network conditions. We carried out 20 calculations with random initial conditions and set the same random seeds for all evaluations. We used agents trained with the mixed model as described in Fig. 7. We also used 10 types of various traffic models for evaluation as described in the first columns of Tables XV and XVI. The row "Mix all Traffic" corresponds to the results in Fig. 8. The row "Ave. all traffic models" means the average results of 10 types of traffic models.

For simple-network conditions, QMIX and VDN performed better than IQL for the average of all traffic models. SA also performed lower than other dynamic allocation methods. In QMIX and VDN, since the mixed model's performance and the average of all model's performance are approximately equal, training with the mixed model was effective for many traffic patterns. Moreover, the performance of QMIX and VDN does not decrease even for traffic models with the anomaly.

TABLE XVI
EVALUATION OF AVERAGE REWARDS FOR VARIOUS TRAFFIC MODELS IN PRACTICAL NETWORK (TRAINING BY **MIXED** MODEL)

Traffic Models	QMIX, Mix	VDN, Mix	IQL, Mix	Static Allocation (SA)
ARMA	0.67 ± 0.07	0.58 ± 0.08	0.20 ± 0.32	0.20 ± 0.21
ARMA with other parameters	0.70 ± 0.04	0.61 ± 0.06	0.23 ± 0.24	0.31 ± 0.14
SARIMA	0.67 ± 0.05	0.60 ± 0.07	0.19 ± 0.20	0.16 ± 0.20
Poisson	0.66 ± 0.05	0.58 ± 0.05	0.11 ± 0.32	0.08 ± 0.13
Random	0.67 ± 0.05	0.58 ± 0.07	0.15 ± 0.36	0.13 ± 0.12
ARMA with anomaly	0.66 ± 0.04	0.57 ± 0.05	0.22 ± 0.15	0.07 ± 0.10
SARIMA with anomaly	0.69 ± 0.05	0.61 ± 0.06	0.22 ± 0.16	0.13 ± 0.19
Poisson with anomaly	0.72 ± 0.06	0.63 ± 0.06	0.30 ± 0.18	0.27 ± 0.14
Random with anomaly	0.66 ± 0.04	0.57 ± 0.05	0.22 ± 0.15	0.07 ± 0.10
Mix All Traffic	0.68 ± 0.06	0.60 ± 0.06	0.26 ± 0.15	0.15 ± 0.12
Ave. All Traffic Models	0.68 ± 0.06	0.59 ± 0.06	0.21 ± 0.24	0.16 ± 0.17

TABLE XVII
EVALUATION OF AVERAGE REWARDS FOR VARIOUS TRAFFIC MODELS IN PRACTICAL NETWORK (TRAINING BY **EACH** MODEL)

Traffic Models	QMIX, ARMA	QMIX, ARMA w/ ano.	QMIX, Poisson	QMIX, Random	QMIX, Mix
ARMA	0.57 ± 0.10	0.58 ± 0.13	0.69 ± 0.06	0.63 ± 0.07	0.67 ± 0.07
ARMA with other parameters	0.59 ± 0.08	0.65 ± 0.08	0.72 ± 0.07	0.67 ± 0.06	0.70 ± 0.04
SARIMA	0.56 ± 0.09	0.62 ± 0.11	0.69 ± 0.05	0.65 ± 0.06	0.67 ± 0.05
Poisson	0.43 ± 0.13	0.56 ± 0.14	0.65 ± 0.06	0.60 ± 0.05	0.66 ± 0.05
Random	0.42 ± 0.09	0.59 ± 0.13	0.67 ± 0.07	0.62 ± 0.07	0.67 ± 0.05
ARMA with anomaly	0.44 ± 0.08	0.60 ± 0.10	0.67 ± 0.04	0.62 ± 0.06	0.66 ± 0.04
SARIMA with anomaly	0.55 ± 0.08	0.62 ± 0.08	0.69 ± 0.06	0.64 ± 0.05	0.69 ± 0.05
Poisson with anomaly	0.59 ± 0.05	0.64 ± 0.12	0.73 ± 0.05	0.69 ± 0.05	0.72 ± 0.06
Random with anomaly	0.44 ± 0.08	0.60 ± 0.10	0.67 ± 0.04	0.62 ± 0.06	0.66 ± 0.04
Mix All Traffic	0.52 ± 0.05	0.58 ± 0.12	0.69 ± 0.05	0.65 ± 0.05	0.68 ± 0.06
Ave. All Traffic Models	0.51 ± 0.11	0.60 ± 0.11	0.69 ± 0.06	0.64 ± 0.06	0.68 ± 0.06

Therefore, QMIX and VDN training with the mixed model obtains a generalization performance for traffic demands. In contrast, QMIX and VDN underperformed in the Poisson and Random models because these models include the traffic patterns which were not experienced. IQL sufficiently performed on the mixed traffic model but unsuccessfully performed on other traffic models. This shows that QMIX and VDN learn more superior policies through cooperative learning.

For practical-network conditions, the results are mostly the same as those under simple-network conditions. As mentioned in Section VI-E, though the average reward for the practical network was higher than that for the simple network, both reward values are not directly comparable. Comparing their performances under simple- and practical-network conditions, QMIX and VDN for the Poisson and Random models improved. We consider that, since freedom of allocation improved as the numbers of nodes and links increased in the practical network, more flexible allocation has become possible even when traffic fluctuates.

Next, we comprehensively analyzed the relationship between the generalization performance for traffic demands and the traffic models used during training. This evaluation can determine the best traffic models used during training for obtaining a generalization performance for traffic demands. Table XVII shows the average rewards of QMIX when we used various traffic models for training and evaluation under practical-network conditions. We adopted five types of traffic models for training: ARMA, ARMA with the anomaly, Poisson, Random, and Mixed model. As in the evaluation in Table XVI, we carried out 20 calculations with random initial conditions and set the same random seeds for all evaluations.

When the ARMA model was used for training, the performance of QMIX did not decrease with the ARMA model with other parameters and the SARIMA model. The performance for some models with the anomaly option was not decreased because of the normalization in the traffic generation process as described in Section VI-B. However, the performance decreased with unknown traffic, such as the Poisson model, the Random model, and models with anomaly options.

When the ARMA model with an anomaly was used for training, all traffic models performed better than when the ARMA model was used for training. This is because agents experience more traffic changes and learn how to act to avoid congestion when anomaly traffic occurs in advance.

When the Poisson, Random, and mixed models were used for training, QMIX performed outstandingly in all evaluation conditions. Moreover, when the ARMA model was used for evaluation in the first row of Table XVII, the average reward for training with the mixed model was higher than that for training with the ARMA model. Therefore, we conclude that training with the mixed model is suitable for obtaining high generalization performance for traffic demands. We consider that the reason is that the mixed model contains various traffic fluctuation patterns. While the mixed model performs well, each traffic model needs to be prepared so that it is composed of a part of a mixed model and assumes user traffic patterns when a mixed model for training is generated. On the other

hand, interestingly, the Poisson model for training performed equal to or better than the mixed model. We assume the reason to be that this model also contains various traffic changes. Since the Poisson model is easy to generate, training with this model will always be beneficial if it leads to a high generalization performance for various traffic demands. To analyze the results in more detail, it is necessary to solve the interpretability problem of DRL, which is an unsolved challenge in machine learning theory and is a future work of this paper.

J. Discussion

We discuss the future work of the proposed method. We revealed the effectiveness of the proposed method, but there are still important challenges that should be researched for applying our method in a commercial environment.

A major challenge is the generalization of agent training, i.e., an agent trained in one environment might not perform in other environments not experienced during training. For example, when changing user placement, the number of VNs, and the physical network topology, or when observing new traffic patterns, the agent should retrain from the beginning. This is a general challenge in machine learning, not unique to DRL-based dynamic allocation. As mentioned in Section VI-F, since the proposed method takes a long time to train agents, such as a few hours or a day, the re-training could be a significant problem when the network conditions are frequently changed. While we show that training by the mixed or Poisson model achieves generalization performance for traffic demands, whole new traffic patterns will possibly emerge with the development of 5G and other technologies in the future. One potential solution is grouping VNs, which can keep the number of VN groups constant (e.g., using [12]). This grouping will also lead to improving scalability. Another solution is aggregating all traffic to a small number of flows and handling only predictable flow patterns (e.g., using [41]). Although these grouping and aggregation are effective to limit the demand patterns, the performance of allocation will be decreased due to coarseâĂŘgrained allocation. The other solution is transfer learning, which focuses on storing knowledge gained while solving one domain and applying it to a different domain so that it can be learned efficiently.

We also discuss the challenge of interpretability of machine learning related to the challenge of generalization of machine learning. Since the DNN model included in the DRL is a black box, it is difficult to explain why the DNN outputs the results. For example, in this paper, it is impossible to understand why the Poisson model achieved the best performance. When applying these DRL-based techniques in actual network operations, the challenge also arises that the operator cannot trust the agents' outputs.

The other challenge is the ideal migration. As mentioned in Section III-A, the proposed method assumes ideal VM migration, i.e., the system can immediately reallocate VM without interrupting the running service. Note that the proposed method can be applied even if the migration takes a long time. However, when migration becomes a bottleneck, the dynamic allocation has difficulty following the demand fluctuations. One possible solution that a control algorithm can contribute is developing a more realistic penalty function of VM migration taking into account the availability and sustainability of the service (e.g., using [42]).

VII. Conclusion

We proposed a dynamic virtual network (VN) allocation method based on cooperative multi-agent deep reinforcement learning (Coop-MADRL). This method can quickly optimize the network resources even when traffic demands change drastically by applying MADRL for dynamic VN allocation. It can also reduce the agents' constraint violations such as network congestion and server overload and reduce the reallocation such as virtual machine (VM) migration by introducing a cooperative element for MADRL. Simulations revealed that the proposed dynamic VN allocation method can reduce the maximum server and link utilization and drastically reduce the constraint violations compared with that of a static VN allocation method under practical-network conditions. In contrast, the evaluation also revealed that the Exhaustive Search (ES) that maximizes the reward at each time does not necessarily maximize the average rewards when the traffic demands fluctuate. Moreover, the computation time of the proposed method was less than 1 s, which is significantly shorter than that of ES. As a result, we revealed that the proposed method simultaneously enables efficient and immediate dynamic VN allocation. Finally, we evaluated the generalization performance for various traffic demands. The results revealed that the agent training with mixed various traffic models could achieve a high generalization performance for all traffic models.

For future work, we plan to evaluate the performance of the proposed method in more complicated use-cases, e.g., SFC, in real-world demands and applications, and a testbed environment. We also plan to improve the interpretability of DRL by analyzing the relationship between the network state, the agent's actions, and the allocation results in detail. Moreover, we plan to evaluate the methods involving the approach described in Section VI-J.

References

[1] A. Suzuki, R. Kawahara, and S. Harada, "Cooperative multi-agent deep reinforcement learning for dynamic virtual network allocation," in *Proc. ICCCN*, 2021, pp. 1–11.

[2] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.

[3] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[4] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, 4th Quart., 2013.

[5] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[6] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1409–1434, 2nd Quart., 2019.

[7] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *Proc. IEEE NetSoft*, 2015, pp. 1–9.

[8] R. Mijumbi, J.-L. Gorricho, J. Serrat, M. Claeys, J. Famaey, and F. De Turck, "Neural network-based autonomous allocation of resources in virtual networks," in *Proc. IEEE EuCNC*, 2014, pp. 1–6.
[9] R. Mijumbi, J.-L. Gorricho, J. Serrat, M. Shen, K. Xu, and K. Yang, "A neuro-fuzzy approach to self-management of virtual network resources," *Expert Syst. Appl.*, vol. 42, no. 3, pp. 1376–1390, 2015.
[10] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "DeepViNE: Virtual network embedding with deep reinforcement learning," in *Proc. IEEE INFOCOM Workshops*, 2019, pp. 879–885.
[11] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1040–1057, Jun. 2020.
[12] A. Suzuki and S. Harada, "Safe multi-agent deep reinforcement learning for dynamic virtual network allocation," in *Proc. IEEE GLOBECOM*, 2020, pp. 1–7.
[13] X. Zheng, Y. Zhang, H. Zhang, and Q. Xue, "An RBF neural network–based dynamic virtual network embedding algorithm," *Concurrency Comput. Pract. Exp.*, vol. 31, no. 23, 2019, Art. no. e4516.
[14] C. K. Dehury and P. K. Sahoo, "DYVINE: Fitness-based dynamic virtual network embedding in cloud computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1029–1045, May 2019.
[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[16] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.
[17] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. ACM HotNets*, 2016, pp. 50–56.
[18] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI*, 2016, pp. 2094–2100.
[19] G. J. Laurent, L. Matignon, and L. Fort-Piat, "The world of independent learners is not Markovian," *Int. J. Knowl. Based Intell. Eng. Syst.*, vol. 15, no. 1, pp. 55–64, 2011.
[20] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, "A survey on virtual machine migration: Challenges, techniques, and open issues," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1206–1243, 2nd Quart., 2018.
[21] A. Agache *et al.*, "Firecracker: Lightweight virtualization for serverless applications," in *Proc. NSDI*, 2020, pp. 419–434.
[22] A. Suzuki, R. Kawahara, M. Kobayashi, S. Harada, Y. Takahashi, and K. Ishibashi,"Extendable NFV-integrated control method using reinforcement learning," *IEICE Trans. Commun.*, vol. E103.B, no. 8, pp. 826–841, 2020, doi: 10.1587/transcom.2019EBP3114.
[23] E. Oki, *Linear Programming and Algorithms for Communication Networks*. Boca Raton, FL, USA: CRC Press, 2012.
[24] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem," *Electron. Notes Discr. Math.*, vol. 52, pp. 213–220, Jun. 2016.
[25] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of Markov decision processes," *Math. Oper. Res.*, vol. 27, no. 4, pp. 819–840, 2002.
[26] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs*. Cham, Switzerland: Springer, 2016.
[27] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht, "Comparative evaluation of multi-agent deep reinforcement learning algorithms," 2020, *arXiv:2006.07869*.
[28] M. TAN, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proc. ICML*, 1993, pp. 487–494.
[29] A. Tampuu *et al.*, "Multiagent cooperation and competition with deep reinforcement learning," *PloS One*, vol. 12, no. 4, 2017, Art. no. e0172395.
[30] P. Sunehag *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proc. AAMAS*, 2018, pp. 2085–2087.
[31] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proc. ICML*, 2018, pp. 4295–4304.
[32] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *Proc. AAAI Fall Symp. Series*, 2015, pp. 1–7.
[33] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *Proc. NIPS Workshop*, 2014, pp. 1–9.
[34] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. NIPS*, 2019, pp. 8026–8037.
[35] M. Samvelyan *et al.*, "The StarCraft multi-agent challenge," 2019, *arXiv:1902.04043*.
[36] "GLPK." [Online]. Available: https://www.gnu.org/software/glpk/ (accessed Jan. 18, 2022).
[37] R. Summerhill, "The new Internet2 network," in *Proc. GLIF Meeting*, 2006, pp. 1–26.
[38] F. Metzger, T. Hoßfeld, A. Bauer, S. Kounev, and P. E. Heegaard, "Modeling of aggregated IoT traffic and its application to an IoT cloud," *Proc. IEEE*, vol. 107, no. 4, pp. 679–694, Apr. 2019.
[39] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0—Survivable network design library," *Netw. Int. J.*, vol. 55, no. 3, pp. 276–286, 2010.
[40] L. Espeholt *et al.*, "IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures," in *Proc. ICML*, 2018, pp. 1407–1416.
[41] Y. Takahashi *et al.*, "Separating predictable and unpredictable flows via dynamic flow mining for effective traffic engineering," *IEICE Trans. Commun.*, vol. 101, no. 2, pp. 538–547, 2018.
[42] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *Proc. IFIP Netw. Conf.*, 2015, pp. 1–9.

Akito Suzuki (Member, IEEE) received the B.E. degree in electronic and physical systems and the M.E. degree in nanoscience and engineering from Waseda University, Tokyo, Japan, in 2013 and 2015, respectively. Since joining NTT Laboratories in 2015, he has been engaged in research on network control and traffic engineering. His current research interests include network functions virtualization, mathematical optimization, and machine learning. He is a member of IEICE.

Ryoichi Kawahara (Member, IEEE) received the M.E. degree in automatic control and the Ph.D. degree in telecommunication engineering from Waseda University, Tokyo, Japan, in 1992 and 2001, respectively. He joined NTT Laboratories in 1992, and was engaged in research on traffic control for telecommunication networks, traffic measurement and analysis for IP networks, and network management, for 26 years. He is currently a Professor with the Department of Information Networking for Innovation and Design, Faculty of Information Networking for Innovation and Design, Toyo University. He received the Telecom System Technology Award from The Telecommunications Advancement Foundation in 2010 and the Best Paper Awards from IEICE in 2003 and 2009. He is a member of IEICE and ORSJ.

Shigeaki Harada (Member, IEEE) received the B.E., M.I.S., and Ph.D. degrees in information science from Tohoku University in 2001, 2003, and 2006, respectively. Since joining NTT Laboratories in 2006, he has been engaged in research on traffic analysis and traffic control in IP networks, and network management. He is a member of IEICE.