IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 19, 2021, accepted April 5, 2021, date of publication April 12, 2021, date of current version April 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3072435

Multi-Agent Reinforcement Learning-Based Resource Management for End-to-End Network Slicing

YOHAN KIM[1], (Student Member, IEEE), AND HYUK LIM[2], (Member, IEEE)
[1]School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology (GIST), Gwangju 61005, Republic of Korea
[2]AI Graduate School, Gwangju Institute of Science and Technology (GIST), Gwangju 61005, Republic of Korea

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

This work was supported by the Agency for Defense Development, South Korea, under Grant UD200007ED.

ABSTRACT To meet the explosive growth of mobile traffic, the 5G network is designed to be flexible and support multi-access edge computing (MEC), thereby improving the end-to-end quality of service (QoS). In particular, 5G network slicing, which allows a physical infrastructure to split into multiple logical networks, keeps the balance of network resource allocation among different service types with on-demand resource requests. However, achieving effective resource allocation across the end-to-end network is difficult due to the dynamic characteristics of slicing requests such as uncertain real-time resource demand and heterogeneous requirements. In this paper, we develop a reinforcement learning (RL)-based dynamic resource allocation framework for end-to-end network slicing with heterogeneous requirements in multi-layer MEC environments. We first design a hierarchical MEC architecture and formulate a resource allocation problem for the end-to-end network slicing as an optimization problem using the Markov decision process (MDP). Using proximal policy optimization (PPO), we develop independently-collaborative and jointly-collaborative dynamic resource allocation algorithms to maximize resource efficiency while satisfying the QoS of slices. Experimental results show that the proposed algorithms can recognize the characteristics of slice requests and coming resource demands and efficiently allocate resources with a high QoS satisfaction rate.

INDEX TERMS 5G, network slicing, multi-access edge computing, network resource management, multi-agent reinforcement learning.

I. INTRODUCTION

In recent years, the exponential growth of mobile services and the advances in the internet of things (IoT) and smart devices has presented significant challenges in the 5G. There is increased data traffic due to the constantly emerging mobile applications, such as augmented reality, face recognition, and 3D video streaming. Moreover, 5G networks need to meet the strong latency requirements of mission-critical services with the rapid development of edge applications such as cyber-physical systems and autonomous driving applications. Since the services have very different resources and deployment requirements, 5G networks need flexible network architectures to achieve high performances for different use cases. Network slicing, which is a method of creating multiple logical networks over a shared physical infrastructure, has been designed as a key technology to address such requirements [1]. Each network slice can have its own network architecture, including on-demand service application, resource capacity, and control policy, to balance the disparate requirements between heterogeneous services. For example, The dynamic resource allocation framework of network slicing can support real-time service requirements by considering service characteristics and time-varying demands of different slices [2]. While this vertical network slicing originally focused on core networks, it has expanded over time to end-to-end network slicing, supporting radio access networks and wireless interfaces [3]. End-to-end network slicing frameworks implemented in a multi-region and multi-tenant manner are well suited to the provision of dynamic on-demand resource orchestration to optimize all cross-layer functionalities in both access and core networks. In the end-to-end approach, the service chain, including virtual network functions (VNFs) and corresponding resources,

The associate editor coordinating the review of this manuscript and approving it for publication was Utku Kose.

is defined to establish end-to-end connections from the access side to the core side. If access network and core network resources are managed separately, performance optimization cannot be achieved. For example, the excessive offloading of the workload of slices to remote data centers without considering appropriate deployment depending on various service application scenarios causes performance bottlenecks because of limited resources, thereby degrading the quality of service (QoS).

To handle the mobile workload very close to network edge, there have been efforts to introduce multi-access edge computing (MEC) to the 5G networks [4]. MEC offers computing capabilities close to mobile devices, providing efficient overhead and delay management by offloading the workload, deployed as VNFs, to neighbor servers in a hierarchical structure. It has recently been expected that combining MEC and network slicing in 5G networks will ensure efficient network operation and service delivery and address heterogeneous requirements of various service types [5]. Nevertheless, research on dynamic resource management for end-to-end network slicing in 5G environments based on MEC is still lacking. If the slices excessively provision more resources at the edge to reduce the latency selfishly, the users may experience a large processing delay in the server with large amounts of computing load. Also, the efficiency of resource utilization is seriously degraded in high-layer networks with low utilization. There are pressing needs in an end-to-end 5G network to improve QoS and resource utilization with limited resources by considering the independent characteristics of each slice. In this context, the purpose of this paper is to achieve the layer-differentiated resource allocation to the different slices with heterogeneous requirements, taking into account the hierarchy of the MEC from the core to the edge. A dynamic and smart multi-layer orchestration capability is required to reserve and allocate virtual resources for slices at the right time in a hierarchical structure.

Reinforcement learning (RL) has been applied to network slicing management problems for 5G networks in [2], [6], [7]. RL is a type of machine learning technique that enables an agent to learn an optimal policy through interaction with an environment. In recent years, RL with deep learning capability such as deep Q-network has been applied to complex network management environments. The works have focused on the dynamic resource allocation for the management of interconnected resources in slices such as computing, storage, and bandwidth to enhance the performance of the systems. However, most of the works did not consider the end-to-end service delay as a performance measure of resource management. Note that guaranteeing the service delay is important in delay-sensitive 5G applications. Also, because each slice with different traffic characteristics and QoS requirements competes for more resources, it is essential to coordinate network slices to achieve network-wide performance enhancement. A certain slice's optimal policy may adversely affect the performance of other slices if it ignores the other slices' behaviors and states.

In this paper, we focus on RL enabling inter-slice cooperation to maximize resource efficiency while satisfying the QoS of the slices in multi-layer MEC environments. We first design a hierarchical architecture to evaluate the end-to-end delay of slices and the efficiency of the required resources depending on the characteristic of the slices. In our model, service delay is considered as an end-to-end delay, including processing delay and transmission delay. The resource allocation problem is modeled as the Markov decision process (MDP) for end-to-end network slicing with RL elements such as local and global states, actions, and a reasonable reward. We develop an RL framework to implement dynamic resource allocation and propose two cooperative PPO-based resource allocation algorithms to maximize resource efficiency by considering different characteristics of the slices. Based on a decentralized execution framework, each proposed algorithm works through distributed and centralized training, respectively. We conduct a performance evaluation in a typical MEC environment to analyze the implicit relationship between the state and action that we designed. Our model well explains the behavior of different types of services in the end-to-end network slicing environment, and the proposed algorithms improve resource efficiency and QoS.

The remainder of this paper is organized as follows. Section II reviews the related work. Section III introduces the considered hierarchical MEC architecture. Section IV illustrates the proposed resource allocation algorithms and heuristics. Section V evaluates the proposed algorithms in a typical network scenario. Finally, Section VI concludes the paper.

II. RELATED WORK

Many research interests from academia and industry were attracted to the MEC concept because of its ability to deploy cloud resources to the edge providing fast and powerful computing, energy efficiency, and storage capacity to end-users [4], [12]–[19]. According to [12], the European Telecommunications Standards Institute (ETSI), which launched the MEC concept in 2014, illustrates ways to deploy and integrate MEC into the 5G system based on the 3rd Generation Partnership Project (3GPP) 5G specifications. A study focused on the fundamental key enabling technologies of MEC taking into account the ETSI reference architectures [4]. A comprehensive survey of definitions, architectures, and deployments in the MEC area is presented in the literature [13], [14]. In [15], Mach and Becvar discussed integrating MEC functionalities to mobile networks and standardization. It focused on computation offloading, including allocation of computing resources within the MEC and mobility management. In [16], Mach and Becvar developed a MEC latency model and an energy consumption model and reviewed the joint radio and computational resource management for different types of MEC systems. In [17], a survey on the studies that have applied the game theory models to MEC environments is conducted, discussing optimizing and balancing resources and various trade-offs in MEC scenarios by applying theoretical model games.

TABLE 1. Comparison of related papers on network slicing with MEC.

Reference	Main objective	End-to-end network slicing	Scenario				Latency	Solution
			Radio	Edge	Transport	Core		
Cominardi *et al.* [5]	Integrating MEC with network slicing	✓	✓	✓	✓	✓	✓	-
Ksentini and Frangoudis [8]	Integrating MEC as a sub-slice	✓	✓	✓	✓	✓	✓	-
D'Oro *et al.* [9]	Instantiating heterogeneous slice services on edge devices	×	✓	✓	✓	✓	×	-
Sun *et al.* [10]	Minimizing content download latency, providing high data rate guarantee	×	✓	✓	×	×	✓	Game theory
Xiang *et al.* [11]	Minimizing latency	×	✓	✓	✓	×	✓	Heuristic
Kim *et al.* [2]	Maximizing profit, satisfying QoS	×	✓	✓	✓	✓	×	RL
Huynh *et al.* [6]	Maximizing accepted slice request	×	✓	✓	✓	✓	×	RL
Wang *et al.* [7]	Minimizing SLA violations, maximizing resource utilization	✓	✓	✓	✓	✓	×	RL

With the explosive growth of mobile devices in 5G, there is a need for the telecommunications industry for a new architecture supporting the performance requirements of various services in MEC environments. To meet such requirements, several standards developing organizations (SDOs), such as ETSI and 3GPP, have recently been working on harmonizing MEC and network slicing [20]. Based on the definitions of those architectures from the SDOs, recent studies proposed network slicing frameworks for MEC environments recently [5], [8], [9]. A summary of the related works on network slicing with MEC is provided in Table 1. In [5], Cominardi *et al.* addressed integrating MEC with network slicing based on an elaboration of ETSI. New solutions are proposed to evolve the current MEC framework toward end-to-end multi-domain network slicing support in 5G networks. Based on the latest ETSI and 3GPP specifications, Ksentini and Frangoudis in [8] proposed an MEC orchestration architecture to enable integrating MEC as a sub-slice. They proposed a new design considering both the multi-tenancy model and in-slice deployment. In [9], an MEC slicing framework is proposed to instantiate heterogeneous slice services on edge devices. Moreover, a distributed algorithm based on similarity among edge nodes and resource virtualization is proposed to instantiate slices faster because the problem of optimally instantiating joint MEC slices without incurring over-provisioning of resources is NP-hard. In [21], Balasubramanian *et al.* proposed a unified service architecture to enable mobility management of MEC and seamless handover between heterogeneous network slices. The proposed model provided subscription-based connectivity services for end-users to improve packet delivery ratio while guaranteeing latency requirements.

Moving on from the existing survey on the combination of MEC and network slicing, the purpose of this paper is to address the resource allocation problem of end-to-end network slicing in response to the dynamic traffic and strict service requirements of network slices. Several studies have been paying attention to resource provisioning optimized for service using network slicing in an MEC environment [10], [11]. In [10], a hierarchical radio resource allocation architecture is proposed to reduce the burden of centralized resource allocation burden. A Stackelberg game is established with the global radio resource manager as the leader and the local radio resource managers as followers to achieve optimization of the resource allocation between two network slices with heterogeneous performance metrics in fog radio access networks slicing. The resource allocation problem considered in the paper only focuses on radio resources, and it is infeasible to keep the changing resource distribution because of the huge operational expenditure. In [11], Xiang *et al.* proposed an optimization algorithm to perform a joint slicing of wireless network and edge computation resources, minimizing the total latency of transmitting, outsourcing, and processing user traffic into account. If the resource allocation problem is formulated to minimize the delay, the slices may greedily consume the resources of low-layer servers without appropriately placing VNFs to other layers of the MEC network. Besides, traffic dynamics, affecting resource allocation in real-time, is not explicitly incorporated into the formulation in the paper.

RL has been considered the most effective way to maximize the long-term reward for dynamic systems among dynamic resource allocation methods [22]. In recent studies, the RL algorithms have been applied successfully as an alternative to model-based optimization, which is difficult to handle because of dynamics complexity involving slice requests and resource allocation [6], [7], [23]–[25]. In [2], we focused on the dynamic resource allocation of network slicing against upcoming resource demand by applying Q-learning. The study aims at maximizing the profit of slices while ensuring the QoS requirements of end-users. This solution is mainly designed for a single small-scale network, which does not account for the end-to-end delay in MEC environments. Besides, it does not consider the cooperation between slices by performing learning within a single slice. In [6], Huynh *et al.* proposed the network slicing framework with the deep dueling neural network taking into account radio, computing, and storage resources in 5G networks.

The algorithm only chooses whether to accept or reject the slice request arrival without determining the amount of resources based on the traffic demand. In [7], Wang *et al.* leveraged deep RL to enable dynamic adjustment of the resources and simultaneously maximize resource utilization. A deep convolutional neural network is adopted to solve the optimization problem under a complex state space effectively. The model does not consider the delay that could occur in the end-to-end slicing environment. Hence, the resource status does not affect each other, and only the satisfaction of each VNF's demand is considered.

In this paper, we propose RL-based algorithms to determine the amount of combined MEC resources (i.e., computing and bandwidth) across the end-to-end network slicing according to real-time traffic demand and delay constraints. This RL-based approach can be effectively applied to practical applications and solve the resource allocation problem as an alternative to the model-based optimization because of its low complexity. Unlike myopic solutions that consider only the resource demand at a given moment, it is effective in optimizing QoS satisfaction against upcoming resource demand using only the current information. Our algorithm enables the recognition of the relationship between service types and the hierarchy of MEC from the edge to the core network and the automation of dynamic resource allocation to meet the QoS of end-to-end network slicing as much as possible. Furthermore, the proposed multi-agent RL approach maximizes resource efficiency by enabling inter-slice cooperation in multi-layer MEC environments.

III. SYSTEM MODEL

In this section, we describe the system model used in this paper. We first introduce the reference network system architecture and define a resource allocation problem for the end-to-end network slicing as an MDP model. Table 2 summarizes the notation adopted in this paper.

TABLE 2. Frequently used notations.

Parameter	Definition
t	Time step
$n \in \mathcal{N}$	Set of network slices
$m \in \mathcal{M}$	Set of MEC nodes
$l \in \mathcal{L}$	Set of layers
l_m	Layer of node m
c_m	Computing capacity of node m
b_m	Output bandwidth of node m
f_n	Packet size of flows of slice n
$r_n(t)$	Data rate of slice n
ω_n	Processing density of slice n
$w_n(t)$	Workload of slice n
$\hat{d}_n$	Tolerable delay requirement of slice n
$k_n(t)$	Set of nodes associated with slice n
$r_{n,m}(t)$	Data rate of slice n passing through node m
$w_{n,m}(t)$	Workload of slice n processed on node m
$\alpha_{n,m}(t)$	Capacity of node m allocated to slice n
$\beta_{n,m}(t)$	Bandwidth of node m allocated to slice n
$\varphi_{n,m}(t)$	Processing delay of slice n on node m
$\phi_{n,m}(t)$	Transmission delay of slice n on node m
$d_{n,m}(t)$	Service delay of slice n on node m
$D_n(t)$	Total delay of slice n
o_t^n	Observation of slice n
κ_n	Service type of slice n
i_n	Ingress node of slice n
$\boldsymbol{\varrho}(t)$	Resource utilization vector of all nodes
$\boldsymbol{\nu}_n(t)$	Utilization vector of resource allocated to slice n
$\boldsymbol{\rho}_n(t)$	Proportion of offloaded workload of slice n
a_t^n	Action space of slice n
u^u	Minimum size of resource adjustment
u^α	Unit size of computing resource adjustment
u^β	Unit size of bandwidth resource adjustment
U	Number of sizes for resource adjustment
ξ_t^n	Reward of slice n
η	Weight factor of reward
q_n	Penalty of QoS violation of slice n
T	Simulation time
E	Number of episodes
K	Number of training epochs
τ	Length of the trajectory segment
ζ	Learning rate
ϵ	Clipping parameter
γ	Discount factor
ϑ	Smoothing parameter

A. END-TO-END NETWORK SLICING ARCHITECTURE

As shown in Figure 1, we consider the end-to-end network slicing system architecture as a hierarchy composed of MEC nodes, denoted as $m \in \mathcal{M} = \{1, 2, \cdots, M\}$, where M is the total number of the MEC nodes. Each MEC server placed in a certain MEC node can create multiple VNFs, processing the mobile devices' workload. We assume that the MEC nodes act as both the computing server and the switch. There are M nodes with capacity $c_m \in \{c_1, c_2, \cdots, c_M\}$ measured in CPU cycle/s. Let $b_m \in \{b_1, b_2, \cdots, b_M\}$ denote the available bandwidth at the MEC node's interface. We consider the tree topology typically used for performance assessment in MEC scenarios. Let $\mathcal{L} = \{1, 2, \cdots, L\}$ denote a set of layers, where L is the total number of the layers. The ingress point of flows belonging to slices can be one of the leaf nodes in the bottom layer of the network. The layer of MEC node m is denoted as $l_m \in \mathcal{L}$. We assume that the size of the layer from the core node up to all leaf nodes is the same.

We consider a network slicing infrastructure consisting of network slices, denoted as $n \in \mathcal{N} = \{1, 2, \cdots, N\}$, where N is the total number of the slices. Based on user requirements, different slices can be created to serve different types of user traffic (e.g., computation-intensive service or delay-sensitive service). Therefore, the network slices can be characterized by specific performance requirements, usually expressed using tolerable delay, data rate, and workload. Considering different service types, we define the $\hat{d}_n$ as a tolerable delay requirement of slice n. To characterize the data rate, we assume that the arrival rate of each flow in slice n follows a Poisson distribution with the parameter λ_n. The flow length follows an exponential distribution with the parameter μ_n. The packet size of each flow and the number of flows belonging to slice n are represented by f_n and $|f_n|$, respectively. These parameters can be used to obtain the total data rate of slice n at time step t, denoted as $r_n(t)$ (bit/s). We characterize each slice by the processing density ω_n (CPU cycle/bit), the CPU demand as per data rate because the processing resource demand is dependent on the service type. Then, $w_n(t) = r_n(t) \cdot \omega_n$ (CPU cycle/s) represents the total workload of slice n at time step t.

所為offload workload，就是指我將此服務的工作，因資料轉移等因素，將工作轉移至下一個node，這個工作的工作量

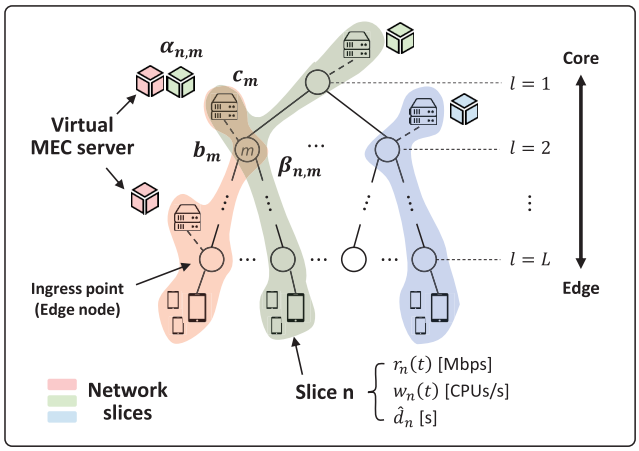


FIGURE 1. End-to-end network slicing system architecture.

B. INTEGRATION OF NETWORK SLICING AND MEC

Network slicing provides customized services to mobile users by integrating multiple resources of the MEC network. In general, a network slice runs VNFs optimally placed to fulfill a specific set of requirements and to offer customized services to realize a particular use-case scenario. For simplicity, we assume the scenario where each slice is associated with only one server at each layer and served by multiple servers across the entire layer. Let k_n denote a set of MEC nodes associated with slice n.

Since resource demand fluctuates over time, a sufficient amount of resources to handle the traffic must be dynamically allocated to the VNFs and links so that the QoS of the slices can be satisfied. The computing resource of the node m allocated to slice n at time step t is defined as $\alpha_{n,m}(t)$. The following constraints ensure that the allocated capacity of all slices placed on the MEC node m at time step t does not exceed the total capacity of the node m:

$$\sum_{n=1}^{N} \alpha_{n,m}(t) \leq c_m. \tag{1}$$

Note that if the amount of computing resource allocated to slice n by the node m is zero, then $m \notin k_n(t)$. Similarly, the bandwidth of the node m allocated to slice n is defined as $\beta_{n,m}$. Then, the constraint for bandwidth is written as follows:

$$\sum_{n=1}^{N} \beta_{n,m}(t) \leq b_m. \tag{2}$$

The hierarchical MEC system provides an offloading capability to enable processing workload far away from the core network. It allows supporting the control functionalities needed to provide computing capabilities to hierarchical edge nodes or to move VNF closer to end-users to satisfy their delay constraint. Also, if the computing capacity is insufficient on a certain server, the slice's workload can be offloaded to the server of the upper or lower layer connected to it. Let $w_{n,m}(t)$ denote the workload of slice n processed on the MEC node m at time step t. We assume that the offloaded workload is determined in proportion to the allocated computing resource according to the following:

$$w_{n,m}(t) = \rho_{n,m}(t) \cdot w_n(t), \tag{3}$$

where $\rho_{n,m}(t) = \alpha_{n,m}(t) / \sum_m \alpha_{n,m}(t)$ represents the proportion of the allocated computing resource, where $\sum_m \rho_{n,m}(t) = 1$. When the slice's workload is outsourced from the ingress point to the MEC node m, the total traffic passing through the MEC node depends on the amount of offloaded workload. Therefore, the data rate of slice n passing through the node m at time step t is defined as follows:

$$r_{n,m}(t) = \sum_{h,\ if\ m \in path(h)} \frac{w_{n,h}(t)}{\omega_n}, \tag{4}$$

where $path(h)$ represents the set of nodes on the path from the ingress point to the MEC node h.

Our goal is to satisfy the delay requirement of the slices in the hierarchical MEC network as much as possible. The data processing and transmission delay can be modeled as G/G/1 queue. For tractability, we use an M/M/1 model under the assumption of Poisson arrivals and exponential inter-departure times of the system. The processing delay $\varphi_{n,m}(t)$ and transmission delay $\phi_{n,m}(t)$ of the slice n on the server m at time step t are defined as follows:

$$\varphi_{n,m}(t) = \begin{cases} \dfrac{1}{\alpha_{n,m}(t) - w_{n,m}(t)}, & \text{if } m \in k_n(t), \\ 0, & \text{otherwise}, \end{cases} \tag{5}$$

$$\phi_{n,m}(t) = \begin{cases} \dfrac{1}{\beta_{n,m}(t) - r_{n,m}(t)}, & \text{if } m \in k_n(t), \\ 0, & \text{otherwise}. \end{cases} \tag{6}$$

The following constraint ensures that $\alpha_{n,m}(t)$ is higher than the $w_{n,m}(t)$ and the $r_{n,m}(t)$ does not exceed the $\beta_{n,m}(t)$, respectively:

$$\alpha_{n,m}(t) > w_{n,m}(t),\ \beta_{n,m}(t) > r_{n,m}(t),\ \forall n \in \mathcal{N}, \forall m \in \mathcal{M}. \tag{7}$$

Note that if the resource demand is larger than the allocated capacity, the delay becomes infinite, and the QoS is violated. The service delay of the slice n serviced by the MEC node m is determined by the sum of the processing and transmission delays occurring on the path:

$$d_{n,m}(t) = \varphi_{n,m}(t) + \sum_{h \in path(m)} \phi_{n,h}(t). \tag{8}$$

Therefore, the total delay of the slice n depends on the longest service delay among the MEC nodes:

$$D_n(t) = max_{m \in k_n(t)} d_{n,m}(t). \tag{9}$$

If the total delay $D_n(t)$ exceeds the delay requirement $\hat{d}_n$, the slice n violates the QoS at time step t. In other words, if the amount of allocated bandwidth and computing resources is insufficient to the service resource demand, the QoS of

some flows may be violated, in which a penalty may be imposed. If we simply formulate our problem to be a minimization problem of delay, the slices may greedily consume the resources of low-layer servers without appropriately placing VNFs to other layers of the MEC network. Therefore, we focus on deciding how much the MEC nodes' resources are allocated to adaptively place the workload at different layers of the MEC network and satisfy the delay requirement of many slices as possible.

C. MDP MODEL FOR RESOURCE ALLOCATION

In this section, we discuss the formulation of the resource allocation problem of end-to-end network slicing. The state observations are obtained from the real-time information of the end-to-end network slicing system. The resource management agent monitors the status of network slices and MEC nodes, including their resource allocation. Based on the obtained information, the agent makes a decision on whether to adjust the allocated resources for each slice to minimize their QoS violations. The decisions are made at each control interval. The value of the control interval depends on the practical application (e.g., from several seconds to tens of minutes). The environment returns a reward, which is calculated depending on the QoS and resource utilization of the slices, and then a new decision process will be activated. Our objective is to complete the decision process as soon as possible to ensure the delay requirement. To achieve the goal, we formulate this whole process as an MDP.

Suppose there are N slices, an MDP is modeled as a tuple $\langle o_1, \cdots, o_N, a_1, \cdots, a_N, \xi_1, \cdots, \xi_N, \mathcal{P}_1, \cdots, \mathcal{P}_N \rangle$, where o_n is the state space, a_n denotes the action space, ξ_n represents a reward function used in measuring the decision quality, and $\mathcal{P}_n$ is the state transition probability, modeling the environment dynamics. The following are the detailed descriptions of the MDP model.

1) State space: The observation of each slice n at the time step t can be represented by

$$o_t^n = \left[t, \kappa_n, i_n, \boldsymbol{\varrho}^\alpha(t), \boldsymbol{\varrho}^\beta(t), \boldsymbol{\nu}_n^\alpha(t), \boldsymbol{\nu}_n^\beta(t), \boldsymbol{\rho}_n(t)\right]. \quad (10)$$

where κ_n represents service type that determines delay constraint, packet size, and processing density, and i_n is the ingress node of the slice n. Let $\boldsymbol{\varrho} = [\varrho_1, \cdots, \varrho_M]$ be a vector with each element denoting the percentage of resource utilization of each MEC node belonging to slice n:

$$\begin{aligned} \varrho_m^\alpha(t) &= \sum_{n=1}^{N} \alpha_{n,m}(t)/c_m, \\ \varrho_m^\beta(t) &= \sum_{n=1}^{N} \beta_{n,m}(t)/b_m. \end{aligned} \quad (11)$$

The vector $\boldsymbol{\nu}_n = [\nu_{n,1}, \cdots, \nu_{n,M}]$ indicates the utilization of the resource allocated from the MEC node m to slice n and is a measure of the efficiency of resource allocation:

$$\begin{aligned} \nu_{n,m}^\alpha(t) &= \min(1,\ w_{n,m}(t)/\alpha_{n,m}(t)), \\ \nu_{n,m}^\beta(t) &= \min(1,\ r_{n,m}(t)/\beta_{n,m}(t)). \end{aligned} \quad (12)$$

The vector $\boldsymbol{\rho}_n = [\rho_{n,1}, \cdots, \rho_{n,M}]$ is the proportion of the offloaded workload of each MEC node belonging to slice n. Depending on our settings, each slice can be associated with the same number of MEC nodes as the number of layers. Therefore, we can set the number of elements of each vector equal to the number of layers.

2) Action space: The action a_t^n serves four functions for different purposes: (1) selecting a node to adjust resources, (2) deciding an adjustment size, and (3, 4) deciding whether to increase or decrease CPU and bandwidth resources. Thus, the action space a_t^n is given as follows:

$$a_t^n = a_n^s(t) \times a_n^u(t) \times a_n^\alpha(t) \times a_n^\beta(t), \quad (13)$$

where the action $a_n^s(t) \in \{1, 2, \ldots, L\}$ consists of actions as the number of MEC nodes to which slice n can be associated (i.e., the number of layers). The action $a_n^u(t) \in \{0, 1, \cdots, U\}$ decides unit size of resource adjustment, $u^\alpha = u^u \cdot 3^{a_n^u(t)}$ and $u^\beta = u^\alpha/\omega_n$, where u^u is minimum size of resource adjustment. The actions $a_n^\alpha(t) \in \{+u^\alpha, 0, -u^\alpha\}$ and $a_n^\beta(t) \in \{+u^\beta, 0, -u^\beta\}$ include the actions of adjusting the amount of computing and bandwidth resources, respectively. At each time step, the agent can sequentially increase or decrease the resources of all MEC nodes associated with the slice n by the unit size (i.e., u^α and u^β). Especially, in the case of bandwidth, the amount of resources of all nodes on the path from the ingress point to the MEC node a_n^s is simultaneously adjusted. Each action only affects one slice at a time, and the agent gets a reward for the action of the slice. The time step moves to the next step after completing the resource adjustment of all slices.

3) Reward function: The reward function should be defined to guide the agents to make decisions toward the direction of optimizing our objective, which is QoS maximization. An effective solution to avoid QoS violation is increasing resource utilization and decreasing the resource wastage simultaneously. Inspired by this solution, we define the reward function associated with resource allocation at time step t as follows:

$$\xi_t^n = \sum_{m=1}^{M} \nu_{n,m}^\alpha(t) + \sum_{m=1}^{M} \nu_{n,m}^\beta(t) + \eta \cdot q_n(t), \quad (14)$$

where the first case indicates the efficiency of resources, and $q_n(t)$ represents the QoS violation penalty, denoted as follows:

$$q_n(t) = \begin{cases} 0, & \text{if } D_n(t) \le \hat{d}_n(t), \\ -M, & \text{otherwise.} \end{cases} \quad (15)$$

The parameter η denotes a weight factor balancing the trade-off between the resource efficiency and the QoS. However, the resource allocation policy that maximizes the local reward ξ_t^n is not optimal for maximizing rewards for all slices. Therefore, we model the reward function as the common reward to improve resource efficiency and reduce

QoS violations of all slices as follows:

$$\xi_t = \sum_{n=1}^{N} \frac{\xi_t^n}{N}. \tag{16}$$

We can balance slices within a limited resource by returning the common reward according to each slice's behavior. In the next section, we will describe our algorithm to maximize the common reward.

IV. REINFORCEMENT LEARNING FOR END-TO-END NETWORK SLICING

For resource allocation for multiple slices, one may consider a single centralized agent that can fully observe the system state and determine the behavior of all slices. When the slices request a virtual network with specific QoS requirements, it is fully up to the provider to manage the slices' resources to ensure QoS. Therefore, it is reasonable that the agent is considered as a super-agent. Even though the super-agent-based decision may be the best in performance, it is usually impractical because it may cause a heavy signaling overhead. Furthermore, in practical implementations, it is difficult for algorithms to find an optimal solution because the action space increases exponentially as the number of slices N increases. Instead, in this paper, we adopt two decentralized approaches based on PPO algorithm, which is explained in the following section.

A. PROXIMAL POLICY OPTIMIZATION

In RL, an agent seeks to select an action that maximizes the accumulated reward over time when interacting with a complex and uncertain environment. One of the most noticeable results was achieved through the policy gradient method, which directly updates policy θ using policy gradient estimator:

$$\hat{g} = \hat{\mathbb{E}}_t\big[\nabla_\theta \log \pi_\theta(a_t|o_t)\hat{A}_t\big], \tag{17}$$

where π_θ represents a stochastic policy, and $\hat{A}_t$ represents the estimation of the true advantage at time step t. The estimator $\hat{g}$ is obtained by an objective function

$$J(\theta) = \hat{\mathbb{E}}_t\big[\log \pi_\theta(a_t|o_t)\hat{A}_t\big]. \tag{18}$$

The general actor-critic algorithms estimate the advantage based on a critic: $\hat{A}_t = \xi_t + \gamma V_\upsilon(o_{t+1}) - V_\upsilon(o_t)$, where the critic V_υ is the estimation of the true value function. The critic can be trained by adjusting the parameter υ to minimize the loss

$$J(\upsilon) = \hat{\mathbb{E}}_t\big[(y_t - V_\upsilon(o_t))^2\big], \tag{19}$$

where $y_t = \xi_t + \gamma V_{\bar{\upsilon}}(o_{t+1})$, and $V_{\bar{\upsilon}}$ is the target value function that is periodically updated with the latest υ. The optimization problem of the aforementioned objective function can be solved by stochastic optimization methods, such as gradient-based methods. However, performing stochastic policy gradient is often unstable because the magnitude of a gradient step in parameter space is often not proportional to its magnitude in policy space, leading to an excessively large step size in policy space. In addition, the original policy-based algorithms have low training efficiency because a collected trajectory is used to train the agent only one time.

To attain the reliable performance and data efficiency of the learning, Schulman *et al.* [26] proposed PPO, optimizing a clipped surrogate objective function that penalizes changes to the parameters that yield large policy changes. In PPO, the agent comprises an actor network and a critic network, which are responsible for decision making and policy evaluation, respectively, as shown in Figure 2. In other words, the actor updates the policy probability distribution with the direction suggested by the critic and determines the best action. The surrogate objective function can be represented as follows:

$$\begin{aligned} J'(\theta) &= \hat{\mathbb{E}}_t\big[\log \pi_\theta(a_t|o_t)\hat{A}_t\big] \\ &= \hat{\mathbb{E}}_t\big[\frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{old}}(a_t|o_t)}\hat{A}_t\big] \\ &= \hat{\mathbb{E}}_t\big[\psi_t(\theta)\hat{A}_t\big], \end{aligned} \tag{20}$$

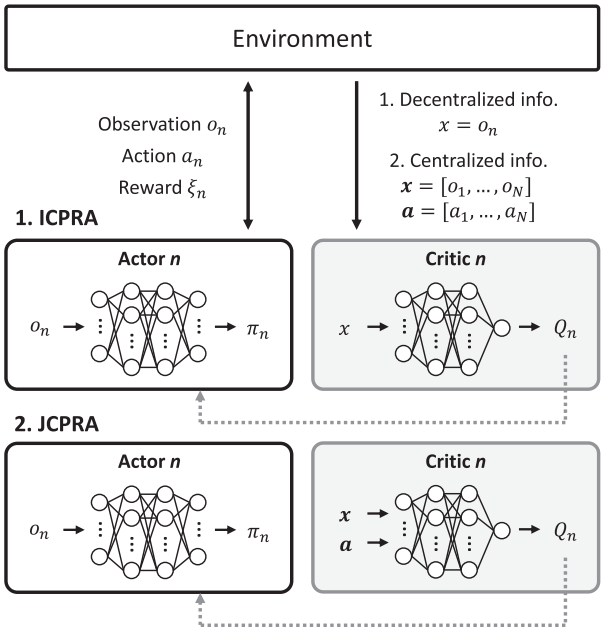


FIGURE 2. PPO-based resource allocation framework.

where $\psi_t(\theta)$ denotes the ratio of the current policy and old policy. Then, the surrogate objective function is clipped to avoid having too large a policy update by imposing a constraint on the difference between the new policy and the old one:

$$\begin{aligned} \bar{J}(\theta) &= \hat{\mathbb{E}}_t\big[\min(\psi_t(\theta)\hat{A}_t,\ \mathrm{clip}(\psi_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\big] \\ &= \hat{\mathbb{E}}_t\big[F(\psi_t(\theta),\ \hat{A}_t)\big], \end{aligned} \tag{21}$$

where ϵ is a clipping parameter, and $\mathrm{clip}(\psi_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ modifies the surrogate objective by clipping the policy probability ratio $\psi_t(\theta)$. In other words, the ratio $\psi_t(\theta)$ is clipped

Algorithm 1 ICPRA algorithm

Initialize critic V_{υ^n} and actor π_{θ^n}
Initialize the current policies $\pi_{\theta^n_{old}}$
Initialize buffer $\mathcal{B}$
for $e = 1$ to E **do**
  for $t = 1$ to T **do**
    if control interval **then**
      Obtain o_t^n
      Execute a_t^n according to $\pi_{\theta^n_{old}}(a_t^n|o_t^n)$
      Compute ξ_t according to (16)
      $\mathcal{B} \leftarrow \mathcal{B} \cup \{(o_t^n, a_t^n, \xi_t, \psi_t^n(\theta^n))\}$
    end if
    if $t/\tau = 1, 2, 3, \cdots$ **then**
      for $k = 1$ to K **do**
        Sample the data from buffer $\mathcal{B}$
        Compute $\hat{A}_t^n$ according to (22)
        Compute $\bar{J}(\theta^n)$ according to (23)
        Compute $J(\upsilon^n)$ according to (24)
        $\theta^n \leftarrow \theta^n + \zeta \nabla_\theta \bar{J}(\theta^n)$
        $\upsilon^n \leftarrow \upsilon^n - \zeta \nabla_\upsilon J(\upsilon^n)$
      end for
      $\theta^n_{old} \leftarrow \theta^n$ and $\bar{\upsilon}^n \leftarrow \upsilon^n$
      $\mathcal{B} \leftarrow \emptyset$
    end if
  end for
end for

Algorithm 2 JCPRA algorithm

Initialize Q_{υ^n}, π_{θ^n}, $\pi_{\theta^n_{old}}$ and $\mathcal{B}$
for $e = 1$ to E **do**
  for $t = 1$ to T **do**
    if control interval **then**
      Obtain o_t^n
      Execute a_t^n according to $\pi_{\theta^n_{old}}(a_t^n|o_t^n)$
      Compute ξ_t according to (16)
      $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\boldsymbol{x}_t, \boldsymbol{a}_t, \xi_t, \psi_t^n(\theta^n))\}$
    end if
    if $t/\tau = 1, 2, 3, \cdots$ **then**
      for $k = 1$ to K **do**
        Sample the data from buffer $\mathcal{B}$
        Compute $\hat{\boldsymbol{A}}_t^n$ according to (26)
        Compute $\bar{\boldsymbol{J}}(\theta^n)$ according to (25)
        Compute $\boldsymbol{J}(\upsilon^n)$ according to (27)
        $\theta^n \leftarrow \theta^n + \zeta \nabla_\theta \bar{\boldsymbol{J}}(\theta^n)$
        $\upsilon^n \leftarrow \upsilon^n - \zeta \nabla_\upsilon \boldsymbol{J}(\upsilon^n)$
      end for
      $\theta^n_{old} \leftarrow \theta^n$ and $\bar{\upsilon}^n \leftarrow \upsilon^n$
      $\mathcal{B} \leftarrow \emptyset$
    end if
  end for
end for

at $1-\epsilon$ or $1+\epsilon$ depending on whether the advantage $\hat{A}_t$ is positive or negative. This method has stability and reliability for policy updates and improves performance over a vanilla policy gradient method.

B. INDEPENDENTLY COOPERATIVE PPO-BASED RESOURCE ALLOCATION

We first propose an independently cooperative PPO-based resource allocation (ICPRA) for end-to-end network slicing. As shown in Figure 2, ICPRA allows each slice to independently update the actor and critic according to the individual trajectory $(o_t^n, a_t^n, \xi_t, \psi_t^n(\theta))$. Actor policies (π_θ) are trained to maximize global reward, given decentralized states, whereas the critic network estimates the value function (V_υ). Recall that the o_t^n and a_t^n refer to observations and actions of the slice n, respectively. Each o_t^n contains 6 elements per slice (i.e., current time step, service type, ingress node, resource utilization of associated nodes, efficiency of resource allocation, and proportion of resource allocation). The action a_t^n indicates how to control the computing and bandwidth resources of a selected node. Our objective is to obtain the optimized policy that maximizes the common reward ξ_t presented in (16).

Algorithm 1 presents an ICPRA's pseudo code. The actor network and the critic network are created with the parameters π_θ and V_υ, respectively. The agent has a buffer memory $\mathcal{B}$ to store the trajectories of interacting with the environment using policy $\pi_{\theta_{old}}$. Each slice sequentially collects trajectory at the moment of resource adjustment, where E is the number of episodes. This data collection period runs during τ, the length of the trajectory segment (where τ is less than the episode length). After the data collection period, the training period runs for K epochs. The following advantage function $\hat{A}_t^n$ is estimated by the generalized advantage estimation [27]:

$$\hat{A}_t^n = \delta_t^n + (\gamma\vartheta)\delta_{t+1}^n + \cdots + (\gamma\vartheta)^{\tau-t+1}\delta_{\tau-1}^n, \tag{22}$$

where γ is the discount factor, ϑ is the smoothing parameter, and $\delta_t^n = \xi_t + \gamma V_{\bar{\upsilon}^n}(o_{t+1}^n) - V_{\bar{\upsilon}^n}(o_t^n)$. The clipped surrogate objective function $\bar{J}(\theta^n))$ in (21) is calculated as follows:

$$\bar{J}(\theta^n) = \hat{\mathbb{E}}_t\big[F(\psi_t(\theta^n),\ \hat{A}_t^n)\big]. \tag{23}$$

The critic is also updated with the following loss function:

$$J(\upsilon^n) = \hat{\mathbb{E}}_t\big[(y_t^n - V_{\upsilon^n}(o_t^n))^2\big], \tag{24}$$

where $y_t^n = \hat{A}_t^n + V_{\bar{\upsilon}^n}(o_t^n)$. The algorithm iteratively updates the parameters in K training epochs using Adam optimizer [28] with learning rate ζ. After the training, θ^n_{old} and $\bar{\upsilon}^n$ are updated. However, the local critic of the ICPRA algorithm does not approximate the true global state-value function. Therefore, the advantage function considering all slices cannot be estimated accurately. It is difficult to estimate the influences of each slice's action on the common reward because only the state information is used.

C. JOINTLY COOPERATIVE PPO-BASED RESOURCE ALLOCATION

We propose a jointly cooperative PPO-based resource allocation (JCPRA) in centralized training with a distributed execution framework. Different from the ICPRA algorithm, the centralized critics estimate the joint action-value function Q_{υ^n} based on global information that includes the observation and action of all slices, whereas the distributed actors make decisions only based on local observations of the slices in the same way as the ICPRA as illustrated in Figure 2. We can update the policy of the actor network using the clipped loss function:

$$\bar{\boldsymbol{J}}(\theta^n) = \hat{\mathbb{E}}_t\big[F(\psi_t(\theta^n),\ \hat{\boldsymbol{A}}_t^n)\big], \tag{25}$$

where $\hat{\boldsymbol{A}}_t^n$ is the joint advantage function indicating how the behavior of all agents, including a given agent, contributes to the common reward:

$$\hat{\boldsymbol{A}}_t^n = \boldsymbol{\delta}_t^n + (\gamma\vartheta)\boldsymbol{\delta}_{t+1}^n + \cdots + (\gamma\vartheta)^{\tau-t+1}\boldsymbol{\delta}_{\tau-1}^n, \tag{26}$$

where $\boldsymbol{\delta}_t^n = \xi_t + \gamma Q_{\bar{\upsilon}^n}(\boldsymbol{o}_{t+1}, \boldsymbol{a}_{t+1}) - Q_{\bar{\upsilon}^n}(\boldsymbol{o}_t, \boldsymbol{a}_{t+1})$. The critic is also updated with the following loss function:

$$\boldsymbol{J}(\upsilon^n) = \hat{\mathbb{E}}_t\big[(\boldsymbol{y}_t^n - Q_{\upsilon^n}(\boldsymbol{o}_t, \boldsymbol{a}_t))^2\big]. \tag{27}$$

Note that the critic for each slice n is updated separately, but has the same joint action-value function. The detailed learning process of JCPRA is presented in Algorithm 2.

V. PERFORMANCE EVALUATION

A. SIMULATION SETUP

We obtained all numerical results from the executions on a server equipped with an Intel(R) Xeon(R) Gold 6136 CPU @ 3.00 GHz and NVIDIA Titan RTX GPU with 24 GB of memory and with CUDA 10.1. The RL environments were implemented with Python 3.3, Pytorch 1.6. Figure 3 shows the network topologies with the different number of layers. One of the leaf nodes, i_n, can be an ingress point of slice n, and the slice's workload is outsourced from the ingress point to the MEC nodes. For measuring the performance, we selected the binary tree topology, which is typically used for performance assessment in hierarchical MEC scenarios. The capabilities of servers close to the core are higher than those of servers close to the edge. Therefore, we set the available CPU capacity c_m to $18000 \cdot 2^{1-l_m}$ CPU cycles/s depending on the layer l_m of each node and the bandwidth b_m to 18000 Mbps. Unless otherwise stated, we set $N = 10$, $M = 7$, $L = 3$, and the initial values of all parameters are listed in Table 3.

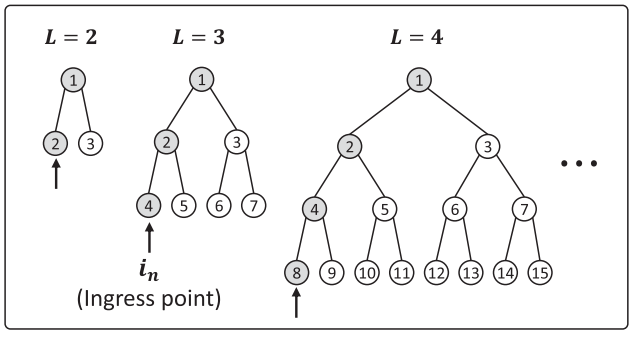


FIGURE 3. Network topologies for performance evaluation.

TABLE 3. Simulation parameters.

Network parameter	Initial value
$n \in \mathcal{N}$	$\{1, 2, \cdots, 10\}$
$m \in \mathcal{M}$	$\{1, 2, \cdots, 7\}$
$l \in \mathcal{L}$	$\{1, 2, 3\}$
c_m [CPU cycle/s]	$18000 \cdot 2^{1-l_m}$
b_m [Mbps]	18000
i_n	Random in $\{4, 5, \cdots, 7\}$
f_n [Mbit]	5, 200, 5
$\lvert f_n \rvert$	50, 50, 50
λ_n	Random in [30,40]
μ_n	Random in [20,40]
ω_n [CPU cycle/Mbit]	1, 0.5, 5
$\hat{d}_n$ [s]	0.01, 0.5, 0.5
u^u	100
U	3
η	1.2
T	100
E	2000
K	3
τ	20
ζ	0.0007
ϵ	0.15
γ	0.98
ϑ	0.95

We considered three common types of slices included in the 5G applications: automotive (low tolerable latency, small traffic, and low computation), data transmission (medium tolerable latency, large traffic, and medium computation), and utility (medium tolerable latency, low traffic, and large computation). We used f_n, λ_n, and μ_n to calculate the data rate $r_n(t)$ at each time step and the processing density ω_n to calculate the computation capacity $w_n(t)$. For example, the packet sizes f_n of requests from each type of services are set to 5, 200, and 5 Mbits, respectively. The flow arrival rate of each slice is governed by a Poisson process (λ_n), and the length of flow follows an exponential distribution (μ_n). The delay requirements for each service type are 0.01, 0.5, and 0.5 s, respectively. The ingress node i_n of slice n is randomly chosen. Check the different parameter settings for different slice types.

To implement the PPO-based algorithm, we used three fully-connected hidden layers with 256 units for the actor network and the critic network, respectively. We applied the rectified linear units (ReLU) activation function, which has proven to be reliable and fast learning. Depending on the environment settings, the size of the input layer of the actor network is 18. We can also obtain the size of the action space, which is the same as the number of $3 \times 3 \times 3 \times 3 = 81$. The number of inputs of the JCPRA's critic network is equal to the number of global information for all agents. Using Adam optimizer, we updated the prediction network with a learning rate $\zeta = 0.0007$. The discount factor γ and the

smoothing parameter ϑ for estimating the advantage function in (22) and (26) were set to 0.98 and 0.95, respectively. We chose reward weight $\eta = 1.2$ for the reward function. For the resource adjustment, we set the control interval to 5, u^u to 100, and U to 3. These parameter settings are fixed unless otherwise noted in this section.

We evaluated the performance of the proposed algorithms and compared it with that of other algorithms, which are fixed-resource (FR) algorithm, previous-demand (PD) algorithm, random action algorithm, and sequential fixing (SF) algorithm in [11]. The most straightforward algorithm, the FR strategy allocates sufficient resources exclusively and excessively. In other words, the algorithm statically allocates the same amount of resources as the peak-time resource demand. The PD strategy is based on allocating resources corresponding to the resource demand of the previous control interval. According to our setting, the algorithm dynamically allocates the same amount of resources as the amount multiplied by 1.2 to the previous demand. This algorithm is a typical dynamic resource allocation method. In the two greedy algorithms, resources are allocated to slices sequentially from edge to core. The random algorithm is based on randomly selecting actions of dynamic resource allocation. The SF algorithm focuses on minimizing the service delay. It determines the ranking of MEC nodes based on the probability of node selection. It also assigns a rank to the slices according to the data rate and delay requirements and sequentially allocates the MEC nodes to the slices.

B. TRAINING PERFORMANCES

To demonstrate the effectiveness of the proposed algorithms, we first observed the cumulative reward of the proposed algorithm for training episodes. In Figure 4, the red and blue curves show the total reward variation of JCPRA and ICPRA, respectively, over the whole training process. For comparison, we used the results derived by applying other algorithms. The reward is observed for 25 sampling intervals with an exponential moving average window of 20. As can be seen, the performance curves of the proposed algorithms increase at different rates as the training process goes on. The reward starts from a low initial value but is quickly converged within hundreds of episodes. The graph shows that the converged reward obtained by the proposed algorithms is significantly higher than that of other algorithms. ICPRA converges faster than JCPRA in the early stages of learning because it is more difficult to learn the policy of JCPRA, which has many inputs for the joint learning of the critic network. However, the converged result demonstrates that JCPRA performs better than ICPRA. Hence, we can find that JCPRA achieves a more reasonable policy compared with ICPRA by considering the interrelationships between the slices.

C. QoS UNDER DIFFERENT LEVELS OF NETWORK LOAD

Figure 5 shows the QoS satisfaction results according to the amount of network resources. To evaluate the performance according to different network loads, we set the peak CPU demand to 50%, 100%, 150%, and 200% compared with

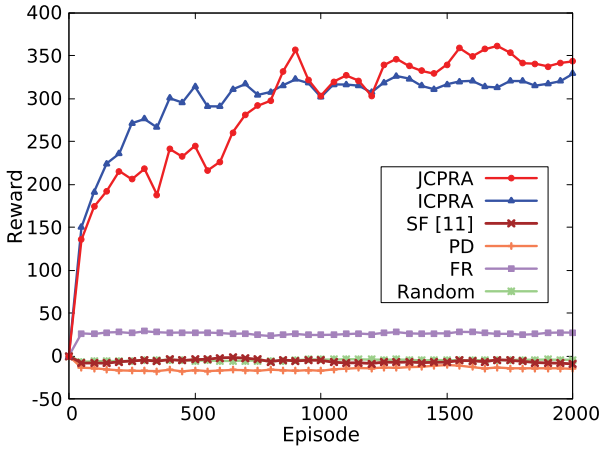


FIGURE 4. Comparison of training performance for different algorithms.

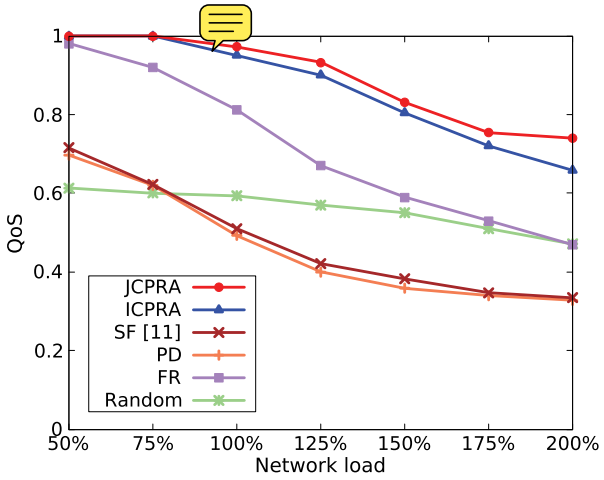


FIGURE 5. QoS satisfaction ratio under different levels of network load.

the core node's available CPU resource. The amount of core resources naturally determined the amount of resources in the lower layer. To guarantee QoS when network resources are limited, it is necessary to balance resource allocation among the slices. The proposed algorithms perform better than other algorithms, and PD, a comparable dynamic allocation method, shows the worst performance. This is due to the fact that the proposed algorithms reserve enough resources for upcoming demands, whereas the PD algorithm allocates the resources based on the previous request. SF and PD have almost the same performance. This is because SF is a greedy method that makes a decision using current information about the data rate and delays in a greedy manner, similarly to PD under the dynamic traffic environments. Under low network loads, FR performs well because of its ample resource occupancy, but its performance drops as the network load increases. We can see that the proposed JCPRA and ICPRA balance resources across slices for future resource demands under all network load conditions. The most remarkable aspect is that JCPRA achieves better performance under higher network loads than ICPRA, suggesting that JCPRA achieves a better end-to-end resource balance, as discussed in Section V-F.

D. RESOURCE EFFICIENCY UNDER DIFFERENT LEVELS OF NETWORK LOAD

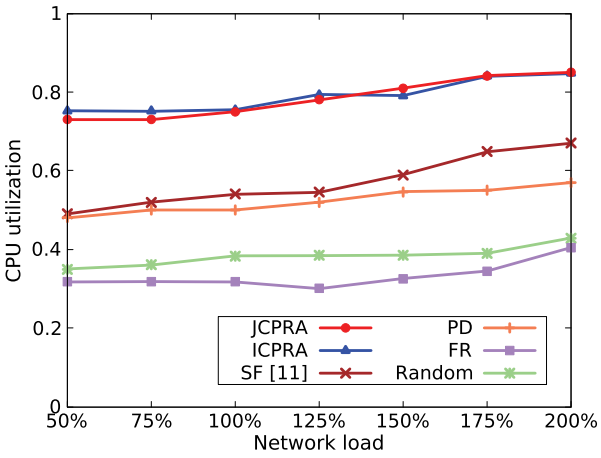
Figure 6(a) and 6(b) show resource utilization according to the network load. The utilization of all algorithms shows a trend of increase as the network load increases. These are the results of the increasing amount of consumed resources, despite allocating available resources as much as possible as the network load increases. As can be observed, the proposed algorithm achieves high utilization while guaranteeing QoS by allocating an appropriate amount of resources compared with PD. PD performs poorly because of its inability to prepare for sudden decreases in resource demand, despite allocating 1.2 times as much amount as resource demand. SF is slightly better than PD under overload conditions. The utilization of FR, which is a static allocation method, is low because it allocates resources excessively even when network demand is low. The random algorithm has similar utilization (∼30%–40%) under all network load conditions.

E. TRADE-OFF BETWEEN QoS AND UTILIZATION WITH DIFFERENT REWARD WEIGHTS

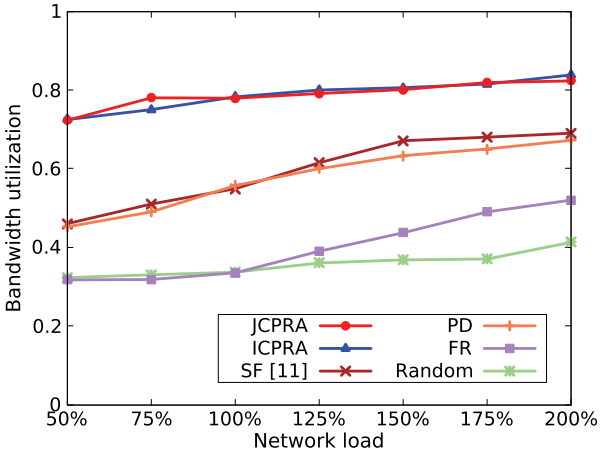
We varied the weight factor η of reward to observe the trade-off between QoS and resource utilization. In Figure 7, the result shows that the algorithm learns to guarantee QoS instead of increasing resource efficiency strictly when increasing the weight. QoS increases significantly from 88% to 99% as the weight increases from 1.0 to 1.6. On the other hand, resource utilization values decrease from approximately 81% to 74%. In particular, as the weight increases, the effects of the weight on QoS and utilization gradually decrease. This result shows that it is meaningful to choose an appropriate reward weight to balance QoS and utilization.

F. RESOURCE ALLOCATION BY LAYER ACCORDING TO SERVICE

Finally, we investigated the effect of different service types on the proportion of resource allocation to layers in the end-to-end network slicing scenario. Figure 8 illustrates the proportion of the CPU resource allocation of each service type according to the learning episode when ICPRA is applied. We considered three types of services: a delay-sensitive service (type-1), a large-bandwidth service (type-2), and a computation-intensive service (type-3). The results show that the CPU resource for the type-1 service is allocated at the largest proportion from edge nodes, which are closest to mobile devices. Meanwhile, the type-2 service, which requires relatively high bandwidth, is assigned more to the middle point of the hierarchy. The rational reason that the closer it is allocated to the core, the higher the bandwidth utilization of the whole network, resulting in performance degradation. It can be seen that the type-3 service with relatively large CPU requirements is most allocated to the core. Figure 9 shows the proportion of CPU resource allocation when JCPRA supporting inter-slice cooperation is applied. Converged results with JCPRA show that the layer dependence of the different services is very high compared



(a) CPU utilization



(b) Bandwidth utilization

FIGURE 6. Resource utilization under different levels of network load.

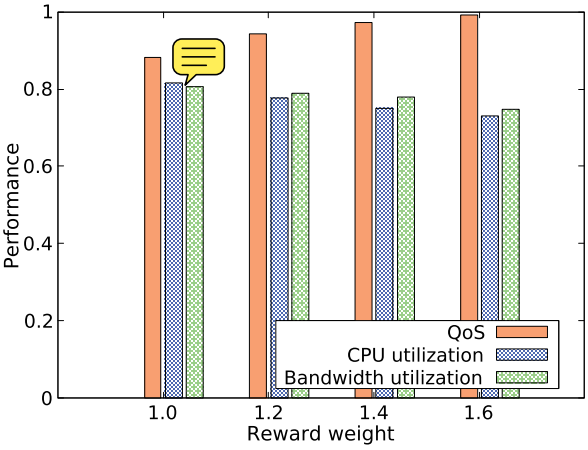
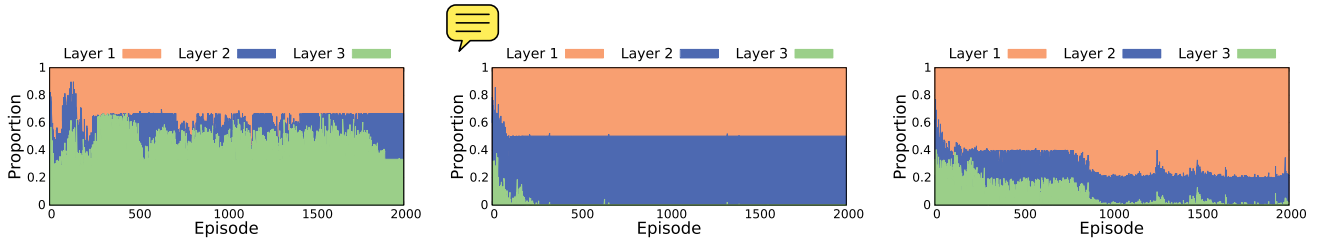


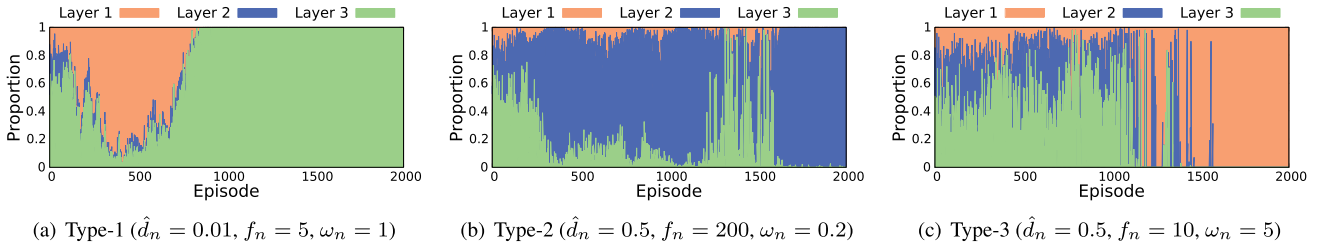
FIGURE 7. QoS and resource utilization under different reward weights.

with ICPRA. This means that resources are effectively allocated to different layers that fit the characteristics of the service as the episode progresses. The results show that the JCPRA algorithm works well to achieve the best balance for resource allocation to layers according to the different service types in end-to-end network slicing scenarios.



(a) Type-1 ($\hat{d}_n = 0.01, f_n = 5, \omega_n = 1$) (b) Type-2 ($\hat{d}_n = 0.5, f_n = 200, \omega_n = 0.2$) (c) Type-3 ($\hat{d}_n = 0.5, f_n = 10, \omega_n = 5$)

FIGURE 8. The proportion of resources allocated by the ICPRA algorithm according to the service type.



(a) Type-1 ($\hat{d}_n = 0.01, f_n = 5, \omega_n = 1$) (b) Type-2 ($\hat{d}_n = 0.5, f_n = 200, \omega_n = 0.2$) (c) Type-3 ($\hat{d}_n = 0.5, f_n = 10, \omega_n = 5$)

FIGURE 9. The proportion of resource allocated by the JCPRA algorithm according to the service type.

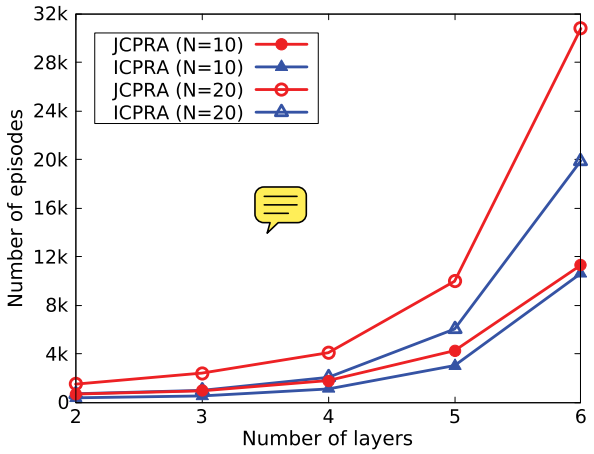


FIGURE 10. Number of episodes for convergence.

G. SCALABILITY

RL methods suffer from scalability issues because of a large number of states [29]. We evaluate the scalability of our algorithms depending on the different number of layers in the network topology shown in Figure 3. We have investigated the convergence speed to reach a goal (i.e., QoS ≥ 0.9). The network load is set to 100%. Figure 10 shows the number of episodes according to the number of layers. The scenarios with a larger number of layers require more training episodes to reach high QoS. The results indicate that our algorithms can achieve the goal of QoS satisfaction in more complex network topologies, even though the convergence speed decreases as the number of layers increases.

We evaluated the performance when the number of slices is greater than 10. While only three slice types are defined in 3GPP (i.e., eMBB, URLLC, MMTC), the use of more slice types has been proposed in a recent study [30]. Figure 10 shows the number of episodes for QoS satisfaction for $N = 20$. It is seen that more training episodes are needed for convergence as the number of slices increases. Especially, the convergence speed difference between $N = 10$ and 20 for JCPRA is larger than that for ICPRA. This is because the number of global states of its critic network significantly increases as the number of agents increases. If N and L are extremely large, we can consider to adopt large-scale RL schemes such as imitation learning [31], hierarchical learning [32] and parallel learning methods [33].

VI. CONCLUSION AND DISCUSSION

We have investigated the dynamic resource allocation of end-to-end network slicing with heterogeneous requirements in MEC environments to increase resource efficiency and guarantee QoS. We formulated the problem as a cooperative multi-agent task. Moreover, we developed the independent and joint learning algorithms, i.e., ICPRA and JCPRA, using policy gradient-based PPO algorithm to solve this problem. Simulation results demonstrated that the proposed algorithms achieve better performance than the existing methods, and JCPRA achieves a balance of resource allocation between the different types of services in a hierarchical MEC environment.

Our proposed work can contribute to network slicing management in various networks including 5G-and-beyond networks and industrial networks. The complexity of the networking systems increased by many devices, applications, and machines pose challenges in managing the network resources. In these networks, network slicing will play a more important role to supply resources on-demand and provide a more dynamic network that can adapt to the needs of emerging applications. To support the increasing heterogeneous services and complex networks, network slicing is an essential element of future networks. Without appropriate coordination among slices, it is difficult to achieve optimal resource allocation among slices in terms of management cost

and time. Our work will be a practical solution for future network slicing that enables automated and efficient resource management.

REFERENCES

[1] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 94–100, May 2017.
[2] Y. Kim, S. Kim, and H. Lim, "Reinforcement learning based resource management for network slicing," *Appl. Sci.*, vol. 9, no. 11, p. 2361, Jun. 2019.
[3] Z. Kotulski, T. W. Nowak, M. Sepczuk, M. Tunia, R. Artych, K. Bocianiak, T. Osko, and J.-P. Wary, "Towards constructive approach to end-to-end slice isolation in 5G networks," *EURASIP J. Inf. Secur.*, vol. 2018, no. 1, pp. 1–23, Dec. 2018.
[4] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.
[5] L. Cominardi, T. Deiss, M. Filippou, V. Sciancalepore, F. Giust, and D. Sabella, "MEC support for network slicing: Status and limitations from a standardization viewpoint," *IEEE Commun. Standards Mag.*, vol. 4, no. 2, pp. 22–30, Jun. 2020.
[6] N. Van Huynh, D. Thai Hoang, D. N. Nguyen, and E. Dutkiewicz, "Optimal and fast real-time resource slicing with deep dueling neural networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1455–1470, Jun. 2019.
[7] H. Wang, Y. Wu, G. Min, J. Xu, and P. Tang, "Data-driven dynamic resource scheduling for network slicing: A deep reinforcement learning approach," *Inf. Sci.*, vol. 498, pp. 106–116, Sep. 2019.
[8] A. Ksentini and P. A. Frangoudis, "Toward slicing-enabled multi-access edge computing in 5G," *IEEE Netw.*, vol. 34, no. 2, pp. 99–105, Mar. 2020.
[9] S. D'Oro, L. Bonati, F. Restuccia, M. Polese, M. Zorzi, and T. Melodia, "SI-EDGE: Network slicing at the edge," 2020, *arXiv:2005.00886*. [Online]. Available: http://arxiv.org/abs/2005.00886
[10] Y. Sun, M. Peng, S. Mao, and S. Yan, "Hierarchical radio resource allocation for network slicing in fog radio access networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3866–3881, Apr. 2019.
[11] B. Xiang, J. Elias, F. Martignon, and E. Di Nitto, "Joint network slicing and mobile edge computing in 5G networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.
[12] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin, K. W. Wen, K. Kim, R. Arora, A. Odjers, L. M. Contreras, and S. Scarpina, "MEC in 5G networks," Sophia Antipolis, France, ETSI White Paper 28, Jun. 2018, pp. 1–28.
[13] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.
[14] H. Liu, F. Eldarrat, H. Alqahtani, A. Reznik, X. de Foy, and Y. Zhang, "Mobile edge cloud system: Architectures, challenges, and approaches," *IEEE Syst. J.*, vol. 12, no. 3, pp. 2495–2508, Sep. 2018.
[15] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.
[16] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 2017.
[17] J. Moura and D. Hutchison, "Game theory for multi-access edge computing: Survey, use cases, and future trends," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 260–288, 1st Quart., 2019.
[18] M. Aloqaily, H. Bany Salameh, I. Al Ridhawi, K. Batieha, and J. B. Othman, "A multi-stage resource-constrained spectrum access mechanism for cognitive radio IoT networks: Time-spectrum block utilization," *Future Gener. Comput. Syst.*, vol. 110, pp. 254–266, Sep. 2020.
[19] Y. Jararweh, S. Otoum, and I. A. Ridhawi, "Trustworthy and sustainable smart city services at the edge," *Sustain. Cities Soc.*, vol. 62, Nov. 2020, Art. no. 102394.
[20] *Multi-Access Edge Computing (MEC): Support for Network Slicing*, ETSI MEC ISG, ETSI GR MEC 024 V2.1.1, Nov. 2019.
[21] V. Balasubramanian, F. Zaman, M. Aloqaily, I. A. Ridhawi, Y. Jararweh, and H. B. Salameh, "A mobility management architecture for seamless delivery of 5G-IoT services," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.
[22] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3133–3174, 4th Quart., 2019.
[23] V. Balasubramanian, F. Zaman, M. Aloqaily, S. Alrabaee, M. Gorlatova, and M. Reisslein, "Reinforcing the edge: Autonomous energy management for mobile device clouds," in *Proc. IEEE INFOCOM-IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 44–49.
[24] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA," *IEEE Access*, vol. 8, pp. 54074–54084, 2020.
[25] M. Nduwayezu, Q.-V. Pham, and W.-J. Hwang, "Online computation offloading in NOMA-based multi-access edge computing: A deep reinforcement learning approach," *IEEE Access*, vol. 8, pp. 99098–99109, 2020.
[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: http://arxiv.org/abs/1707.06347
[27] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv:1506.02438*. [Online]. Available: http://arxiv.org/abs/1506.02438
[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980
[29] Z. Ding and H. Dong, "Challenges of reinforcement learning," in *Deep Reinforcement Learning*. Singapore: Springer, 2020, pp. 249–272.
[30] ATIS, *IOT Categorization: Exploring the need for Standardizing Additional Network Slices*, document ATIS-I-0000075, 2019.
[31] W. Sun, A. Vemula, B. Boots, and D. Bagnell, "Provably efficient imitation learning from observation alone," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6036–6045.
[32] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," 2018, *arXiv:1805.08296*. [Online]. Available: http://arxiv.org/abs/1805.08296
[33] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "QT-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," 2018, *arXiv:1806.10293*. [Online]. Available: http://arxiv.org/abs/1806.10293

YOHAN KIM (Student Member, IEEE) received the B.S. degree in information and communication engineering from Chungbuk National University, Cheongju, South Korea, in 2015. He is currently pursuing the integrated M.S./Ph.D. degrees with the School of Electrical Engineering and Computer Science (EECS), Gwangju Institute of Science and Technology (GIST), Gwangju, South Korea. His research interests include resource management for 5G network slicing and artificial intelligence.

HYUK LIM (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the School of Electrical Engineering and Computer Science, Seoul National University, Seoul, South Korea, in 1996, 1998, and 2003, respectively. From 2003 to 2006, he was a Postdoctoral Research Associate with the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL, USA. He is currently a Full Professor with the AI Graduate School and jointly with the School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology (GIST), Gwangju, Republic of Korea. His research interests include wired and wireless networks, cyber-security, and artificial intelligence.