






QoS-Aware Cooperative Computation Offloading for Robot Swarms in Cloud Robotics

Zicong Hong , Huawei Huang , *Member, IEEE*, Song Guo , *Senior Member, IEEE*,
Wuhui Chen , *Member, IEEE*, and Zibin Zheng , *Senior Member, IEEE*

Abstract—Computation offloading is a promising solution to extend the capacity of robot swarms for computation-intensive applications because it allows robot swarms to benefit from the powerful computing resources of modern data centers. However, the existing computation-offloading approaches still face challenges: 1) multi-hop cooperative computation offloading, 2) joint computation offloading and routing, and 3) task slicing. In this paper, we propose a quality of service (QoS)-aware cooperative computation-offloading scheme for robot swarms using game theory. We analyze the multi-hop cooperative communication model in robot swarms and investigate the computation offloading and routing decision-making problems with the goals of both latency minimization and energy efficiency. We formulate the joint optimization problem as a multi-hop cooperative computation-offloading game and show the existence of a Nash equilibrium (NE) of the game for both unsliceable and sliceable tasks. We further propose a QoS-aware distributed algorithm to attain an NE and provide an upper bound on the price of anarchy in the game. Finally, our simulated results show that our algorithm scales well as the swarm size increases and it has a stable performance gain in various parameter settings.

Index Terms—Computation offloading, QoS, robot swarms, cloud robotics, game theory.

I. INTRODUCTION

TO SATISFY the needs of some sophisticated missions that one single powerful robot cannot accomplish alone, robot swarms, e.g., unmanned aerial vehicle (UAV) swarms, have attracted increasing research attention in recent years [1]–[4]. For example, companies such as Facebook and Google are planning to integrate UAV swarms as low-altitude platforms into a cellular network to provide broadband connectivity and compensate for cell overload or site outages. As another example, a robot swarm was used to remove debris from the interior and exterior

areas of the nuclear reactor buildings, and to monitor radiation levels at the Fukushima Daiichi nuclear power plant [5]. However, because of the limited on-board computational resources in robot swarms, it is still a challenging task for such swarms to conduct computation-intensive applications, which usually generate large amounts of data and demand huge processing power. Performing such applications with robot swarms would cause poor quality of service (QoS), such as high latency and high energy consumption, which are critical in these applications. Therefore, it is important to study computation offloading, which allows robot swarms to benefit from the abundant computing power of cloud data centers.

Many efforts have been made in computation offloading and can be classified into three main categories. 1) Computation offloading for computational tasks (e.g., simultaneous localization and mapping (SLAM), and object recognition). Garca [6] demonstrated the viability of cloud-based computation offloading for a vision-based navigation assistance task. 2) Centralized QoS-aware computation offloading. Barbarossa *et al.* [7] proposed a centralized computation-offloading strategy to minimize the energy expenditure under a delay constraint. 3) Distributed QoS-aware computation offloading in a single-layer structure. Chen *et al.* [8] proposed a distributed single-layer computation-offloading scheme based on game theory. However, the above three categories of existing methods do not achieve QoS-aware computation offloading for robot swarms because of the following challenges.

1) *Multi-Hop Cooperative Computation Offloading*: This approach has been considered only by few scholars in previous studies. Most existing offloading approaches [7]–[11] assume that all edge devices can connect to cloud data centers directly via wired or wireless networks, called here *single-hop computation offloading*. However, a mobile robot may experience poor or even intermittent connectivity and may be unable to connect to a cloud access point directly (e.g., robots in a tunnel or in an areas of damage after a disaster). Robots must then collaborate with each other to forward transmitted messages to the remote cloud via neighboring robots for computation offloading, called here *multi-hop computation offloading*. A robot node must consider not only its own tasks but also relay tasks from neighboring robots, which makes the computation-offloading problem more complex. Hence, the multi-hop computation-offloading problem should be investigated carefully.

2) *Joint Computation Offloading and Routing*: Furthermore, because the above existing studies [7]–[11] only consider

Manuscript received September 24, 2018; revised December 28, 2018; accepted February 14, 2019. Date of publication February 26, 2019; date of current version April 16, 2019. This work was supported in part by the National Key Research and Development Plan under Grant 2018YFB1003803, in part by the National Natural Science Foundation of China under Grant 61802450 and Grant 61722214, in part by the Natural Science Foundation of Guangdong under Grant 2018A030313005, and in part the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant 2017ZT07X355. The review of this paper was coordinated by Prof. J.-M. Chung. (*Corresponding author: Wuhui Chen.*)

Z. Hong, H. Huang, W. Chen, and Z. Zheng are with the School of Data and Computer Science and the National Engineering Research Center of Digital Life, Sun Yat-Sen University, Guangzhou 510006, China (e-mail: hongzc@mail2.sysu.edu.cn; hwhuang@media.kyoto-u.ac.jp; chenwuh,huanghw28; zhzhbin@mail.sysu.edu.cn).

S. Guo is with the Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong (e-mail: song.guo@polyu.edu.hk).

Digital Object Identifier 10.1109/TVT.2019.2901761

single-hop communication, routing strategies for data transfer for cost minimization have been ignored. However, because of limited communication resources in robot swarms and large data volumes from multiple sources (e.g., sensors or cameras on robots), the data transfer cost (e.g., communication latency or energy consumption) may become a bottleneck of the whole cloud robotic system, and thus efficient routing strategies are required to reduce the data transfer cost.

3) *Task Slicing*: Most of existing studies [8]–[11] that apply game theory to computation offloading only consider unsliceable tasks; i.e., each task can only be executed as a whole by a computing node in the cloud or locally. However, computation tasks can often be sliced into several subtasks using mechanisms like the framework proposed in [12], which allows more flexible system design and achieves maximum speed in data processing. In this paper, we consider both unsliceable and sliceable tasks for efficient task processing.

To deal with the first challenge, we formulate the multi-hop cooperative communication model (MCCM) as a multi-hop cooperative computation-offloading game (MCCG) and show the existence of a Nash equilibrium (NE). For the joint computation offloading and routing issue, we propose a QoS-aware distributed algorithm that can reach an NE. Finally, for the task-slicing issue, we extend the base model using continuous potential theory. To the best of our knowledge, this is the first attempt to study the QoS-aware distributed computation-offloading problem considering multi-hop cooperative offloading and joint computation offloading and routing.

Our main contributions can be summarized as follows.

- 1) We first study the multi-hop cooperative computation offloading and routing problems jointly for QoS improvement. Aiming for latency minimization and energy efficiency, we then formulate the MCCM as an MCCG using game theory.
- 2) We propose a locked-free mechanism and a QoS-aware distributed algorithm for MCCG. Based on the locked-free mechanism, potential game theory and congestion game theory, we solve the MCCG problem and show the existence of an NE. Furthermore, we propose a multi-hop cooperative messaging mechanism for our algorithm, so that each robot can gain enough information to obtain its real-time cost function and make decisions.
- 3) We consider both sliceable and unsliceable tasks for computation offloading. We first extend our model to address the offloading problem for sliceable tasks. We then use the continuous potential game theory to analyze the extended model and prove that the sliceable task can be reduced to an unsliceable task in MCCM.
- 4) We also describe extensive simulations to evaluate our algorithm. The numerical results demonstrate that our algorithm scales well as the swarm size increases and is more efficient than existing algorithms. We also show that our algorithm achieves stable performance gain for various parameter settings.

The remainder of this paper is structured as follows. In Section II, we discuss related research. In Section III, we introduce our MCCM. In Section V, we design the MCCG bases on our system model. In Section VI, we propose a QoS-aware

distributed algorithm. In Section VII, we evaluate our proposed solution, and Section VIII concludes the paper.

II. RELATED WORK

The computation-offloading problem has been widely studied for cloud and edge computing [13], [14]. Kehoe *et al.* [15] studied the computation-offloading problem in the cloud to facilitate computing of optimal 3D robot grasping. He *et al.* [16] proposed a novel cloud-based computation framework for real-time multi-robot collision avoidance. Mohanarajah *et al.* [17] presented some techniques for parallelizing the computationally expensive operations of map optimization and map merging in a commercial data center for collaborative 3D mapping. However, these solutions do not consider QoS optimization for computation offloading.

The QoS-aware computation-offloading problem has also been studied for cloud robotics systems [18], [19]. Rahman *et al.* [20] proposed a genetic algorithm-based computation-offloading scheme considering QoS guarantees. Li *et al.* [21] proposed a centralized architecture to extend the capability of robots by leveraging the rich services provided in the cloud without sacrificing QoS. Ko *et al.* [22] proposed a spatial and temporal offloading algorithm considering transmission cost and energy efficiency. However, these centralized offloading schemes require all the UAVs to submit their personal information (e.g., resource and network status, data size, and the complexity of tasks) to a centralized computing node (e.g., the remote cloud) for offloading decisions, which will increase the controlling and signaling overhead of the computing node.

To our knowledge, only a few studies have addressed the QoS-aware distributed computation-offloading problem. Those most related to our work are [8]–[11], because they all use game theory to solve the computation-offloading problem. Chen *et al.* [8] constructed a single-channel wireless model in which each mobile device user has a binary decision variable: offload computation or compute locally. Based on that work, Chen *et al.* [9] proposed a multi-channel wireless model and presented a distributed algorithm to compute an NE. Ma *et al.* [10] considered a multi-channel, fair bandwidth sharing, and nonelastic cloud model, and prove that the game has an exact potential. Josilo *et al.* [11] also considered fair bandwidth and a nonelastic cloud. But in her model, the computation capability of the cloud is a nonincreasing function of the number of members offloading, which is closer to real situations.

In this paper, we also combine potential game theory and congestion game theory to solve our problem. However, we extend the range of application to multi-hop computation offloading, where there are multiple layers in our model instead of the single layer in the traditional model presented in [8]–[11]. The most important contribution of our work is that we further extend the feasible decision range for robot swarms so that game theory can be applied to more computation-offloading scenarios such as multi-hop communication.

III. SYSTEM MODEL

In MCCM, we consider that a set of robots $\mathcal{P} = \{1, 2, \dots, N\}$ equipped with different wireless communication and

TABLE I
NOTATION DEFINITIONS

Symbol	Definition
\mathcal{P}	Set of N robots
\mathcal{B}	Set of b base stations
M_i	Computation task of robot i
S_i	Input data size for M_i
L_i	Total number of CPU cycles for M_i
\mathcal{P}_l	Set of robots with l hops to \mathbf{B}
F_i	Computing capacity of robot i
$F_{cloud}^{comp}(\mathbf{d})$	Computing capacity of the cloud in \mathbf{d}
$p(i)$	Parent of robot i
\mathbf{r}_i	Offloading path of robot i
\mathbf{D}_i	Set of all possible decisions for robot i
\mathbf{d}_i	Decision of robot i
\mathbf{D}	Set of all possible decision profile for \mathcal{P}
\mathbf{d}	Decision profile of \mathcal{P}
$\mathbf{M}_i^{off}(\mathbf{d})$	Set of tasks in the charge of player i
(i, j)	Communication link between i and j
$f(\mathbf{r}_i)$	A function used to separate \mathbf{r}_i into a set of links
μ_i	Energy consumed per CPU cycle for robot i
$R_{(i,j)}$	Capacity of communication link between i and j
$n_{(i,j)}^{off}(\mathbf{d})$	Number of tasks transferred between (i, j) in \mathbf{d}
$\omega_{(i,j)}(\mathbf{d})$	Channel capacity for each task between (i, j)
T_i^0	Time cost of local computing for M_i
E_i^0	Energy cost of local computing for M_i
$T_{(i,j)}^{off}(\mathbf{d}, M_k)$	Time cost of transferring between (i, j) for M_k in \mathbf{d}
$E_{(i,j)}^{off}(\mathbf{d}, M_k)$	Energy cost of transferring between (i, j) for M_k in \mathbf{d}
$T_{cloud}^{comp}(\mathbf{d}, M_i)$	Time cost of computing in the cloud for M_k in \mathbf{d}
C_i^0	Cost of task M_i computed locally
$C_i^{off}(\mathbf{d})$	Cost of task M_i computed offload in \mathbf{d}
$C_i^{relay}(\mathbf{d})$	Cost of robot i as a relay in \mathbf{d}
$C_i(\mathbf{d})$	Total cost of robot i in \mathbf{d}
$C(\mathbf{d})$	Total cost of the system in \mathbf{d}

onboard computation capabilities and a set of base stations (BSs) $\mathcal{B} = \{1, 2, \dots, b\}$ are distributed in the space. Both robot-to-base station (R2B) communications and robot-to-robot (R2R) communications are based on LTE Direct technology [23]. In order to facilitate the communication and computation resource sharing among heterogeneous robots, the Software-Defined Networking (SDN) based control management frameworks [24]–[26] can be adopted in practice.

Similar to other works in computation offloading (e.g. [8]–[11]), MCCM is quasi-static, which means each robot moves slowly or keeps unchanged during a computation offloading period. Each robot $i \in \mathcal{P}$ has a computation task $M_i = (S_i, L_i)$ to complete, where S_i is the size of computation input data and L_i is the total number of CPU cycles required to accomplish the task. If a robot decides to offload its computation task to the cloud, the data of its task needs to be transferred to one of the BSs. As shown in Fig. 1, the task is computed locally if the color of its name is black, such as M_4, M_6, M_7 . If the color of the task's name is not black, it means the task is offloaded to the cloud through the path with the same color, such as M_1, M_2, M_3 and M_5 .

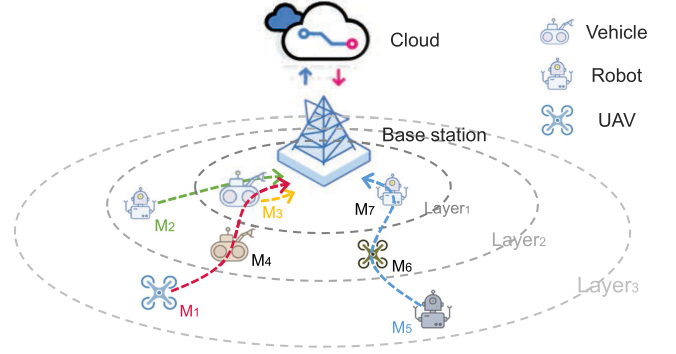
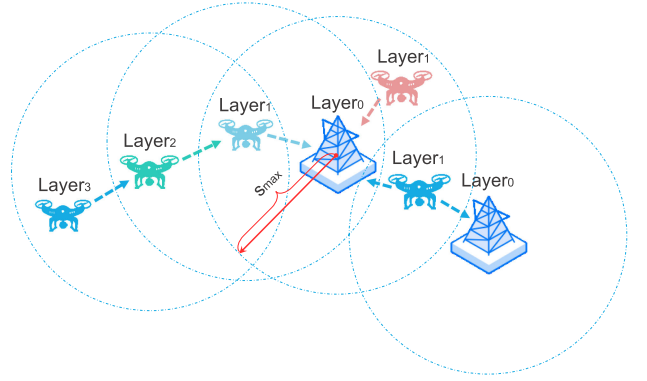


Fig. 1. System model for robot swarm.

Fig. 2. Members in different layers according to s_{max} .

However, in reality, some robots may be located far away from the BSs or be blocked by obstacles, so they can not offload their tasks to the cloud directly. In order to give them a chance to offload and to improve the overall performance of the robots, MCCM allows them to offload their tasks to the BSs via some other robots. For example in Fig. 1, the robot located in Layer2 can offload its task via one robot to the BS and the robot located in Layer3 offloads its task via two robots.

In this paper, we use the term *member* to refer to communication-capable members such as robots or BSs, and we denote by s_{max} the maximum range over which a member can transfer data to other members. For example, in Fig. 2, a member (robot or BS) is located at the center of a circle and its communication range covers the area of the circle with radius s_{max} and any other members within that circle can communicate directly with the member at the center. In contrast, two members in different circles cannot communicate with each other directly; they need the help of one or more relays for communication.

According to the value of s_{max} and the distance between each pair of robots, we can classify every robot into different layers; by definition, only members in adjacent layers can communicate directly. We assume all BSs belong to layer 0, which is denoted by $\mathcal{B} = \mathcal{P}_0$. Then, the robots that can communicate with BSs belong to layer 1, and so forth for the remainder of the swarm. The most remote layer in the swarm is layer L . Thus, we can deduce that the robots in set \mathcal{P}_l that have the same hop count $h_i = l \in [1, L]$ to the BS lie in the same layer l , where $\mathcal{P}_l := \{i \in \mathcal{P} | h_i = l\}$.

We define the following three roles for members in a robot swarm for a clear exposition in later sections:

- 1) *Relay*: The members that help others transmit data indirectly or directly are collectively called relays.
- 2) *Parent*: The relay i that helps another member j transmit directly is the parent of j .
- 3) *Child*: The member i that transmits its data to another member j directly is the child of j .

Each robot i in \mathcal{P}_l will choose one and only one member in the \mathcal{P}_{l-1} as its parent $p(i)$. If robot i decides to offload a task, it will transmit both its own task and relay tasks to its parent. Otherwise, if it decides to compute locally, it will only transmit its relay tasks to the parent. Thus, each robot i has a path $\mathbf{r}_i = \{i, p(i), p(p(i)), \dots, b\}$, where $b \in \mathcal{B}$ and $p(b) = \text{cloud}$. In addition, we define the notation $k(i)$, which refers to all children of i , where $\forall j \in k(i), p(j) = i$.

IV. PROBLEM STATEMENT AND FORMULATIONS

Based on the system model introduced above, we propose a decision-based description of the multi-hop computation-offloading and routing problem, with joint consideration of latency minimization and energy efficiency.

A. Decision-Based Description

We first define \mathbf{D}_i to be the decision set (transferable members) for i , where robot i 's decision $\mathbf{d}_i \in \mathbf{D}_i$ is a tuple (a, \mathbf{r}_i) with $a \in \{0, 1\}$. Accordingly, $\mathbf{d}_i = (0, \mathbf{r}_i)$ means computing locally, $\mathbf{d}_i = (1, \mathbf{r}_i)$ means using the path \mathbf{r}_i to offload. We use an operator $[\]$ to index the decision; thus, $\mathbf{d}_i[0] = a \in \{0, 1\}$ and $\mathbf{d}_i[1] = \mathbf{r}_i$.

Then, we refer to the collection $\mathbf{d} = \{\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_N\}$ as a *decision profile* consisting of all robots' decisions. The set of all possible decision profiles is $\mathbf{D} = \mathbf{D}_1 \times \mathbf{D}_2 \times \dots \times \mathbf{D}_N$, thus we get $\mathbf{d} \in \mathbf{D}$.

Using the above notation, the set of tasks which are in the charge of player i denotes:

$$\mathbf{M}_i^{\text{off}}(\mathbf{d}) = \{M_j | \mathbf{d}_j = (1, \mathbf{r}_j) \text{ and } i \in \mathbf{r}_j\}, \quad (1)$$

and the number of tasks in the charge of player i is $n_i^{\text{off}}(\mathbf{d}) = |\mathbf{M}_i^{\text{off}}(\mathbf{d})|$ includes all tasks that require the help of i .

We denote $\mathbf{d}_{-i} = \mathbf{d} \setminus \mathbf{d}_i = \{\mathbf{d}_1, \dots, \mathbf{d}_{i-1}, \mathbf{d}_{i+1}, \dots, \mathbf{d}_N\}$ which consists of all decisions in the decision profile \mathbf{d} except the i 's decision.

If one relay robot i decides to change its parent from $p(i)$ to $p'(i)$ to gain some benefit, its path will change to $\mathbf{r}'_i = \{i, p'(i), p(p'(i)), \dots, b\}$ and the path \mathbf{r}_j for robot j , which includes i , i.e., $i \in \mathbf{r}_j$ will also change to $\mathbf{r}'_j = \{j, \dots, i, p'(i), p(p'(i)), \dots, b\}$. Note that although i decides to change its parent, robot $p'(i)$ is not affected, which means that the parent of $p'(i)$ remains $p'(p'(i)) = p(p'(i))$.

Based on the fact that path \mathbf{r}_i is composed of several communication links between members, we propose a function $f(\mathbf{r}_i)$, which can be used to transfer path \mathbf{r}_i into a different expression which is about a set of communication links. For example, if $\mathbf{r}_i = \{i, p(i), p(p(i)), \dots, b\}$, $f(\mathbf{r}_i) = \{(i, p(i)), (p(i), p(p(i))), \dots, (p^{-1}(b), b)\}$, where each communication link is denoted by a tuple in parentheses.

B. Local Computing Cost

For local computing, robots perform the calculation task locally using its own CPU or GPU. We set F_i as robot i 's computing capacity (i.e., CPU cycles per second) and we assume that different robots may have different computing capabilities. Thus, the time cost of task M_i is $T_i^0 = \frac{L_i}{F_i}$ and the energy cost of task M_i is $E_i^0 = \mu_i L_i$, where μ_i denotes the energy consumed per CPU cycle for robot i . The cost when task M_i is computed locally by robot i can, therefore, be expressed as:

$$C_i^0 = \gamma_i^T T_i^0 + \gamma_i^E E_i^0 = \gamma_i^T \frac{L_i}{F_i} + \gamma_i^E \mu_i L_i, \quad (2)$$

where γ_i^E and γ_i^T denote the weights of computation energy and time, respectively, for robot i 's decision-making. Based on the related studies [8], [9], and [11], we propose a novel method of weight calculation for more accurate description of such weights, which takes the difference between the units of energy and time into account. We set $\gamma_i^T = \delta_i^T / T_i^0$ and $\gamma_i^E = \delta_i^E / E_i^0$, where $0 \leq \delta_i^T$ and $\delta_i^E \leq 1$ denote the importance of computational energy and time, respectively. If a robot has limited battery capacity, we can set $\delta_i^E > \delta_i^T$, thus making energy cost more important. If a robot is doing some latency-sensitive tasks, we can set $\delta_i^E < \delta_i^T$ to make latency play a predominant role.

C. Computation-Offloading Cost

When computation is offloaded, the robot must first transfer its data to the cloud. If the robot is located in a distant layer, it requires some support from intermediate robots. Thus, we define the capacity in bits/second $R_{(i,j)}$ of the communication link (i, j) between sender i and receiver j and the bandwidth is divided equally for each relay task in robot i . Given the number of relay tasks transferred between i and j in decision profile \mathbf{d} as $n_{(i,j)}^{\text{off}}(\mathbf{d})$, we can compute the channel capacity for each task from i to j in decision profile \mathbf{d} as:

$$\omega_{(i,j)}(\mathbf{d}) = \frac{R_{(i,j)}}{n_{(i,j)}^{\text{off}}(\mathbf{d})}. \quad (3)$$

Therefore, using the channel capacity for each task $\omega_{(i,j)}(\mathbf{d})$ and the size S_k of computation task M_k , we can compute the transfer time for task M_k from i to j as

$$T_{(i,j)}^{\text{off}}(\mathbf{d}, M_k) = \frac{S_k}{\omega_{(i,j)}(\mathbf{d})}. \quad (4)$$

Similar to other studies [9], [10], we do not consider the time required to transmit the computing results because the results data are usually much smaller than the origin data.

In the related studies [8], [11], the transmission energy cost is defined as the product of transfer time and the transmit power for each robot. Using the same calculation, in our model, the energy cost for transferring M_k from i to j is depicted as:

$$E_{(i,j)}^{\text{off}}(\mathbf{d}, M_k) = W_i T_{(i,j)}^{\text{off}}(\mathbf{d}, M_k) = \frac{S_k W_i}{\omega_{(i,j)}(\mathbf{d})}, \quad (5)$$

where W_i denotes the transmit power of i while sending data.

The computing time for task M_i in the cloud is:

$$T_{\text{cloud}}^{\text{comp}}(\mathbf{d}, M_i) = \frac{L_i}{F_{\text{cloud}}^{\text{comp}}(\mathbf{d})}, \quad (6)$$

where $F_{cloud}^{comp}(\mathbf{d})$ is a function of the number of robots which decide to offload computation in decision profile \mathbf{d} . In our model, we assume that the cloud is an elastic cloud, which means that the cloud infrastructure is large and the cloud computing resources are sufficient. Then, $F_{cloud}^{comp}(\mathbf{d}) = F_{cloud}$, where F_{cloud} is a constant.

Using Eq. (4), the total time for transferring M_i from player i to the cloud is:

$$T_{(i,cloud)}^{off}(\mathbf{d}, M_i) = \sum_{j \in f(\mathbf{r}_i)} T_j^{off}(\mathbf{d}, M_i) = S_i \sum_{j \in f(\mathbf{r}_i)} \frac{n_j^{off}(\mathbf{d})}{R_j}. \quad (7)$$

Using Eq. (5), the transfer energy cost for M_i from player i to its parent is:

$$E_{(i,p(i))}^{off}(\mathbf{d}, M_i) = \frac{S_i n_{(i,p(i))}^{off}(\mathbf{d}) W_i}{R_{(i,p(i))}}. \quad (8)$$

According to Eq. (6), Eq. (7), and Eq. (8), we can compute the computation offload cost for player i with task M_i as:

$$C_i^{off}(\mathbf{d}) = \gamma_i^E E_{(i,p(i))}^{off}(\mathbf{d}, M_i) + \gamma_i^T (T_{cloud}^{comp}(\mathbf{d}, M_i) + T_{(i,cloud)}^{off}(\mathbf{d}, M_i)), \quad (9)$$

where $\gamma_i^T = \delta_i^T / T_i^0$, $\gamma_i^E = \delta_i^E / E_i^0$, and $0 \leq \delta_i^T$ and $\delta_i^E \leq 1$ likewise. We can see that the equation does not include the energy cost for the intermediates, only consisting of the cost for i . Because we assume that every robot is selfish and has benefit-oriented mechanisms, they will only consider their own interests and will not care about other robots.

D. Relay Cost

The robots may need to help other robots offload as a relay. In this paper, the social responsibility of the relay is to transmit the relay task to its parent and the return data of the task are not considered. Thus, robots only consider the energy cost of the relay tasks but do not care how long the relay tasks take to arrive at the destination. Thus, the cost of i as a relay is:

$$C_i^{relay}(\mathbf{d}) = \gamma_i^E \sum_{M_k \in \mathbf{M}_i^{off}(\mathbf{d}) \setminus M_i} E_{(i,p(i))}^{off}(\mathbf{d}, M_k). \quad (10)$$

E. Total Cost

In order to define robot i 's cost in the decision profile \mathbf{d} , we set an indicator function for robot i :

$$I(\mathbf{d}_i, j) = \begin{cases} 1, & \text{if } \mathbf{d}_i[0] = j \\ 0, & \text{otherwise,} \end{cases} \quad (11)$$

where j is a binary integer.

The cost for robot i of task M_i in the decision profile \mathbf{d} is:

$$C_i(\mathbf{d}) = C_i^{off}(\mathbf{d}) I(\mathbf{d}_i, 1) + C_i^0 I(\mathbf{d}_i, 0) + C_i^{relay}(\mathbf{d}). \quad (12)$$

Finally, Eq. (12) gives us the total cost in the system:

$$C(\mathbf{d}) = \sum_{i \in \mathcal{P}} C_i(\mathbf{d}). \quad (13)$$

V. GAME FORMULATION FOR MULTI-HOP COOPERATIVE COMPUTATION-OFFLOADING

When solving the proposed problem, the centralized algorithms (such as the heuristic algorithm presented in [18]) face two challenges.

- 1) Because of the complexity of the model, the problem of finding the best decision to minimize the total cost can be extremely hard.
- 2) Due to commercial competitions between different counterparties, the production information with respect to hardware parameters of their robots are possibly confidential. Thus, the information isolation makes a centralized algorithm that operates over multiple robot platforms impossible in practice.

Therefore, we attempt to solve our problem using game theory, which is a useful tool for decision-making. For our problem, it can leverage the intelligence and computing capacity of each robot, which can ease the heavy burden of computing and management at the center. It also allows the privacy of information and individual interests of each robot to be maintained while also guaranteeing collective interests.

A. Definition of the Game

We consider that each robot is rational and the goal of each robot is to adapt its decision to minimize its cost $C_i(\mathbf{d})$, i.e., $\mathbf{d}_i^* \in \argmin_{\mathbf{d}_i \in \mathbf{D}_i} C_i(\mathbf{d}_i, \mathbf{d}_{-i})$. Then, we can formulate the above problem as a non-cooperative game $G = \langle \mathcal{P}, (\mathbf{D}_i)_{i \in \mathcal{P}}, (C_i)_{i \in \mathcal{P}} \rangle$, where \mathcal{P} is the set of robot players, \mathbf{D}_i is the decision space of player i , and $C_i : \mathbf{D}_i \rightarrow R$ is the payoff function associated with player i . We define the game as a *multi-hop cooperative computation-offloading game* (MCCG) and our goal is to find an NE in which each player cannot decrease its cost by unilaterally changing its own decision. We define NE as follows:

Definition 1: An NE of $G = \langle \mathcal{P}, (\mathbf{D}_i)_{i \in \mathcal{P}}, (C_i)_{i \in \mathcal{P}} \rangle$ is a decision profile \mathbf{d}^* with the property that for $\forall i \in \mathcal{P}$:

$$C_i(\mathbf{d}_i^*, \mathbf{d}_{-i}^*) \leq C_i(\mathbf{d}_i, \mathbf{d}_{-i}^*), \forall \mathbf{d}_i \in \mathbf{D}_i. \quad (14)$$

The NE can be interpreted as a self-stability property by which all robots can achieve a mutually satisfactory state in which no robot wants to change its decision. However, if one robot is willing to make a sacrifice, others may be able to make better decisions. Otherwise, if no one changes its own decision first, there will be no better decision for every player; thus, no one has an incentive to deviate.

B. Definition of the Better Decision

In MCCG, every rational player (or robot) has an incentive to choose a decision that benefits itself, so we next define the “better decision” for players in different states, which is helpful for us to construct the following proofs.

Definition 2: If a player i chooses another player $p'(i)$ as its parent and $C_i(\mathbf{d}_{-i}, (1, \mathbf{r}'_i)) < C_i(\mathbf{d}_{-i}, (1, \mathbf{r}_i))$, then using the path $\mathbf{r}'_i = \{i, p'(i), \dots, b'\}$ is a better decision than using the path $\mathbf{r}_i = \{i, p(i), \dots, b\}$.

Definition 3: For a player i , if $\mathbf{d}'_i = (0, \mathbf{r}_i)$ and $\mathbf{d}_i = (1, \mathbf{r}_i)$ and $C_i(\mathbf{d}_{-i}, \mathbf{d}'_i) < C_i(\mathbf{d}_{-i}, \mathbf{d}_i)$, where \mathbf{r}'_i and \mathbf{r}_i can be the same

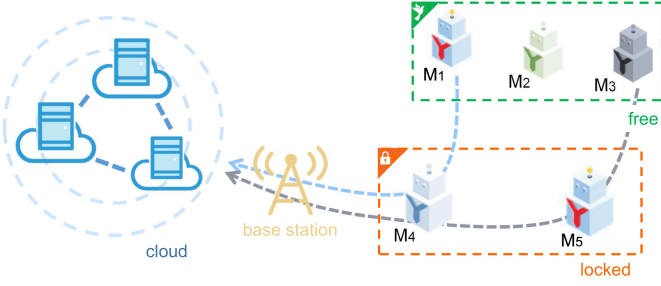


Fig. 3. The status of each robot is either locked or free.

Algorithm 1: ImproveRobot Algorithm.

Input: Original decision profile \mathbf{d}

Output: NE decision profile \mathbf{d}^*

```

1 while  $\exists i \in \mathcal{P}, |M_i^{off}(\mathbf{d}) \setminus M_i| = 0,$ 
    $\exists \mathbf{d}'_i \in \mathcal{D}_i, C_i(\mathbf{d}'_i, \mathbf{d}_{-i}) < C_i(\mathbf{d}_i, \mathbf{d}_{-i})$  do
2    $\mathbf{d}_i = \mathbf{d}'_i$ 
3  $\mathbf{d}^* = \mathbf{d}$ 

```

or different, then local computing is a better decision than offload computing.

Definition 4: For a player i , if $\mathbf{d}'_i = (1, \mathbf{r}_i)$ and $\mathbf{d}_i = (0, \mathbf{r}_i)$ and $C_i(\mathbf{d}_{-i}, \mathbf{d}'_i) < C_i(\mathbf{d}_{-i}, \mathbf{d}_i)$, where \mathbf{r}'_i and \mathbf{r}_i can be the same or different, then computing offload through path \mathbf{r}'_i is a better decision than computing locally.

C. Locked-Free Mechanism

To ensure an NE in MCCG, we then introduce a control mechanism, which we call the *locked-free mechanism*. We will show that MCCG can only attain an NE using our algorithm with the mechanism in Section VII.

The locked-free mechanism means that a relay robot in charge of other robots' tasks is forbidden to change its decision (e.g., from computation offloading to local computation, or from one parent to another parent). Only robots with no relay tasks can change their decisions freely. For example in Fig. 3, robot M_1 decides to offload and robot M_4 is the parent of robot M_1 , thus we say M_4 is *locked*. Because robot M_1 doesn't have any relay tasks, robot M_1 is *free*. In the same way, we get robot M_5 is locked and robot M_2 and M_3 is free.

D. Unslicable Tasks

In this section, to analyze the core of our problem, we first ignore any communication delays in the robot swarms and assume all robots know the global congestion information in the system, which will be discussed again in Section VI.

Based on the locked-free mechanism in Section V-C and the finite-improvement property (FIP) for potential game proposed in [27], we propose a theoretic algorithm in Algorithm 1 and a theorem about it as follows:

Theorem 1: For an unslicable task, given an arbitrary state in MCCG, the free robots can reach to an NE through a finite number of steps by using the ImproveRobot algorithm.

Proof: See Appendix A. ■

Algorithm 2: QoS-aware Distributed Algorithm.

Input: Decision profile in this time slot $\mathbf{d}(t)$

Output: Decision profile in next time slot $\mathbf{d}(t+1)$

```

1 initialization:
2 the cloud send BM to  $\mathcal{P}_1$ 
3 for each  $i \in \mathcal{P}$  do
4   if receive BM then
5     transmit BM to  $k(i)$  and collect CM
6 stage 1:
7 for each  $i \in \mathcal{P}$  do
8   wait until receive all CMs from  $j \in \mathcal{P} \setminus i$  within the
     range of  $s_{max}$ 
9   pack and send CM to  $j \in \mathcal{P} \setminus i$  within the range of
      $s_{max}$ 
10 stage 2:
11 for each free  $i \in \mathcal{P}$  do
12   if  $\exists \mathbf{d}'_i \in \mathcal{D}_i, C_i(\mathbf{d}'_i, \mathbf{d}_{-i}(t)) < C_i(\mathbf{d}_i(t))$  then
13     send RM to the cloud using the path  $\mathbf{r}_i(t)$ 
14   else
15     send KM to the cloud using the path  $\mathbf{r}_i(t)$ 
16 stage 3:
17 if the cloud receive RM from  $i$  then
18   the cloud send AM to  $i$ 
19 else
20   the game reach to NE
21 stage 4:
22 for each  $i \in \mathcal{P}$  do
23   if receive AM then
24      $\mathbf{d}_i(t+1) = \mathbf{d}'_i$  inform  $j \in \mathbf{r}_i(t+1) \cup \mathbf{r}_i(t)$ 
25   else
26      $\mathbf{d}_i(t+1) = \mathbf{d}_i(t)$ 
27 for each  $i \in \mathcal{P}$  do
28   check if  $|M_i^{off}(\mathbf{d}) \setminus M_i| = 0$  lock or free itself

```

E. Sliceable Tasks

We now extend the range of decision by allowing all robots to partition their tasks into two arbitrary parts. For example, a robot might choose to compute 40% of its task in the cloud and 60% locally. We believe that this extension can benefit our multi-hop computation-offloading system and achieve maximum speed in processing the data.

We refer to the weight α_i as the proposition of task M_i to offload computing and weight $(1 - \alpha_i)$ as the proposition of the task to compute locally, where $\forall i \in \mathcal{P}, \alpha_i \in [0, 1]$. For a better description of player i 's decision, we use a new notation: $\mathbf{y}_i = (\alpha_i, \mathbf{r}_i) \in \mathbf{Y}_i$, where \mathbf{r}_i is the path for robot i and \mathbf{Y}_i is the decision set for player i . We can then redefine the cost of player i in this case as:

$$U_i(\mathbf{y}) = \alpha_i C_i^{off}(\mathbf{d}) + (1 - \alpha_i) C_i^0 + U_i^{relay}(\mathbf{y}), \quad (15)$$

where $\mathbf{d}_i[1] = \mathbf{y}_i[1] = \mathbf{r}_i$, which is consistent when each \mathbf{d} and \mathbf{y} and $U_i(\mathbf{y})$ is continuously differentiable.

In contrast to the relay cost for an inseparable task, here we adjust $U_i^{relay}(\mathbf{y})$ as:

$$U_i^{relay}(\mathbf{y}) = \sum_{M_k \in \mathbf{M}_i^{off} \setminus M_i} \gamma_i^E \alpha_k E_{(i,p(i))}^{off}(\mathbf{d}, M_k). \quad (16)$$

Similarly, we obtain the total cost to the system as $U(\mathbf{d}) = \sum_{i \in \mathcal{P}} U_i(\mathbf{d})$ and thus construct the MCCG in a new way as $H = \langle \mathcal{P}, (\mathbf{Y}_i)_{i \in \mathcal{P}}, (U_i)_{i \in \mathcal{P}} \rangle$. As with the goal for unsliceable tasks, our objective is to find a decision profile \mathbf{y}^* in NE that satisfies the following condition:

$$U_i(\mathbf{y}_i^*, \mathbf{y}_{-i}^*) \leq U_i(\mathbf{y}_i, \mathbf{y}_{-i}^*), \forall \mathbf{y}_i \in \mathbf{Y}_i, \forall i \in \mathcal{P}. \quad (17)$$

To attain the NE in H , we propose the following theorem:

Theorem 2: For an sliceable task, given an arbitrary state in MCCG, the free robots can reach to an NE through a finite number of steps by using the ImproveRobot algorithm.

Proof: See Appendix B. ■

VI. QOS-AWARE DISTRIBUTED ALGORITHM

In this section, to enable each robot to gather enough information to obtain its own real-time cost function for the ImproveRobot algorithm (Algorithm 1), we propose a QoS-aware distributed algorithm to reach an NE in a realistic multi-hop cooperative communication system.

A. Distributed Algorithm Design

In the ImproveRobot algorithm, we assume that every robot knows the real-time congestion of the whole system, so every robot can predict the future condition and change its decision according to the latest communication flow information. However, in realistic systems, it will be very difficult to realize.

The traditional method lets all robots transmit their condition information to the cloud and the cloud can aggregate all this information into a global database and send it to every robot, but this will also be very expensive.

Therefore, we construct a *multi-hop cooperative messaging mechanism* and we first introduce some message types for communication between the cloud and the robots.

- **Begin Message (BM):** Initially, the cloud will send a Begin Message that includes the condition of the cloud and instructs each robot to collect Conditional Messages.
- **Conditional Message (CM):** The message includes information about the uplink rates to the parent and the number of relay tasks for a robot.
- **Keep Message (KM):** If a free robot does not have a better decision or it is locked, it will generate a KM to tell the cloud it does not want to change.
- **Request Message (RM):** If a free robot has a better decision, it will generate an RM to request an update from the cloud.
- **Allow Message (AM):** After receiving the RM from a robot, the cloud will send an AM to that robot to update the current decision of the robot.

All communications between any pair of robots are based on the above five kinds of messages, so the information in these messages should be shared among different organizations. Using only these five kinds of messages can substantially protect the

privacy of each robot because the hardware parameters of robots are isolated with each other.

Then, we can construct a communication mechanism to solve the above problem as follows. The cloud first sends a BM to begin the game. The BM is transferred to the robots in \mathcal{P}_1 and they will, in turn, transfer the BM to \mathcal{P}_2 and so forth until it is delivered to every robot in the swarm and they will all know that they are players in MCCG. Some robots' positions may be too distant to communicate with anyone, so they will not be included in the game. After receiving the BM, each robot will begin to collect a CM about itself; for example, its current decision and uplink rates to transferring robots. This massive information collection is distributed to all the robots so it can both reduce the burden of the center and protect each robot's privacy. In some scenarios, the robots can be owned by different individuals and they may cooperate to finish some tasks, but they do not want to leak their respective private information.

Next, at the same time as BM is sent, the center will also send a CM containing the computation capacity of the cloud to the robots in the same way as BM is sent. Every robot will thus know the computation capacity of the cloud. Before each robot transfers the cloud's CM to the robots in its next layer, it will also pack its own CM and all CMs it received to the next layer within the range of s_{max} .

After receiving CMs from all robots within the range of s_{max} , a robot can compute its set of feasible decision profiles and then choose a better decision from the set. If the set of better decisions remains empty, the current decision is the best option, so it keeps its decision unchanged and sends KM to the cloud. However, if there exists a better decision, it will have an incentive to choose a new decision for its own benefit and send RM to the cloud.

From Algorithm 1, we know that if in every step we let only one free robot improve its decision, the MCCG can reach an NE within a finite number of improvement steps. To utilize this, we set the cloud as a judge that can select one robot to change its decision. There are many possible selection approaches. In our mechanism, the cloud selects the first RM it receives. If the cloud doesn't receive any RM and the number of KMs received equals the total number of robots, we know the system has reached an NE and the game is finished.

If a free robot improves its decision and stops offloading computing or changes the offload to a new path, it will inform the robots in the original path and the new path. Each locked robot in these paths will check whether it still has any relay tasks. Those without any relay tasks will change their state from locked to free.

B. Performance Analysis

We introduce a metric named "price of anarchy" (PoA), which is used to measure the damage suffered by a robot swarm or the inefficiency of the equilibrium caused by the absence of a central authority. It is defined as the ratio between the worst social cost of an NE in our game and of the optimal result [28]. In this paper, we consider the social cost as the sum of all costs $C(\mathbf{d}) = \sum_{i \in \mathcal{P}} C_i(\mathbf{d})$. Hence for our game, the PoA is equal to the social cost of the worst decision profile \mathbf{d}^* in NE over the

optimum social cost:

$$PoA = \frac{\max_{\mathbf{d}^*} \sum_{i \in \mathcal{P}} C_i(\mathbf{d}^*)}{\min_{\mathbf{d} \in \mathcal{D}} \sum_{i \in \mathcal{P}} C_i(\mathbf{d})} \quad (18)$$

Theorem 3: The PoA for the multi-hop cooperative computation-offloading game has the upper bound:

$$\frac{\sum_{i \in \mathcal{P}_L} C_i^0 + \sum_{i \in \mathcal{P} \setminus \mathcal{P}_L} V_i}{\sum_{i \in \mathcal{P}} K_i}$$

which can be calculated by three different constants for each UAV $i \in \mathcal{P}$: $V_i = \max\{C_i^0, \max_{\mathbf{d} \in \mathcal{D}}\{C_i^{off}(\mathbf{d})\}\} + \max_{\mathbf{d} \in \mathcal{D}} C_i^{relay}(\mathbf{d})$, $K_i = \min\{C_i^0, \min_{\mathbf{d} \in \mathcal{D}}\{C_i^{off}(\mathbf{d})\}\}$ and C_i^0 is the local computing cost.

Proof: See Appendix C. ■

VII. EVALUATION

A. Simulation Design

Evaluation Metrics: In this section, all algorithms will be evaluated based on the following two metrics:

- 1) Similar to the approach presented in [29], we use *performance gain* to evaluate the overall performance of the system. The performance gain of an algorithm is calculated as the ratio between the system cost of local computing of all tasks and the system cost of the algorithm. The bigger performance gain indicates a higher overall performance of the player set.
- 2) Similar to the approach presented in [9], we also consider how many robots become more efficient when adopting the proposed algorithm. Thus, in this paper, we compute a *profit ratio*, which is the proportion of robots that gain profit. The bigger profit ratio means more players experience improvement.

Comparing Approaches: From an optimization perspective, the recent related work solving the distributed computation offloading problem can be generally classified into three categories: 1) latency-minimization algorithms, 2) energy-efficient algorithms, and 3) joint latency- and energy-aware algorithms. Therefore, the following four algorithms are used as compared algorithms:

- 1) *Latency-Dominant Algorithm:* The latency-dominant distributed algorithm is developed by modifying the weight parameters of our QoS-aware distributed algorithm, i.e. $\delta_i^T = 1$ and $\delta_i^E = 0$. Based on this setting, the energy cost is ignored and only the latency is minimized.
- 2) *Energy-Dominant Algorithm:* The energy-dominant distributed algorithm is proposed by modifying the weight parameters of our QoS-aware distributed algorithm, i.e. $\delta_i^T = 0$ and $\delta_i^E = 1$. Based on this setting, the latency is ignored and only the energy cost is minimized.
- 3) *Single-Hop Offloading Algorithm:* A distributed single-hop computation offloading algorithm [9] is used, which considers both latency and energy optimization. In this algorithm, the robots located far away from the BSs or blocked by obstacles can't offload their tasks to the BSs. Only the robots located around the BSs are allowed to perform task offloading.
- 4) *Greedy Algorithm:* In addition, we also implement a centralized algorithm which picks the best solution from 500

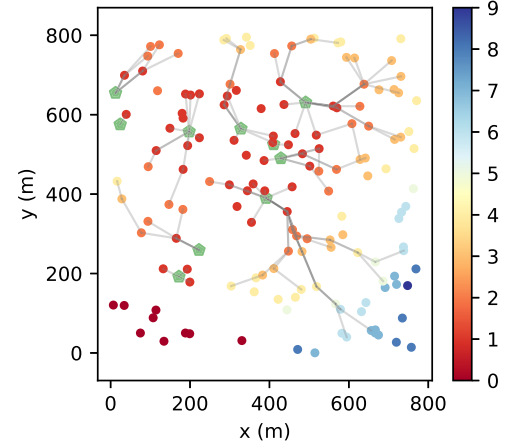


Fig. 4. Simulation map for UAVs and BSs.

iterations using randomized decision profiles to compare with the proposed algorithm.

B. Simulation Settings

Following the system model described in Section III, we first study some UAV swarms that are randomly located over an $800 \text{ m} \times 800 \text{ m}$ plane, which is much bigger than the range of mobile users modeled in [8], [9], and [11]. For wireless access, we set the bandwidth as $B = 10 \text{ MHz}$ and the transmitting power of each UAV i , P_i , as 5 W . We assume that $\sigma^2 = -120 \text{ dBm}$ denotes the noise power and $\beta_0 = -50 \text{ dB}$ denotes the channel power at unit distance $d_{(i,j)} = 1 \text{ m}$, as in [30] and [31]. According to the expression in [31], we find that the capacity of the communication channel between sender i and receiver j is $R_{(i,j)} = \log_2(1 + \lambda_i^0/d_{(i,j)}^2)$ where $\lambda_i^0 = \beta_0 P_i / \sigma^2$ denotes the reference of i 's received signal-to-noise ratio at $d_{(i,j)} = 1 \text{ m}$ and $d_{(i,j)}$ denotes the distance between i and j .

For the communication between members, we set the maximum communication range between them as $s_{max} = 60 \text{ m}$. For the computation task, we consider the visual track application of UAV swarms such as [32] where the data size of each computation task is uniformly distributed on $[200, 250] \text{ K}$ and is about the size of a picture. The total number of CPU cycles for each task is uniformly distributed in $[1.5, 2] \text{ Gcycles}$. For the computation capacity, we selected the computation capacity F_i of UAV i from a continuous uniform distribution on $[1, 2] \text{ GHz}$ and the computation capacity of the cloud is set to 100 GHz , which is 100 times the capacity of a UAV, according to [33].

C. Simulation Methodology

Using the above data, we first simulated a scenario with 150 UAVs and 10 BSs and then let the system reach an NE as shown in Fig. 4. In the figure, the lines denote the communication paths between members. A darker line means more data are transferred through the path. The green pentagons identify the BSs and the circles refer to UAVs; the color of a circle depends on both its layer number and the color bar. For example, dark red UAVs are located in layer 1 and dark blue UAVs are located in layer 8. From the distribution and darkness of all lines, we can see intuitively that the lines close to a BS are darker and

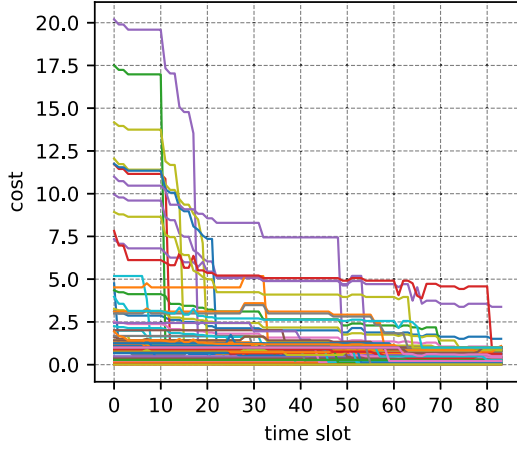


Fig. 5. Time-varying cost of each UAV yielded by our distributed algorithm.

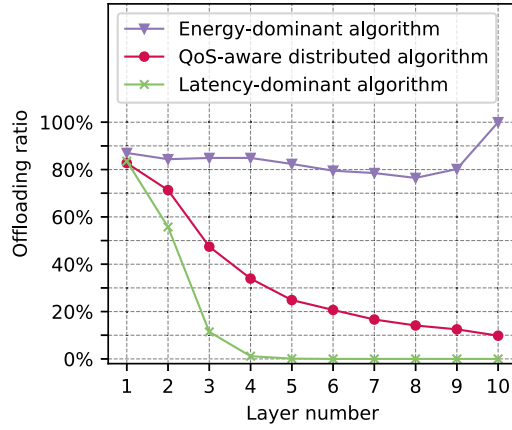


Fig. 6. Offloading ratio vs. layer number.

denser. In summary, we assume that the UAVs close to a BS are more likely to offload their tasks because it costs less to transmit their data to the cloud.

Next, to confirm our assumption, we show the offloading ratio in different layers for three algorithms in Fig. 6. We can see that the offloading ratio for the energy-dominant distributed algorithm is nearly 90% for all layers. This is because, in this algorithm, each UAV only considers the energy used to transmit data to its parent, so the offloading ratio will not differ greatly in the different layers. In addition, we can see the offloading ratio of the outmost layer is extremely high, because there are few UAVs in the outmost layer and it will cause large error. However, the latency-dominant distributed algorithm considers the total time from the UAV to the cloud, so the close UAVs are more likely to offload computation and the distant UAVs are more likely to compute locally. Then, because the QoS-aware distributed algorithm considers both, the offloading ratio lies between the other two. In summary, we can confirm the above assumption from Fig. 6.

Theorem 1 states that only one robot can choose better decision in each time slot so that the ImproveRobot algorithm can finish in a finite number of steps. Fig. 5 illustrates the change in each UAV's cost over time. We can see that our algorithm allows only one UAV to choose a better decision in a time slot and all UAVs finally reach a stable state.

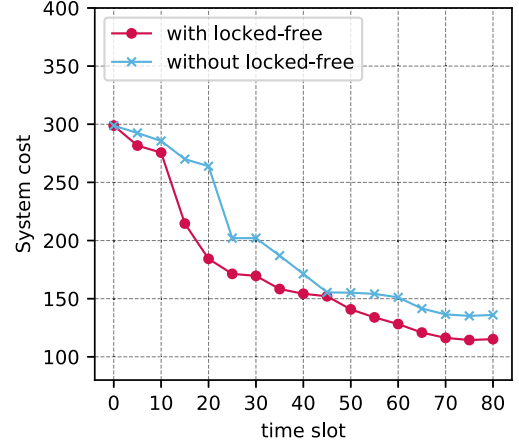


Fig. 7. Dynamics of system cost with locked-free mechanism and without locked-free mechanism.

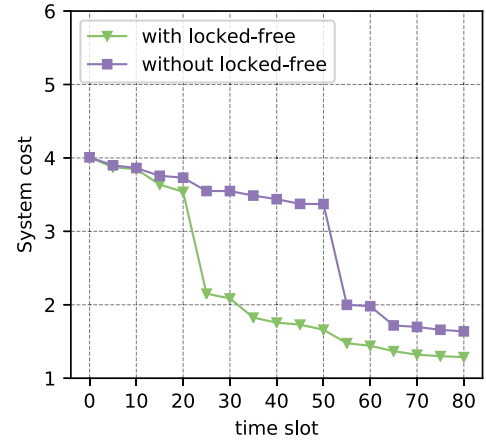


Fig. 8. Dynamics of potential function with locked-free mechanism and without locked-free mechanism.

In Fig. 7, we can see that the system cost with locked-free mechanism has a faster descent trend than it does without the mechanism. In Fig. 8, we can see that the locked-free mechanism also make the potential function be able to decrease faster in each step. Furthermore, from Theorem 1, we know that the locked-free mechanism can guarantee that MCCG converges to an equilibrium. Without the locked-free mechanism, MCCG will not converge fast and can't guarantee to reach an NE. In summary, the locked-free mechanism not only guarantees an NE in theory but also has a beneficial effect on our algorithm in practice.

D. Topology Varying Adaption

In the real scenarios, the topology in mobile environment usually is time-varying. Therefore, our algorithm needs to run continuously. In order to study the performance of our algorithm when the topology changes, we conduct a simulation to evaluate how our algorithm adapts to the varying topology.

In Fig. 9, we firstly let the robot swarms reach an NE by our QoS-aware distributed algorithm with the locked-free mechanism as we have done in the simulation shown in Fig. 7. After 80 time slots, the robots reach a stable state (denoted by the most left-hand-side green circle), i.e. NE. Then, each robot is allowed

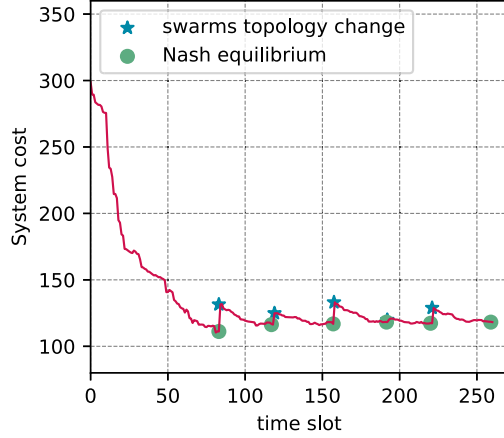


Fig. 9. QoS-aware distributed algorithm quickly adapts to multiple-time topology changing.

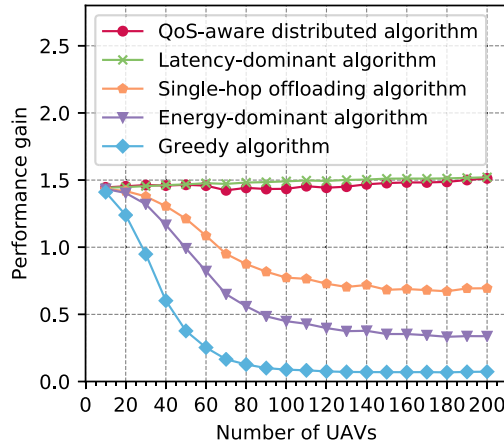


Fig. 10. Performance gain vs. number of UAVs.

to move freely in the speed 20 m/s heading any direction for 2 seconds [34]. When the topology of the robot swarms changes, the system cost increases (denoted by the most left-hand-side blue star) shown in Fig. 9. To our surprise, we observe that the system cost only increases to 130, which is much lower than the original system cost 300. Besides, the swarms can converge to a new NE again only using 30 time slots (denoted by the second green circle), which is much faster than the first time. Similarly, we observe other 4 times topology changing. Each time, the system cost can be controlled within a small range and our algorithm can converge quickly to a new NE.

In summary, our QoS-aware distributed algorithm is capable to quickly adapt to the topology changing in the practical mobile environment and keep the system cost in a relative stable state.

E. Performance Comparison

To benchmark the performance of our QoS-aware distributed algorithm, we compare it with the latency-dominant algorithm, the energy-dominant distributed algorithms, the greedy algorithm and the single-hop offloading algorithm. We ran our simulation with $N = 10, \dots, 190, 200$ UAVs and for each case, we performed each simulation 1000 times. We show the average performance gain by each algorithm in Fig. 10 and the average profit ratio by each algorithm in Fig. 11.

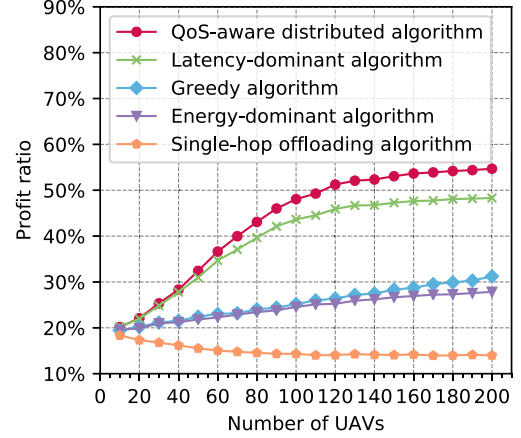


Fig. 11. Profit ratio vs. number of UAVs.

Fig. 10 shows that the performance gain of our algorithm is approximated by that of the latency-dominant algorithm but much better than the performance gain of any other algorithms. According to Eq. (7), Eq. (8) and Eq. (9), we can see that the latency cost takes the total transmission time of path $f(r_i)$ into account but the energy cost for i doesn't include the energy cost for the intermediate nodes in path $f(r_i)$. In other words, each robot will experience the latency cost for the whole path and the energy cost for only a small part of the whole path. Therefore, the latency will dominate the system cost, which makes our QoS-aware algorithm have similar performance gain with the latency-dominant algorithm. No matter how many UAVs there are, the performance gain of our algorithm is about 1.5, which means it performs 1.5 times better than the situation that all UAVs compute locally performs. All of the greedy algorithm, the single-hop offloading algorithm and the energy-dominant distributed algorithm perform worse as the number of UAVs increases. For the greedy algorithm, this is because there will be more possible situations with more UAVs, and the greedy algorithm will gain less during 500 random iterations. For the energy-dominant distributed algorithm, there will be more UAVs located in the distant layer when the number increases. For the single-hop offloading algorithm, if there are more UAVs existing in the system, more of them are possibly located far away from the BSs and unable to offload their tasks due to the constraint of the single-hop offloading.

Fig. 11 shows that our QoS-aware distributed algorithm outperforms any other algorithms in terms of profit ratio. This implies that our QoS-aware algorithm enables more players gain profit. To illustrate the importance of profit ratio, we can draw an analogy with realistic social phenomena. If two different policies can deliver the same social economic benefits, then the one that can allow more persons to gain benefit will be a better policy. In summary, although the performance gain in our algorithm and the latency-dominant algorithm is similar, the profit ratio of our algorithm is much bigger, which means our algorithm outperforms any other algorithms.

F. Computation Complexity

In addition to the above optimization results, the time required for each algorithm should also be considered. For our algorithm,

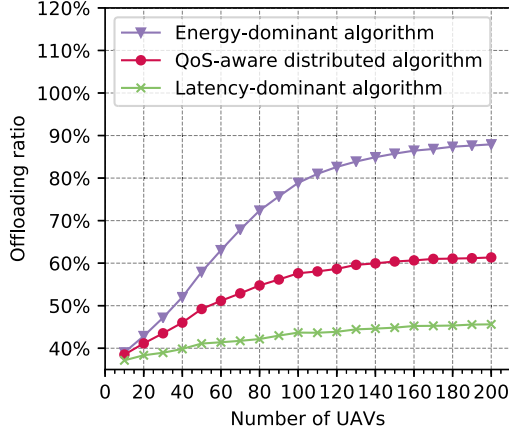


Fig. 12. Offloading ratio vs. number of UAVs.

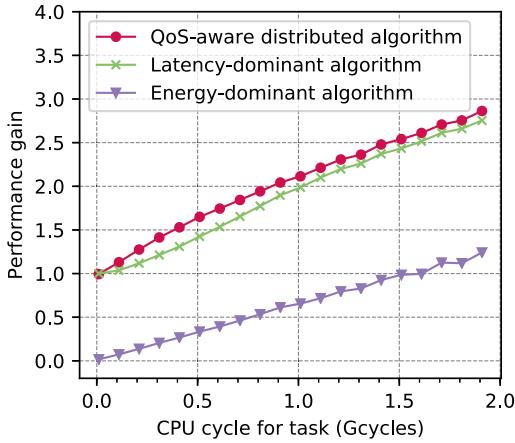


Fig. 13. Performance gain vs. CPU cycles of computation task.

we recorded the average number of iterations to NE with different number of UAVs and show them in Fig. 15. We can see that the average convergence time increases linearly with the number of UAVs from 10 to 200, where $RMSE_{(0)} = 6.398147$ and $R^2_{(0)} = 0.980456$, almost one. We observe that the relationship between time and number of UAVs is not linear with fewer UAVs. As the number increases, the relationship becomes more linear, where $RMSE_{(100)} < RMSE_{(40)} < RMSE_{(0)}$ and $R^2_{(100)} > R^2_{(40)} > R^2_{(0)}$.

We see that this is because the distances between members are large and the number of relays are limited when there are few UAVs, and it may be expensive for each locked UAV to offload its computing to the cloud. However, when the number of UAVs is large enough for the space, the communication distances become shorter and there are more relay UAVs, so the cost will reduce slowly as the number of UAVs increases. We can offer a definition: If $R^2_{(n)}$ is bigger than 0.999, we can assume that the number n of UAVs is enough to perform tasks in such a space. We now use the offloading ratio to validate the assumption in Section VII-G as follows.

G. Offloading Ratio

The offloading ratio equals to the number of offloading UAVs divided by the number of all UAVs. In Fig. 12, we can again validate the conclusion from Fig. 6 because the offloading ratio

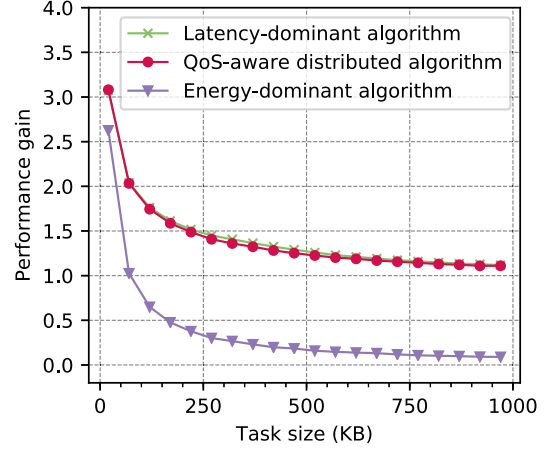


Fig. 14. Performance gain vs. data size of computation task.

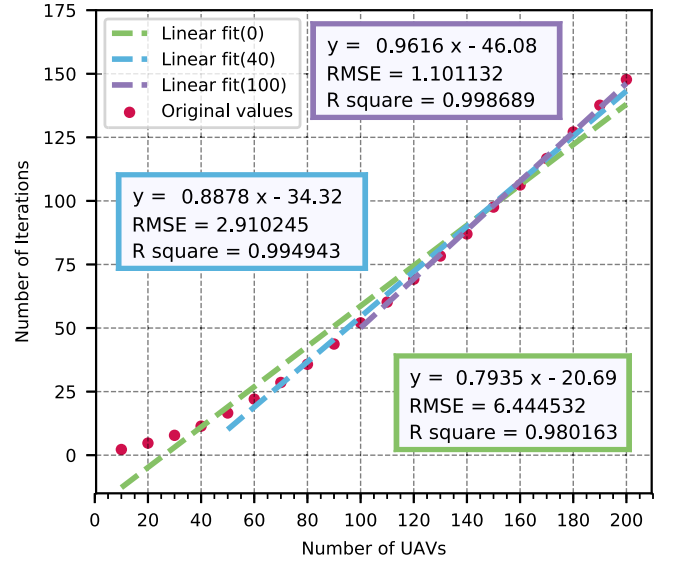


Fig. 15. Average number of iterations vs. number of UAVs.

for the energy-dominant distributed algorithm is higher than that for the other distributed algorithms. One cause is the selfishness of the UAVs.

We can also validate the summary in Section VII-F from 11 and Fig. 12. In both figures, we can see that the ratio for our distributed algorithm clearly increases as the number of UAVs ranges from 0 to 100. When the number is more than 100, the offloading and profit ratios begin to stabilize. This is also because at that time the number of UAVs is enough for the space and the impact of more UAVs will be small.

H. Impact of Computation Amount and Data Size

To study the impact of computation amount on MCCG, we compare the performance gain of different CPU cycles for the computation task in Fig. 13. We can see that our distributed algorithm works better than any other distributed algorithm. The performance gain of both our algorithm and the latency-dominant distributed algorithm are greater than 1 for any computation amount. When the CPU cycles for a task exceed 1.5 Gcycles, the energy-dominant algorithm can gain a better

result than local computing. This is because if the computation amount is huge, it will be expensive to compute locally because of the limited computation resources in each UAV. In summary, we can make more profit with our algorithm if the amount of computation per task increases.

Next, we evaluate the impact of the input data size of computation tasks. In Fig. 14, we assumed that all UAVs must offload tasks that require a computation amount of $[1.5, 2]$ Gcycles, and record the performance gain for different data sizes of tasks (0, 1000] KB by the different algorithms. We can see that the performance gains of these three algorithms decrease as the data size increases. For large data sizes, each offloading UAV may spend more time and energy in offloading its task to the cloud and the offloading UAV will gain less from the choice of offloading.

VIII. CONCLUSION

In this paper, we studied multi-hop computation offloading jointly with the routing problem and proposed a QoS-aware distributed algorithm using game theory for robot swarms in cloud robotics. We first studied the MCCM for robot swarms and formulated the QoS-aware computation-offloading and routing problem as a MCCG using game theory. To solve the problem, we then introduced a locked-free mechanism to constrain the relay robots and used potential game theory to prove the game has an NE with unsliceable tasks. We proved that sliceable tasks can be treated the same as unsliceable tasks in our model. We designed a QoS-aware distributed algorithm that can reach the NE and proved the convergence of the algorithm. Finally, our simulated results show the efficiency of our algorithm, which scales well as the swarm size increases and has more stable and better performance than other algorithms under a variety of parameter settings.

In future work, we plan to study our MCCM in a nonelastic cloud or edge-cloud scenario. We also hope to adopt an on-demand bandwidth-sharing model, which would allow the bandwidth of relay robots to be divided depending on the data size or urgency of the relay tasks.

APPENDIX A PROOF OF THEOREM 1

Before proving the theorem, we give the definition about the ω -potential game and its properties according to [27]:

Definition 5: Let $\omega = (\omega_i)_{i \in \mathcal{P}}$ be a vector of positive numbers as weights. A game with payoff function $(C_i)_{i \in \mathcal{P}}$ is called an ω -potential game if it admits an ω -potential that is a function $\Phi : \mathbf{D} \rightarrow R$ such that for every $i \in \mathcal{P}$, $\mathbf{d}_{-i} \in \mathbf{D}_{-i}$, it satisfies $C_i(\mathbf{d}_{-i}, \mathbf{d}'_i) - C_i(\mathbf{d}_{-i}, \mathbf{d}_i) = \omega_i(\Phi(\mathbf{d}_{-i}, \mathbf{d}'_i) - \Phi(\mathbf{d}_{-i}, \mathbf{d}_i))$ for every $\mathbf{d}_i, \mathbf{d}'_i \in \mathbf{D}_i$.

The ω -potential game has FIP property which can be explained in simple terms: if we let only one player choose a better decision in each time slot and then finally the system can achieve an NE after a finite number of steps. So if our game is an ω -potential game, we can get an NE by only allowing one and only one player to choose a better decision in each time slot as shown in ImproveRobot algorithm. After a finite number of steps, no free player can decrease its cost further by unilaterally changing its own decision, i.e., an NE.

First, to show that our game is an ω -potential game, we use an idea from a congestion model according to [27]. We first define the set of path segments \mathcal{L} , where each path segment $j \in \mathcal{L}$ can be expressed as the communication link between member $j[0] = a$ and member $j[1] = b$, i.e., $j = (a, b)$. Then, for $j \in \mathcal{L}$, $c_j \in R^N$ denotes the vector of congested levels for path segment j , where $c_j(k) = k/R_j$ is the congested level related to each user of segment j if there are exactly k players using j to transfer their tasks. Using the notation c_j , we can simplify the cost expression Eq. (12) for player i to:

$$\begin{aligned} C_i(\mathbf{d}) &= C_i^{off}(\mathbf{d})I(\mathbf{d}_i, 1) + C_i^0 I(\mathbf{d}_i, 0) + C_i^{relay}(\mathbf{d}) \\ &= (\gamma_i^T (T_{cloud}^{comp}(\mathbf{d}, M_i) + T_{(i, cloud)}^{off}(\mathbf{d}, M_i)) \\ &\quad + \gamma_i^E E_{(i, p(i))}^{off}(\mathbf{d}, M_i))I(\mathbf{d}_i, 1) + \left(\gamma_i^T \frac{L_i}{F_i} + \gamma_i^E \mu_i L_i \right) \\ &\quad \times I(\mathbf{d}_i, 0) + \sum_{M_k \in \mathbf{M}_i^{off}(\mathbf{d}) \setminus M_i} \gamma_i^E E_{(i, p(i))}^{off}(\mathbf{d}, M_k) \\ &= \gamma_i^T (T_{cloud}^{comp}(\mathbf{d}, M_i) + S_i \sum_{j \in f(\mathbf{r}_i)} c_j(n_j^{off}(\mathbf{d})))I(\mathbf{d}_i, 1) \\ &\quad + \left(\gamma_i^T \frac{L_i}{F_i} + \gamma_i^E \mu_i L_i \right) I(\mathbf{d}_i, 0) \\ &\quad + \gamma_i^E W_i c_{(i, p(i))}(n_{(i, p(i))}^{off}(\mathbf{d})) \sum_{M_k \in \mathbf{M}_i^{off}(\mathbf{d})} S_k. \end{aligned} \quad (19)$$

To map each payoff of the individual robot to a global function, we define a potential function $\Phi : \mathbf{D} \rightarrow R$ as:

$$\begin{aligned} \Phi(\mathbf{d}) &= \sum_{i \in \mathcal{L}} \sum_{j=0}^{n_i^{off}(\mathbf{d})} c_i(j) - \sum_{i \in \mathcal{P}} Q_i I(\mathbf{d}_i, 0) \\ &\quad + \sum_{i \in \mathcal{P}} \frac{\gamma_i^E W_i}{\gamma_i^T} c_{(i, p(i))}(1) I(\mathbf{d}_i, 1), \end{aligned} \quad (20)$$

where $Q_i = \frac{L_i}{S_i} (\frac{1}{F_c} - \frac{1}{F_i} - \frac{\gamma_i^E \mu_i}{\gamma_i^T})$. Then, we can prove that MCCOG is an ω -potential game by Eq. (19), Eq. (20), and Def. 5. The proof includes the three following cases, corresponding to the three kinds of better decisions defined in Sec. V-B.

Case 1: If player i chooses originally to compute locally but then finds that there is a better decision \mathbf{d}'_i to offload computing. We find the variation of the potential function:

$$\begin{aligned} \Delta \Phi &= \Phi(\mathbf{d}_{-i}, \mathbf{d}'_i) - \Phi(\mathbf{d}_{-i}, \mathbf{d}_i) \\ &= Q_i + \sum_{j \in f(\mathbf{r}_i)} c_j(n_j^{off}(\mathbf{d}) + 1) + \frac{\gamma_i^E W_i}{\gamma_i^T} c_{(i, p(i))}(1). \end{aligned} \quad (21)$$

The above variation of potential function can be interpreted as the scenario that only robot i changes its decision in the robot swarm while other robots don't change. The congestion level of path \mathbf{r}_i increases by one due to robot i 's offloading:

$$\begin{aligned} \Delta C_i &= C_i(\mathbf{d}_{-i}, \mathbf{d}'_i) - C_i(\mathbf{d}_{-i}, \mathbf{d}_i) \\ &= (\gamma_i^E W_i c_{(i, p(i))}(n_{(i, p(i))}^{off}(\mathbf{d}')) S_i + \gamma_i^T \frac{L_i}{F_c} \end{aligned}$$

$$\begin{aligned}
& + \gamma_i^T S_i \sum_{j \in f(\mathbf{r}_i)} c_j(n_j^{off}(\mathbf{d}')) - \left(\gamma_i^T \frac{L_i}{F_i} + \gamma_i^E \mu_i L_i \right) \\
& = \left(\gamma_i^T \frac{L_i}{F_c} - \gamma_i^T \frac{L_i}{F_i} - \gamma_i^E \mu_i L_i \right) + \gamma_i^T S_i \sum_{j \in f(\mathbf{r}_i)} c_j(n_j^{off}(\mathbf{d}')) \\
& + \gamma_i^E W_i c_{(i,p(i))}(n_{(i,p(i))}^{off}(\mathbf{d}')) S_i \\
& = \gamma_i^T S_i \left(Q_i + \sum_{j \in f(\mathbf{r}_i)} c_j(n_j^{off}(\mathbf{d}) + 1) \right. \\
& \quad \left. + \frac{\gamma_i^E W_i}{\gamma_i^T} c_{(i,p(i))}(1) \right) \\
& = \gamma_i^T S_i \Delta \Phi. \tag{22}
\end{aligned}$$

Case 2: We can prove similarly as Case 1 that if $\mathbf{d}_i = (1, \mathbf{r}_i)$ and $\mathbf{d}'_i = (0, \mathbf{r}_i)$, $\Delta C_i = \gamma_i^T S_i \Delta \Phi$.

Case 3: For a free robot i , if it is a better decision to use another path \mathbf{r}'_i to offload than the original path \mathbf{r}_i :

$$\begin{aligned}
\Delta \Phi & = \Phi(\mathbf{d}_{-i}, \mathbf{d}'_i) - \Phi(\mathbf{d}_{-i}, \mathbf{d}_i) \\
& = \sum_{j \in f(\mathbf{r}'_i) - f(\mathbf{r}_i)} c_j(n_j^{off}(\mathbf{d}) + 1) - \sum_{j \in f(\mathbf{r}_i) - f(\mathbf{r}'_i)} \\
& \quad c_j(n_j^{off}(\mathbf{d})) + \frac{\gamma_i^E W_i}{\gamma_i^T} (c_{(i,p'(i))}(1) - c_{(i,p(i))}(1)), \tag{23}
\end{aligned}$$

where we can use $f(\mathbf{r}'_i) - f(\mathbf{r}_i)$ to find the set of links in path \mathbf{r}'_i but not in path \mathbf{r}_i .

$$\begin{aligned}
\Delta C_i & = (\gamma_i^E E_{i,p'(i)}^{off}((\mathbf{d}_{-i}, \mathbf{d}'_i), M_i) + \gamma_i^T T_{cloud}^{comp}((\mathbf{d}_{-i}, \mathbf{d}'_i), M_i) \\
& + \gamma_i^T S_i \sum_{j \in f(\mathbf{r}'_i)} c_j(n_j^{off}(\mathbf{d}_{-i}, \mathbf{d}'_i))) - (\gamma_i^E E_{i,p(i)}^{off}((\mathbf{d}_{-i}, \mathbf{d}_i), M_i) \\
& + \gamma_i^T T_{cloud}^{comp}((\mathbf{d}_{-i}, \mathbf{d}_i), M_i) + \gamma_i^T S_i \sum_{j \in f(\mathbf{r}_i)} c_j(n_j^{off}(\mathbf{d}_{-i}, \mathbf{d}_i))) \\
& = \left(\gamma_i^E W_i c_{(i,p'(i))}(n_{(i,p'(i))}^{off}(\mathbf{d})) S_i \right. \\
& \quad \left. + \gamma_i^T S_i \sum_{j \in f(\mathbf{r}'_i) - f(\mathbf{r}_i)} c_j(n_j^{off}(\mathbf{d}) + 1) \right) \\
& - \left(\gamma_i^E W_i c_{(i,p(i))}(n_{(i,p(i))}^{off}(\mathbf{d})) S_i \right. \\
& \quad \left. + \gamma_i^T S_i \sum_{j \in f(\mathbf{r}_i) - f(\mathbf{r}'_i)} c_j(n_j^{off}(\mathbf{d})) \right) \\
& = \gamma_i^T S_i \left(\sum_{j \in f(\mathbf{r}'_i) - f(\mathbf{r}_i)} c_j(n_j^{off}(\mathbf{d}) + 1) - \sum_{j \in f(\mathbf{r}_i) - f(\mathbf{r}'_i)} \right. \\
& \quad \left. c_j(n_j^{off}(\mathbf{d})) + \frac{\gamma_i^E W_i}{\gamma_i^T} (c_{(i,p'(i))}(1) - c_{(i,p(i))}(1)) \right) \\
& = \gamma_i^T S_i \Delta \Phi. \tag{24}
\end{aligned}$$

The above analysis shows that $\Delta C_i = \gamma_i^T S_i \Delta \Phi$ holds in all cases, so G is an ω -potential game with the weight $\omega = \{\gamma_i^T S_i\}_{i \in \mathcal{P}}$. Furthermore because the number of decision pro-

files is finite, $\Phi(\mathbf{d})$ cannot decrease indefinitely and the above algorithm must terminate after a finite number of improvement steps. We can conclude that the game has an NE, so Theorem 1 is proven, so the ImproveRobot algorithm can apply to the scenario of robot swarms in the case of an unsliceable task.

APPENDIX B PROOF OF THEOREM 2

According to the characteristics of game H , we define a potential function $\Omega : \mathbf{Y} \rightarrow R$ as:

$$\begin{aligned}
\Omega(\mathbf{y}) & = \sum_{i \in \mathcal{L}} \sum_{j=0}^{n_i(\mathbf{d})} c_i(j) - \sum_{i \in \mathcal{P}} Q_i (1 - \alpha_i) \\
& \quad + \sum_{i \in \mathcal{P}} \frac{\gamma_i^E W_i}{\gamma_i^T} c_{(i,p(i))}(1) \alpha_i, \tag{25}
\end{aligned}$$

where $Q_i = \frac{L_i}{S_i} (\frac{1}{F_c} - \frac{1}{F_i} - \frac{\gamma_i^E \mu_i}{\gamma_i^T})$ and Ω is continuously differentiable.

Therefore, we get:

$$\begin{aligned}
\frac{\partial U_i(\mathbf{y})}{\partial \alpha_i} & = C_i^{off}(\mathbf{d}) - C_i^0 \tag{26} \\
\frac{\partial \Omega(\mathbf{y})}{\partial \alpha_i} & = Q_i + \frac{\gamma_i^E W_i}{\gamma_i^T} c_{(i,p(i))}(1) \\
& = \frac{1}{\gamma_i^T S_i} \left(\gamma_i^T \frac{L_i}{F_c} + \gamma_i^E W_i S_i c_{(i,p(i))}(1) - \gamma_i^T \frac{L_i}{F_i} - \gamma_i^E \mu_i S_i \right) \\
& = \frac{1}{\gamma_i^T S_i} (C_i^{off}(\mathbf{d}) - C_i^0) = \frac{1}{\gamma_i^T S_i} \frac{\partial U_i(\mathbf{y})}{\partial \alpha_i} \tag{27}
\end{aligned}$$

From the above, we can see that $\frac{\partial U_i(\mathbf{y})}{\partial \alpha_i}$ is equal to $C_i^{off}(\mathbf{d}) - C_i^0$; therefore, if $C_i^{off}(\mathbf{d}) > C_i^0$, then $\frac{\partial U_i(\mathbf{y})}{\partial \alpha_i}$ is positive and $U_i(\mathbf{y})$ does not reach its maximum value until $\alpha_i = 1$. Otherwise, if $C_i^{off}(\mathbf{d}) < C_i^0$, then $U_i(\mathbf{y})$ does not reach its maximum value until $\alpha_i = 0$. Furthermore, if $C_i^{off}(\mathbf{d}) = C_i^0$, $U_i(\mathbf{d})$ remains constant for any α_i and we can assume $\alpha_i = 1$ in this case. In conclusion, sliceable tasks can be reduced to unsliceable tasks because every robot only has two conditions, i.e., $\alpha_i = 1$ or $\alpha_i = 0$, and there are no intermediate states.

For the free player i who offloads all computing, i.e., $\alpha_i = 1$, keeping α_i unchanged, then if it is a better decision to use the path of \mathbf{r}'_i to transfer its task than to use \mathbf{r}_i , so that $\mathbf{y}'_i = (\alpha_i, \mathbf{r}'_i)$ is a better decision than $\mathbf{y}_i = (\alpha_i, \mathbf{r}_i)$, then:

$$\begin{aligned}
\Delta \Omega & = \Omega(\mathbf{y}_{-i}, \mathbf{y}'_i) - \Omega(\mathbf{y}_{-i}, \mathbf{y}_i) \\
& = \sum_{j \in f(\mathbf{r}'_i) - f(\mathbf{r}_i)} c_j(n_j^{off}(\mathbf{d}) + 1) \\
& - \sum_{j \in f(\mathbf{r}_i) - f(\mathbf{r}'_i)} c_j(n_j^{off}(\mathbf{d})) \\
& + \alpha_i \frac{\gamma_i^E W_i}{\gamma_i^T} (c_{(i,p'(i))}(1) - c_{(i,p(i))}(1)) \tag{28}
\end{aligned}$$

$$\begin{aligned}
\Delta U_i & = \alpha_i ((\gamma_i^E E_{i,p'(i)}^{off}((\mathbf{d}_{-i}, \mathbf{d}'_i), M_i) \\
& + \gamma_i^T T_{cloud}^{comp}((\mathbf{d}_{-i}, \mathbf{d}'_i), M_i)
\end{aligned}$$

$$\begin{aligned}
& +\gamma_i^T S_i \sum_{j \in f(\mathbf{r}_i)} c_j(n_j^{off}(\mathbf{d}_{-i}, \mathbf{d}'_i))) \\
& -(\gamma_i^E E_{i,p(i)}^{off}((\mathbf{d}_{-i}, \mathbf{d}_i), M_i) \\
& +\gamma_i^T T_{cloud}^{comp}((\mathbf{d}_{-i}, \mathbf{d}_i), M_i) \\
& +\gamma_i^T S_i \sum_{j \in f(\mathbf{r}_i)} c_j(n_j^{off}(\mathbf{d}_{-i}, \mathbf{d}_i))) \\
& = \alpha_i \gamma_i^T S_i \left(\sum_{j \in f(\mathbf{r}_i) - f(\mathbf{r}_i)} c_j(n_j^{off}(\mathbf{d}) + 1) \right. \\
& \quad \left. - \sum_{j \in f(\mathbf{r}_i) - f(\mathbf{r}'_i)} c_j(n_j^{off}(\mathbf{d})) \right. \\
& \quad \left. + \frac{\gamma_i^E W_i}{\gamma_i^T} (c_{(i,p'(i))}(1) - c_{(i,p(i))}(1)) \right) \\
& = \alpha_i \gamma_i^T S_i \Delta \Omega \tag{29}
\end{aligned}$$

Therefore, according to Eq. (29), sliceable tasks can be reduced to unsliceable tasks. We can still use ImproveRobot Algorithm to solve MCCM in the case of sliceable tasks.

APPENDIX C PROOF OF THEOREM 3

To compute a worst social cost of NE, we set \mathbf{d}^* as an arbitrary NE and divide all robots into two groups such that one group contains the robots in the outermost layer \mathcal{P}_L while the other group contains the remaining robots. For the robots in the outermost layer, \mathcal{P}_L , $C_i(\mathbf{d}_i^*, \mathbf{d}_{-i}^*) \leq C_i^0$ holds for every robot i , and the worst case for them is that they all perform local computations. Because robots in the outermost layer have no relay tasks, they can change their decisions freely according to their respective interests. Thus, if $\exists i \in \mathcal{P}_L$, $C_i(\mathbf{d}_i^*, \mathbf{d}_{-i}^*) > C_i^0$, robot i can change its decision to compute locally, which contradicts our assumption that \mathbf{d}^* is an NE. For the other part $i \in \mathcal{P} \setminus \mathcal{P}_L$, robots may be locked. To find the worst condition, we assume not only that all of them are locked but also that they choose the worst action. Furthermore, if computation is offloaded and robot $i \in \mathcal{P} \setminus \mathcal{P}_L$ chooses the path \mathbf{r}_i to perform the offload, we can consider the worst case is that all the robots use \mathbf{r}_i to offload, which make the congestion of \mathbf{r}_i the most serious. Robot i must then be the relay for each robot in the outer layer. In addition, to create the worst case, we can assume \mathbf{r}_i is also the slowest path for i , i.e., $\sum_{j \in f(\mathbf{r}_i)} R_j = \min_{\mathbf{r}_i \in \mathcal{L}_i} \sum_{j \in f(\mathbf{r}_i)} R_j$ where \mathcal{L}_i is the feasible set of paths for i . For local computation, their cost equals C_i^0 . Thus, in any NE, we have $\max_{\mathbf{d}} \sum_{i \in \mathcal{P} \setminus \mathcal{P}_L} C_i(\mathbf{d}^*) \leq \max\{C_i^0, \max_{\mathbf{d} \in \mathbf{D}} \{C_i^{off}(\mathbf{d})\}\} + \max_{\mathbf{d} \in \mathbf{D}} C_i^{relay}(\mathbf{d})$.

We now derive a lower bound for the optimum social cost in the MCCG. First, we set $\mathbf{d}^{optimal} \in \mathbf{D}$ as the optimal decision in the game. For robot i , if $\forall \mathbf{d} \in \mathbf{D}$, $C_i^0 \leq C_i(\mathbf{d})$, then the best decision for i is to compute locally, i.e., $C_i(\mathbf{d}^{optimal}) = C_i^0$. Otherwise, if offloading the computation through \mathbf{r}_i is a better decision for i and we know that in the best case no others compute offload through \mathbf{r}_i^* , i.e., $\forall j \in f(\mathbf{r}_i^*), n_j^{off}(\mathbf{d}) = 1$, and \mathbf{r}_i^* is also the fastest path for i , i.e., $\sum_{j \in f(\mathbf{r}_i^*)} R_j = \max_{\mathbf{r}_i \in \mathcal{L}_i} \sum_{j \in f(\mathbf{r}_i)} R_j$. For each robot i , if it has no relay tasks, i.e., $C_i^{relay}(\mathbf{d}) =$

0, its total cost will be less. Thus, $\min_{\mathbf{d} \in \mathbf{D}} \sum_{i \in \mathcal{P}} C_i(\mathbf{d}) \geq \sum_{i \in \mathcal{P}} \min\{C_i^0, \min_{\mathbf{d} \in \mathbf{D}} \{C_i^{off}(\mathbf{d})\}\}$.

Hence, we can prove the above theorem using the upper bound for the arbitrary NE and the lower bound for the optimal decision:

$$\begin{aligned}
PoA &= \frac{\max_{\mathbf{d}} \sum_{i \in \mathcal{P}} C_i(\mathbf{d}^*)}{\min_{\mathbf{d} \in \mathbf{D}} \sum_{i \in \mathcal{P}} C_i(\mathbf{d})} \\
&\leq \frac{\sum_{i \in \mathcal{P}_L} C_i^0 + \sum_{i \in \mathcal{P} \setminus \mathcal{P}_L} \max_{\mathbf{d} \in \mathbf{D}} C_i(\mathbf{d})}{\sum_{i \in \mathcal{P}} \min\{C_i^0, \min_{\mathbf{d} \in \mathbf{D}} \{C_i^{off}(\mathbf{d})\}\}} \\
&= \frac{\sum_{i \in \mathcal{P}_L} C_i^0 + \sum_{i \in \mathcal{P} \setminus \mathcal{P}_L} V_i}{\sum_{i \in \mathcal{P}} K_i}
\end{aligned}$$

REFERENCES

- [1] M. Saska, V. Vonásek, J. Chudoba, J. Thomas, G. Loianno, and V. Kumar, "Swarm distribution and deployment for cooperative surveillance by micro-aerial vehicles," *J. Intell. Robot. Syst.*, vol. 84, no. 1-4, pp. 469-492, 2016.
- [2] F. Tang, Z. M. Fadlullah, N. Kato, F. Ono, and R. Miura, "AC-POCA: Anticoordination game based partially overlapping channels assignment in combined UAV and D2D-based networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 2, pp. 1672-1683, Feb. 2018, doi: [10.1109/TVT.2017.2753280](https://doi.org/10.1109/TVT.2017.2753280).
- [3] Z. M. Fadlullah, D. Takaishi, H. Nishiyama, N. Kato, and R. Miura, "A dynamic trajectory control algorithm for improving the communication throughput and delay in UAV-aided networks," *IEEE Netw.*, vol. 30, no. 1, pp. 100-105, Jan.-Feb. 2016, doi: [10.1109/MNET.2016.7389838](https://doi.org/10.1109/MNET.2016.7389838).
- [4] F. Tang, Z. M. Fadlullah, B. Mao, N. Kato, F. Ono, and R. Miura, "On a novel adaptive UAV-mounted cloudlet-aided recommendation system for LBSNs," *IEEE Trans. Emerg. Topics Comput.*, to be published, doi: [10.1109/TETC.2018.2792051](https://doi.org/10.1109/TETC.2018.2792051).
- [5] H. Osumi, "Application of robot technologies to the disaster sites," JSME Res. Committee, Japan Soc. Mech. Eng., 2014, pp. 58-73.
- [6] J. Salmern-Garcia, P. Igo Blasco, F. D. del Ro, and D. Cagigas-Muiz, "A tradeoff analysis of a cloud-based robot navigation assistant using stereo image processing," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 444-454, Apr. 2015.
- [7] S. Barbarossa, S. Sardellitti, and P. D. Lorenzo, "Computation offloading for mobile cloud computing based on wide cross-layer optimization," in *Proc. IEEE Future Netw. Mobile Summit*, 2013, pp. 1-10.
- [8] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 974-983, Apr. 2015.
- [9] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795-2808, Oct. 2016.
- [10] X. Ma, C. Lin, X. Xiang, and C. Chen, "Game-theoretic analysis of computation offloading for cloudlet-based mobile cloud computing," in *Proc. 18th ACM Int. Conf. Model., Anal., Simul. Wireless Mobile Syst.*, 2015, pp. 271-278.
- [11] S. Joilo and G. Dn, "A game theoretic analysis of selfish mobile computation offloading," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1-9.
- [12] L. Yang, J. Cao, S. Tang, T. Li, and A. T. S. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," in *Proc. IEEE Fifth Int. Conf. Cloud Comput.*, Jun. 2012, pp. 794-802.
- [13] T. G. Rodrigues, K. Suto, H. Nishiyama, N. Kato, and K. Temma, "Cloudlets activation scheme for scalable mobile edge computing with transmission power control and virtual machine migration," *IEEE Trans. Comput.*, vol. 67, no. 9, pp. 1287-1300, Sep. 2018, doi: [10.1109/TC.2018.2818144](https://doi.org/10.1109/TC.2018.2818144).
- [14] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through VM migration and transmission power control," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 810-819, May 2017, doi: [10.1109/TC.2016.2620469](https://doi.org/10.1109/TC.2016.2620469).
- [15] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, "Cloud-based robot grasping with the Google object recognition engine," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2013, pp. 4263-4270.

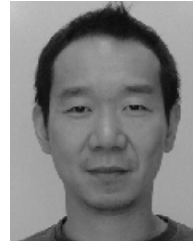
- [16] H. He, S. Kamburugamuve, G. Fox, and W. Zhao, "Cloud based real-time multi-robot collision avoidance for swarm robotics," *Int. J. Grid Distrib. Comput.*, vol. 9, no. 6, pp. 339–358, 2016.
- [17] G. Mohanarajah, V. Usenko, M. Singh, R. D'Andrea, and M. Waibel, "Cloud-based collaborative 3d mapping in real-time with low-cost robots," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 423–431, Apr. 2015.
- [18] P. Pandey, D. Pompili, and J. Yi, "Dynamic collaboration between networked robots and clouds in resource-constrained environments," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 471–480, Apr. 2015.
- [19] Y. Nan *et al.*, "Adaptive energy-aware computation offloading for cloud of things systems," *IEEE Access*, vol. 5, pp. 23947–23957, Oct. 2017.
- [20] A. Rahman, J. Jin, A. Cricenti, A. Rahman, and D. Yuan, "A cloud robotics framework of optimal task offloading for smart city applications," in *Proc. IEEE Global Commun. Conf.*, Dec. 2016, pp. 1–7.
- [21] Y. Li, H. Wang, B. Ding, P. Shi, and X. Liu, "Toward QoS-aware cloud robotic applications: A hybrid architecture and its implementation," in *Proc. Int. IEEE Conf. Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data Comput., Internet People, Smart World Congress*, Jul. 2016, pp. 33–40.
- [22] H. Ko, J. Lee, and S. Pack, "Spatial and temporal computation offloading decision algorithm in edge cloud-enabled heterogeneous networks," *IEEE Access*, vol. 6, pp. 18920–18932, Mar. 2018.
- [23] Third Generation Partnership Project (3GPP). [Online]. Available: <http://www.3gpp.org/>, accessed on: Dec. 20, 2018.
- [24] P. Shantharama, A. S. Thyagaturu, N. Karakoc, L. Ferrari, M. Reisslein, and A. Scaglione, "Layback: SDN management of multi-access edge computing (MEC) for network access services and radio resource sharing," *IEEE Access*, vol. 6, pp. 57 545–57 561, Oct. 2018.
- [25] O. Narmanlioglu, E. Zeydan, and S. S. Arslan, "Service-aware multi-resource allocation in software-defined next generation cellular networks," *IEEE Access*, vol. 6, pp. 20348–20363, Mar. 2018.
- [26] X. Li *et al.*, "5G-crosshaul network slicing: Enabling multi-tenancy in mobile transport networks," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 128–137, Aug. 2017.
- [27] D. Monderer and L. S. Shapley, "Potential games," *Games Econ. Behav.*, vol. 14, no. 1, pp. 124–143, 1996.
- [28] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. Cambridge, U.K.: Cambridge Univ. Press, 2007.
- [29] S. Josilo, "Decentralized algorithms for resource allocation in mobile cloud computing systems," Ph.D. dissertation, KTH Roy. Inst. Technol., Stockholm, Sweden, 2018.
- [30] M. Mozaffari, W. Saad, M. Bennis, and M. Debbah, "Unmanned aerial vehicle with underlaid device-to-device communications: Performance and tradeoffs," *IEEE Trans. Wireless Commun.*, vol. 15, no. 6, pp. 3949–3963, Jun. 2016.
- [31] Y. Zeng and R. Zhang, "Energy-efficient UAV communication with trajectory optimization," *IEEE Trans. Wireless Commun.*, vol. 16, no. 6, pp. 3747–3760, Jun. 2017.
- [32] C. Fu, R. Duan, D. Kircali, and E. Kayacan, "Onboard robust visual tracking for UAVs using a reliable global-local object model," *Sensors*, vol. 16, no. 9, p. 1406, 2016.
- [33] T. Soyata, R. Muraledharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *Proc. IEEE Symp. Comput. Commun.*, Jul. 2012, pp. 59–66.
- [34] DJI, "Mavic 2 specs." Accessed. [Online]. Available: <https://www.dji.com/cn/mavic-2/info#specs>, Dec. 20, 2018.



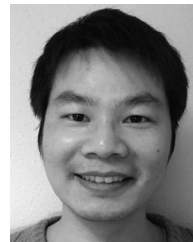
Zicong Hong is currently working toward the B.Eng. degree in software engineering at the School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China. His current research interests include game theory, internet of things, and blockchain and edge/cloud computing.



He received the Best Paper Award in the International Conference on Trust, Security, and Privacy in Computing and Communications in 2016.



His research has been sponsored by the Japan Society for the Promotion of Science, the JST, the MIC, the National Science Foundation, the National Natural Science Foundation of China, and the industrial companies. He has served as an Editor of several journals, including IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING, IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING, IEEE COMMUNICATIONS MAGAZINE, and *Wireless Networks*. He has been actively participating in international conferences serving as the General Chair and the Technical Program Committee Chair. He is a Senior Member of the Association for Computing Machinery. He is the IEEE Communications Society Distinguished Lecturer.



He is the IEEE Communications Society Distinguished Lecturer.



and the IBM Ph.D. Fellowship Award at 2010. He served as program committee (PC) member of the IEEE International Conference on Cloud Computing, the International Conference on Web Services, the international conference on service computing (SCC), the International Conference on Service-Oriented Computing, the international symposium on service-oriented system engineering (SOSE), etc.

Huawei Huang (M'16) received the Ph.D. degree in computer science and engineering from the University of Aizu, Aizu-Wakamatsu, Japan. He was a Visiting Scholar with The Hong Kong Polytechnic University from 2017 to 2018. He was a Post-Doctoral Research Fellow with the Japan Society for the Promotion of Science between 2016 and 2018. He was an Assistant Professor with Kyoto University, Japan, from 2018 to 2019. He is currently an Associate Professor with Sun Yat-Sen University, Guangzhou, China. His research interests mainly include SDN/NFV, edge computing, and blockchain. He received the Best Paper Award in the International Conference on Trust, Security, and Privacy in Computing and Communications in 2016.

Song Guo (M'02–SM'11) received the Ph.D. degree in computer science from the University of Ottawa, Ottawa, ON, Canada. He is currently a Full Professor in the Department of Computing, The Hong Kong Polytechnic University (PolyU), Kowloon, Hong Kong. Prior to joining PolyU, he was a Full Professor with the University of Aizu, Aizu-Wakamatsu, Japan. His research interests are mainly in the areas of cloud and green computing, big data, wireless networks, and cyber-physical systems. He has published more than 300 conference and journal papers in these

Wuhui Chen received the Bachelor's degree from Northeast University, Shenyang, China, in 2008 and the Master and Ph.D. degrees from the University of Aizu, Aizu-Wakamatsu, Japan, in 2011 and 2014, respectively. From 2014 to 2016, he was a Research Fellow with the Japan Society for the Promotion of Science, Japan. From 2016 to 2017, he was a Researcher with the University of Aizu. He is currently an Associate Professor with Sun Yat-Sen University, Guangzhou, China. His research interests include edge/cloud computing, cloud robotics, and blockchain.

Zibin Zheng is currently a Professor with the School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China. His research interests include service computing and cloud computing. He received the Outstanding Ph.D. Thesis Award of the Chinese University of Hong Kong in 2012, the Association for Computing Machinery's Special Interest Group on Software Engineering Distinguished Paper Award at the International Conference on Science and Engineering, 2010, the Best Student Paper Award at the International Conference on Web Services, 2010,