Computation Offloading With Data Caching Enhancement for Mobile Edge Computing

Shuai Yu, *Member, IEEE*, Rami Langar, *Member, IEEE*, Xiaoming Fu, *Senior Member, IEEE*, Li Wang, *Senior Member, IEEE*, and Zhu Han, *Fellow, IEEE*

***Abstract*—Computation offloading is a proven successful paradigm for enabling resource-intensive applications on mobile devices. Moreover, in view of emerging mobile collaborative application, the offloaded tasks can be duplicated when multiple users are in the same proximity. This motivates us to design a collaborative offloading scheme and cache the popular computation results that are likely to be reused by other mobile users. In this paper, we consider the scenario where multiple mobile users offload duplicated computation tasks to the network edge, and share the computation results among them. Our goal is to develop the optimal fine-grained collaborative offloading strategies with caching enhancements to minimize the overall execution delay at the mobile terminal side. To this end, we propose an optimal offloading with caching-enhancement scheme (OOCS) for femto-cloud scenario and mobile edge computing scenario, respectively. Simulation results show that compared to six alternative solutions in literature, our single-user OOCS can reduce execution delay up to 42.83% and 33.28% for single-user femto-cloud and single-user mobile edge computing, respectively. Our multi-user OOCS can further reduce 11.71% delay compared to single-user OOCS through users' cooperation.**

***Index Terms*—Computation Offloading, Data Caching, Mobile Collaborative Applications, Mobile Edge Computing, Coalitional Game.**

Manuscript received February 10, 2018; revised June 6, 2018; accepted August 18, 2018. Date of publication September 10, 2018; date of current version November 12, 2018. This work was supported in part by the FUI PODIUM project under Grant 10703, in part by the National Science Foundation of China under Grant U1711265, in part by the Fundamental Research Funds for the Central Universities under Grant 17lgjc40, in part by the Program for Guangdong Introducing Innovative and Enterpreneurial Teams under Grant 2017ZT07X355, and in part by the US MURI under Grants NSF CNS-1717454, CNS- 1731424, CNS-1702850, CNS-1646607, and ECCS-1547201. The review of this paper was coordinated by A.-C. Pang. *(Corresponding author: Shuai Yu.)*

S. Yu is with Laboratoire d'Informatique de Paris 6 (LIP6), Sorbonne Université, Paris 75005, France, and also with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510275, China (e-mail: shuai.yu@lip6.fr).

R. Langar is with the Laboratoire d'Informatique Gaspard-Monge, University Paris Est Marne-la-Vallée (UPEM), Champs-sur-Marne 77420, France (e-mail: Rami.Langar@u-pem.fr).

X. Fu is with the Institute of Computer Science, University of Göttingen, Göttingen 37077, Germany (e-mail: fu@cs.uni-goettingen.de).

L. Wang is with the Key Laboratory of the Universal Wireless Communications, Ministry of Education, Beijing University of Posts and Telecommunications, Beijing 100876, China, and also with the School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: liwang@bupt.edu.cn).

Z. Han is with the Laboratoire d'Informatique Gaspard-Monge, University of Houston, Houston, TX 77004 USA, and also with the Department of Computer Science and Engineering, Kyung Hee University, Seoul 02447, South Korea (e-mail: zhan2@uh.edu).

A preliminary version of this paper appeared in the proceedings of the 2016 IEEE International Conference on Communications (ICC 2016) [1].

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVT.2018.2869144

I. Introduction

NOWADAYS, the advances in hardware technology enable our smartphones equipped not only with more memory, better processor and improved network connectivity, but also with numerous sensors. Accompanied by the emergence of near-to-eye display technologies, such as Google Glass, a variety of mobile collaborative applications (MCA) [2], such as mobile audio/video conferences, collaborative editing and augmented reality [3], are designed to support collaboration among mobile users. These applications use complex algorithms for camera tracking and object recognition, requiring mobile devices (google glass, smartphones, etc.) not only with considerable energy, memory size and computational resources, but also with resource sharing capacity.

A potential solution to address the challenges is through mobile cloud computation offloading [4]. However, when considering 1 millisecond to 5 milliseconds end-to-end latency required by 5G for a class of applications (called the "Tactile Internet" [5]), the traditional cloud may not be suitable for code offloading due to the high and variable latency to distant datacenters, especially for delay-sensitive applications. On the other hand, caching most popular contents at the network edge can reduce latency and improve user's quality of experience (QoE) [6], [7]. Thus, a promising approach to tackle this challenge is to move cloud infrastructure (computation and storage abilities) closer to the end users [8].

Motivated by above facts, the European project TROPIC (distributed computing, storage and radio resource allocation over cooperative femtocells) and the European Telecommunications Standards Institute (ETSI) proposed femto-cloud [9] and multi-access mobile edge computing (MEC) [10], respectively. In the proposed architectures, fixed and powerful servers are located at the network edge to reduce communication overhead and execution delay for mobile users.

In this paper, we will investigate collaborative computation offloading with the data caching enhancement strategy for the MCA execution in femto-cloud and MEC, respectively. Our objective is to reduce the average execution delay for the mobile users within the network. Our proposal can dramatically reduce the MCA's execution latency based on the following two facts:

- Multiple mobile users in the same MCA execution environment can share computation and outcome results. Indeed, through sharing the same components (e.g., Mapper and Objective Recognizer as in [2]) and computation results (e.g., detected feature points of environment), the

environment will be generated faster and more complete. Through this kind of cooperation and sharing, mobile users can not only save computational resources but also gain information from the input of others.

- For cache-enabled MEC (e.g., mobile cloud gaming), caching parts of computation results that are likely to be reused by others can further boost the computation performance of the entire system [11]. This idea is motivated by the fact that users in one small region are likely to request similar computing services. For example, visitors in a museum tend to use Augmented Reality (AR) for better sensational experience. Thus, it is desirable to cache multiple AR services and output data at the MEC server of this region to provide the real-time services.

The major contributions of our paper are summarised as follows:

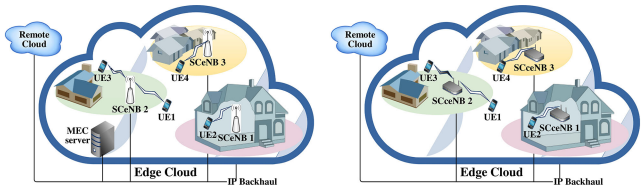
- We propose a fine-grained collaborative computation offloading and caching strategy that optimizes the offloading decisions on the mobile terminal side with caching enhancement. The objective is to minimize the overall execution latency for the mobile users within the network. Most of previous works either focus on the offloading decisions [4], [12]–[18] or caching placement strategy [7], [19]. To the best of our knowledge, our work is the first of its kind that optimizes offloading decisions, while considering data caching.
- Based on the concept of the call graph [20], we propose in this paper the concept of the cooperative call graph to model the offloading and caching relationship within multiple mobile users, and then compute the delay and energy overhead for each single user.
- We first evaluate our algorithm in a single-user scenario for both femto-cloud and MEC networks. Moreover, to further reduce the execution delay in the multi-user case, we explore the concept of the coalition formation game for the distributed caching scenario.
- We compare our approach with six benchmark policies in literature and discuss the associated gains in terms of required CPU cycles per byte (cpb) for applications and content's popularity factor.

The reminder of this paper is organized as follows. Section II, introduces an overview of related works. In Section III, we present the system model. Section IV introduces the collaborative call graph and formulates the optimaztion problems, followed by a description of our proposed algorithm in Section V. Simulation results are presented in Section VI. Finally, conclusions are drawn in Section VII.

II. Related Work

A. Mobile Edge Computing

In the context of mobile edge computing (MEC), the key element is the edge server, which provides computing resource, storage capability and connectivity. The server can be deployed at high-end or low-end proximate in the small cell based edge cloud network [21]. The high-end deployment is a current focus of the ETSI multi-access mobile edge computing standard, as



(a) High-end MEC deployment (ETSI multi-access mobile edge computing [12])

(b) Low-end MEC deployment (femto-cloud [11])

Fig. 1. High-end MEC deployment vs. Low-end MEC deployment: (a) The high-end server is located in the access network that is an aggregation point of a set of small cell base stations (SCeNB i.e., Small Cell e-Node B), (b) The low-end servers are traditional small cell base stations with cloud capacity (SCceNB, i.e., small cell cloud-enhanced e-Node B).

shown in Fig. 1 (a). In this case, typically a high-end standard server is located in the access network that is an aggregation point of a set of small cell base stations (SCeNB i.e., Small Cell e-Node B). The server is also regarded as a central coordinator that is required to design a cache placement strategy. However, in the low-end case, application servers can be deployed in low-end devices which can be routers, access points, or home-based nodes as shown in Fig. 1 (b). Femto-cloud [9], [15]–[17] is a typical low-end deployment. The idea is to endow small cell base stations with cloud functionalities (computation, storage and server), thus providing mobile user equipments with proximity access to the mobile edge cloud. The novel base stations are called small cell cloud-enhanced e-Node B (SCceNB) that deployed with high-capacity storage units but have limited capacity backhaul links. In this case, the popular contents are cached in a distributed manner among the SCceNBs without a central coordinator. With respect to the high-end deployment, this deployment brings three advantages: 1) strong reduction of latency with respect to centralized clouds, because small cells are the closest points in the mobile network to the mobile users with only one wireless hop, and therefore with minimum latency, 2) storage of large amounts of local contents or software by mobile equipments over proximity SCceNB, data can be temporarily stored (local caching) in the nearby SCceNB, with very low latency, and 3) no central coordinator is required to collect the information of the whole network, which significantly saves signaling overhead. In view of this, authors in [15] proposed a framework for joint optimization of radio and computation resource under energy consumption, computational and delay constraints. Sardellitti *et al.* in [16] considered a scenario composed by multiple mobile users asking for computation offloading of their applications to a set of cloud servers. The objective is to minimize the overall energy consumption, at the mobile terminal side, while meeting the latency constraints. In addition, TROPIC considers a fine-grained (i.e. code partitioning) offloading framework [17] which they exploits the concept of the call graph to model the application.

However, in most of these previous works, the main focus is on mobile devices, considering the cloud as a system with unlimited resources. Moreover, the offloading decision is based either on the optimization of computation/communication resources (as in [15], [16]) or on the code partitioning of each

single user (as in [17]). They do not indeed consider sharing computation results between multiple users to avoid repeated computation. It is worth noting that through sharing the computation results (by local caching), such relationship can affect the offloading decision for all mobile users.

B. Computation Offloading and Data Caching in MEC

In order to meet the low latency and improved QoE requirements of emerging applications, we need code offloading and local caching strategies that empower mobile applications with resourceful cloud equipments. Offloading mechanisms have been studied extensively [13], [14], [22]–[26] and generally fallen into two categories: coarse-grained offloading [13] and fine-grained (typically at a method-level granularity) offloading [14], [22]–[26]. Coarse-grained offloading also refers to the full offloading or total offloading in which full task is migrated to the cloud. This approach does not require estimating the computation overhead prior to the execution. For example, Zhang *et al.* [13] proposed a computational offloading policy to decide whether an entire application should be offloaded to remote cloud or executed locally to reduce energy consumption on the mobile terminal side. However, fine-grained offloading (partial offloading or dynamic offloading) dynamically transmits as little code as possible and offloads only the computation-hungry parts of the application. Zhang *et al.* [14] investigated the problem of collaborative task execution by strategically offloading task to the cloud. They proposed a fine-grained offloading strategy to reduce energy consumption under a latency constraint. Despite the introduced burden on the application programmers and additional computational overhead, the proposed approach can reduce unnecessary transmission overhead, achieving a reduced latency and energy consumption. Moreover, a lot of current works study the Device-to-Device (D2D) computation offloading [12] or D2D caching [6], [27], [28] strategies, in order to reduce latency or energy consumption for mobile users. This means that mobile devices can be regarded as servers for other mobile users. However, compared with some fixed servers, mobile devices remain resource poor. Especially, due to their mobility, it is hard for mobile devices to cache some popular computation results in a fixed area.

Many emerging mobile applications involve intensive computation based on data analytics, thus caching parts of computation results that are likely to be reused by others can further boost the computation performance of the entire MEC system [11]. One typical example is mobile cloud gaming [29]. Note that certain game rendered videos, e.g., gaming scenes, can be reused by other players, caching these computation results would not only significantly reduce the computation latency of the players with the same computation request, but also ease the computation burden for edge servers. In the high-end MEC case, massive data are cached in a centralized manner. Thus, the key problem is how to balance the tradeoff between massive database and limited storage capacity of MEC servers. Whereas in the low-end MEC case, the design principle is how to cache the massive data in the distributed low-end MEC servers. Furthermore, it is also essential to establish a practical database popularity distribution model that is able to statistically characterize the usage of each database set for different MEC applications. We will address the above concerns and design optimal offloading strategies leveraging local data caching, as proposed in this paper.

C. Computation and Data Sharing for Mobile Collaborative Applications

For multi-user computation offloading scenario, it is hard to share the output data when the mobile users run different applications. However, it is possible to share the output data when i) multiple mobile users run the same components (subtasks) of the same application, or ii) multiple mobile users run different components of the same application. Thus, it make sense to i) select part of mobile users to execute the common components or ii) allocate the components to different mobile users for parallel processing. The output data are shared among the corresponding mobile users. As a result, the execution energy and delay can be reduced significantly. In this paper, we consider the first sharing scenario, which means that all mobile users run the same component and share part of the data. It is worth noting that the second sharing scenario has been extensively studied in our recent work in [30].

For certain types of MCA, multiple users in the same neighborhood typically look at the same scene, track the same environment, and need to recognize the same objects, so they can benefit from collaboration and computation/data sharing [31]. A typical example is emerging mobile crowd sensing applications [32]–[34], where individual mobile user with sensing and computing devices collectively share data and extract information to measure and map phenomena of common interest. Similarly, AR applications [2], [31] have the unique property that different users with the same objective can share part of the computational tasks and of the input and output data. Verbelen *et al.* [2] proposed a component-based offloading framework that optimizes application-specific metrics. They split an immersive application into several loosely coupled software components. Each components with its dependency, configuration parameters and constraints can be offloaded and shared among multiple users. Specifically, one kind of Mapper component can be shared between multiple mobile devices in the same physical environment. All mobile users receive the same world model to track the camera position and share the computation results. Because the model is updated and refined using camera images from multiple devices, the model will often be more accurate than one created by just one device. Through this computation/data sharing, the cloudlet agent allows users to not only save computational resources to avoid repeated calculation, but also gain information from the input of others. However, they only consider a two-user scenario with one kind of immersive application executed in a laptop, the data rate is set to be a fixed value. It is not enough to model the scenario where multiple mobile users process MCAs in MEC.

To overcome these limitations, we first propose in [1] a cooperative computation offloading strategy based on coalitional game formation for multi-cell and multi-user femtocloud

networks. In this paper, we extend our previous work and propose a new computation offloading strategy with caching enhancements to further minimize the overall execution delay at the mobile terminal side. Our proposal, namely Optimal Offloading with Caching-enhancement Scheme (OOCS) is based on the new concept of collaborative call graph. In addition, we consider both high-end and low-end MEC deployments in our analysis and add new baselines for performance comparison.

III. System Model

In this section, we present the model of our optimal offloading with caching-enhancement scheme (OOCS).

A. Network Model

In our work, we consider both high-end and low-end deployments of small cell-based mobile edge cloud. In the case of low-end deployment, we consider an LTE femto-cloud network composed of N_l small cells with computation and memory enhancement (i.e. SCceNBs in LTE terminology). In the case of high-end deployment, we consider an edge cloud network consisting of N_h traditional small cell base stations (SCeNBs) and a centralized high-end server. The number of mobile users for both network types is M. We also consider both of the network deployments are based on the orthogonal frequency division multiple-access (OFDMA) in which M users within the same SCeNB/SCceNB are separated in the frequency domain. Note that, using such a transmission scheme for the uplink offloading implies that the users do not interfere with one another.

Let p_u and p_s denote the transmit power for the UEs and small cells (i.e., SCeNBs and SCceNBs), respectively. The maximum achievable rate [35], [36] (in bps) over an additive white Gaussian noise (AWGN) channel for user m ($m \in \mathcal{M}$) to offload its application to small cell n ($n \in \mathcal{N}_h$ or $n \in \mathcal{N}_l$) can be expressed as follows:

$$r_{n,m}^{ul} = B \log_2 \left(1 + \frac{p_u |h_{ul}|^2}{\Gamma(g_{ul}) d^{\beta} N_0}\right) \tag{1}$$

where B is the bandwidth, d is the distance between UE and SCeNB/SCceNB. In this paper, we consider the Rayleigh-fading environment, and h_{ul} and h_{dl} are the channel fading coefficient for uplink and downlink, respectively. N_0 denotes the noise power and β is the path loss exponent. Note that $\Gamma(BER) = -\frac{2log(5BER)}{3}$ represents the SNR margin introduced to meet the desired target bit error rate (BER) with a QAM constellation [37]. g_{ul} and g_{dl} are the target BER for uplink and downlink, respectively.

Similarly, the maximum achievable rate (in bps) for a user m receiving its computation results from small cell n ($n \in \mathcal{N}_h$ or $n \in \mathcal{N}_l$) is given by:

$$r_{n,m}^{dl} = B \log_2 \left(1 + \frac{p_s |h_{dl}|^2}{\Gamma(g_{dl}) d^{\beta} N_0}\right). \tag{2}$$

B. Application Model

We assume that a mobile application can be split into multiple components [2] which in the granularity of either method [22]–[26] or thread [13] (i.e., a fine-grained partitioning). We then exploit the concept of the call graph [20], which is used for modelling the relationship between components as a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of components, and $\mathcal{E}$ the data dependencies between components. Fig. 2 represents an example of two call graphs for an immersive application [2], which shows two users (UE1 and UE2) offloading their components to the same edge server individually. The immersive application can be split into different components. From the example illustrated in Fig. 2, there are 5 different components: Interface, Tracker, Mapper, Recognizer, and Renderer [2]. The Tracker component denotes the input camera frames with delay constraints. Such a component is used to estimate the camera position. Note that, some of these components cannot be offloaded such as the user's interface and renderer in Fig. 2, and must necessarily be evaluated locally. The components in the edge server denote the component clones for the components of mobile users.

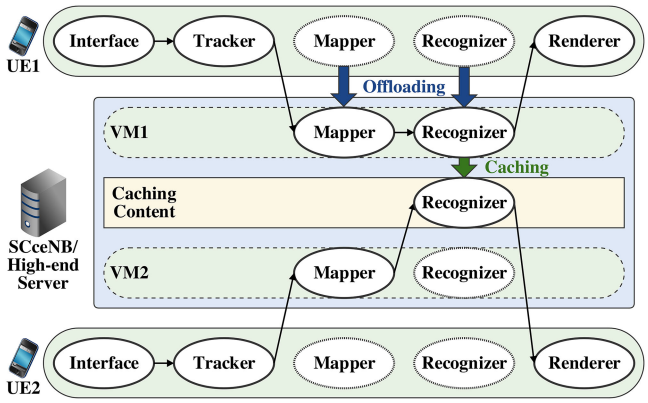


Fig. 2. Collaborative call graph with caching enhancement.

Here, we consider M UEs equipped with the same mobile device, so the local energy consumption and latency are same among the UEs when executing the same component. As stated earlier, $\mathcal{E}$ denotes the set of weight for all the edges, we assume each edge $\mathcal{E}_{u,v}$ ($\mathcal{E}_{u,v} \in \mathcal{E}$) represents the data communication (computation result) between two components. We let ϕ_v ($v \in \mathcal{V}$) denotes the weight of component v, which specifies the workload (CPU cycles) for the component v. For a given input data size $\mathcal{E}_{u,v}$, ϕ_v can be derived from [38], [39] as $\phi_v = \omega \cdot \mathcal{E}_{u,v}$, where the ω in CPU cycles/byte (cpb) [40] indicates the number of clock cycles a microprocessor will perform per byte of data processed in an algorithm. The parameter depends on the nature of the component, e.g., the complexity of the algorithm. The estimation of this value has been studied in [41], [42] which is thus beyond the scope of our work.

C. Caching Model

As stated earlier, a library consists of $|\mathcal{E}|$ computation results, denoted by $\mathcal{E} = \{\mathcal{E}_{u,v}, (u, v \in \mathcal{V})\}$, is cached at one or multiple edge servers which can be either in high-end or low-end deployment. Let $|\mathcal{E}_{u,v}|$ denotes the size (in bytes) of $\mathcal{E}_{u,v}$. The computation result's popularity distribution conditioned on

the event that user m makes an offloading request for its current component v. We also assume that each edge server has a finite-capacity storage storing part of the popular computation results. Specifically, we denote by Q_h the storage capacity of the high-end server, and by Q_l the storage capacity of the low-end SCceNB. In this work, we consider the popularity-based caching policy [19] that the edge servers store computation results based on their highest popularity until the storage capacity is achieved. Thus, offloading requests is ranked from the most popular to the least, such that the request probability for a computation result $\mathcal{E}_{u,v}$ is:

$$f(\mathcal{E}_{u,v}, \delta, |\mathcal{E}|) = \frac{1}{\mathcal{E}_{u,v}^{\delta}} \sum_{n=1}^{|\mathcal{E}|} \frac{1}{n^{\delta}} \tag{3}$$

where δ models the skewness of the popularity profile. For $\delta = 0$, the popularity profile is uniform over files, and becomes more skewed as δ grows larger [19].

D. Execution Model

As stated earlier, we focus in this paper on both low-end and high-end deployments for small cell-based mobile edge cloud. For low-end deployment, each component can be executed either on the mobile device or offloaded to a SCceNB. Whereas for high-end deployment, each component can be executed either on the mobile device or offloaded to the centralized server through a nearby SCeNB. The offloading decision is based on the workload of components $\mathcal{V}$, data communication $\mathcal{E}$, data rate $r_{n,m}^{dl}$ and $r_{n,m}^{ul}$. In this context, we consider the model of local execution, low-end execution and high-end execution, respectively.

1) Local Execution: If the component v is executed on mobile device, the completion time is $t_v^{local} = \phi_v \cdot f_m^{-1}$, and the corresponding energy consumption of the mobile device is $E_v^{local} = p_c \cdot t_v^{local}$, where f_m denotes the CPU rate of mobile devices (Million Instructions Per Second, MIPS), p_c represents the computing power for mobile device, ϕ_v denotes the number of instructions to be executed for component v.

2) Low-End Execution: If component v is offloaded to a Virtual Machine (VM) [43] in a SCceNB, the mobile device is idle before receiving the computation results. We denote $t_v^{low} = \frac{q_l \cdot \phi_v}{f_s}$ as the completion time of component v executed on the SCceNB, where f_s denotes the CPU rate of SCceNB ($f_s > f_m > 0$). Due to the limited computation capacity of SCceNBs, we assume that each SCceNB can serves at most q_l mobile users. The corresponding energy consumption of the mobile device during the remote execution is $E_v^{low} = p_i \cdot t_v^{low}$, where p_i is the idle power for mobile devices.

When component v is offloaded to SCceNB and the input data $\mathcal{E}_{u,v}$ from its previous component u is stored locally (i.e., stored in the mobile device), $\mathcal{E}_{u,v}$ must be sent to SCceNB before the execution of component v. We define this procedure as **Sending Input Data (SID)** for component v. Therefore, the time for UE m sending input data from component u to component v in low-end SCceNB n is $t\_s\_low_{n,m}^{u,v} = \frac{|\mathcal{E}_{u,v}|}{r_{n,m}^{ul}}$, and the corresponding energy consumption for UE m during the **SID** time is $E\_s\_low_{n,m}^{u,v} = p_u \cdot t\_s\_low_{n,m}^{u,v}$.

Conversely, if component v is executed locally, and its previous component u is executed in SCceNB, the output data $\mathcal{E}_{u,v}$ of component u must be sent back to mobile device before the execution of component v. We define this procedure as **Receiving Output Data (ROD)** for component v. Therefore, the delay for UE m receiving output data from component u to component v in low-end SCceNB n is $t\_r\_low_{n,m}^{u,v} = \frac{|\mathcal{E}_{u,v}|}{r_{n,m}^{dl}}$ and the corresponding energy consumption for UE m during the **ROD** time is $E\_r\_low_{n,m}^{u,v} = p_i \cdot t\_r\_low_{n,m}^{u,v}$.

3) High-end Execution: Then, we analyze the task execution for high-end deployment. When component v is offloaded to a VM in a centralized high-end server. We denote by $t_v^{high} = \frac{q_h \cdot \phi_v}{f_c}$ the completion time of component v executed on the centralized high-end server, where f_c denotes the CPU rate of the high-end server ($f_c > f_s > f_m > 0$) and q_h the maximum number of UEs that the server can serve. The corresponding energy consumption for the mobile device during the remote execution is $E_v^{high} = p_i \cdot t_v^{high}$.

Similarly, when component v is offloaded to the high-end server whereas the input data $\mathcal{E}_{u,v}$ from its previous component u is stored locally, the **SID** delay for UE m sending input data from component u to component v in high-end deployment is $t\_s\_high_m^{u,v} = \frac{|\mathcal{E}_{u,v}|}{r_{n,m}^{ul}} + t_e$, where t_e is the end-to-end delay from SCeNBs to the high-end server. The corresponding energy consumption for UE m during the **SID** time is given as follows:

$$E\_s\_high_m^{u,v} = p_u \cdot \frac{|\mathcal{E}_{u,v}|}{r_{n,m}^{ul}} + p_i \cdot t_e. \tag{4}$$

Conversely, if component v is executed locally and its previous component u is executed in the server, the **ROD** delay for component v of UE m receiving output data from component u in high-end deployment is given by

$$t\_r\_high_m^{u,v} = t_e + \frac{|\mathcal{E}_{u,v}|}{r_{n,m}^{dl}}, \tag{5}$$

and the corresponding energy consumption for UE m during the **ROD** time is $E\_r\_high_m^{u,v} = p_i \cdot t\_r\_high_m^{u,v}$.

Note that, when components u and v are processed on the same side (i.e., both processed locally or remotely), the **SID** and **ROD** delay as well as the energy consumption are equal to zero in both of the deployments. This is based on the fact that the parameters passing on the same side do not involve wireless communication, and hence the overhead is not significant. In addition, we do not consider the transmission energy consumption from SCeNB/SCceNB to UE. We only consider the idle energy consumption for mobile users when its serving SCeNB/SCceNB sending the computation results back. In this work, the latency and energy consumption for decoding at the server side are assumed to be negligible for the following reasons: (i) compared with mobile devices, low-end and high-end servers are usually equipped with more powerful processors, and (ii) the servers are usually located at fixed areas with stable power supply.

IV. Proposed Collaborative Call Graph and Problem Formulation

In this section, we first propose a concept of collaborative call graph to model the collaborative offloading and caching problem within multiple mobile users, and then formulate the optimization problems for high-end and low-end deployments, respectively.

A. Collaborative Call Graph With Caching Enhancement

We propose a concept of the collaborative call graph for multi-user MCA execution scenario, as shown in Fig. 2. When a group of UEs connect to the same edge server (high-end server or SCceNB), they can cooperate through sharing their input data and computation results in the server. For example, Fig. 2 shows that UE1 and UE2 offload their computation to the edge server and share the corresponding computation results. They can benefit from the fact that the result of component "Recognizer" is already cached in the edge server to reduce the execution latency. Or if there is no such result cached in the edge server, they can collaboratively execute the component in the edge server for one time, instead of two times separately.

B. Problem Formulation for Low-End MEC Deployment

1) Single-user Scenario: We first consider the single-user low-end deployment case, where UE m ($m \in \mathcal{M}$) offloads its components to a low-end server (i.e., SCceNB) n ($n \in \mathcal{N}_l$). In addition, we define $T^l_{n,m,v}$ as the offloading execution delay for component v which is given by the sum of **SID/ROD** period of its previous component u and its execution period. Whereas $E^l_{n,m,v}$ is the corresponding energy consumption at the mobile terminal side. In this case, the total latency $T^l_{n,m}$ and energy consumption $E^l_{n,m}$ for UE m to execute the application $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be expressed by (6) and (7) as shown at the bottom of this page, respectively, where $t\_d_{u,v}$ is the time needed by the UE m to decode the computation result (from component u to v) transmitted back by SCceNB n, and $E\_d^{u,v}_{n,m}$ denotes the corresponding energy consumption to decode the computation results transmitted back by SCceNB n.

Note that $I^l_{n,m,v}$ and $K^l_{n,v}$ are the binary indicator variables. Specifically, $I^l_{n,m,v} \in \mathcal{I}^l_{n,m}$ is the offloading decision variable, which is equal to one, if component v is processed remotely, or zero, if the component is executed locally. The UE's offloading requests are normalized such that $\sum_{n=1}^{N_l} I^l_{n,m,v} = 1$. $K^l_{n,v} \in \mathcal{K}^l_n$ is the computation results caching variable: which is equal to one, if UE m cannot find the computation results of component v cached in SCceNB n, and zero, if UE m can find the results in the corresponding SCceNB n through local caching.

Note also that a delay cost to transfer the data between UE m and SCceNB n occurs in (6) only if the two components u and v are processed at different locations, as illustrated in Table I (a).

Specifically, when $I^l_{n,m,u} = 0$ and $I^l_{n,m,v} = 1$ (i.e., the **SID** period of component v), UE m must send the results of component u to the SCceNB n. In this case, $T^l_{n,m,v}$ denotes the delay cost as shown in (6), where: (i) $t\_s\_low^{u,v}_{n,m}$, represents the **SID** delay cost for component v; and (ii) t^{low}_v, represents the delay cost when SCceNB executes component v (if no results of v cached in SCceNB).

Whereas, when $I^l_{n,m,u} = 1$ and $I^l_{n,m,v} = 0$ (i.e., the **ROD** period of component v), SCceNB n must send the computation results from component u back to UE m. In this case, $T^l_{n,m,v}$ will be equal to the sum of the following three delay costs: (i) $t\_r\_low^{u,v}_{n,m}$, the **ROD** delay for UE m to receive the results back firstly; (ii) $t\_d_{u,v}$ the latency for UE to decode the computation results from u; and (iii) t^{local}_v the delay for the UE m to process the component v locally.

On the other hand, if the two components u and v are processed at the same location, e.g, on the UE side, when $I^l_{n,m,u} = 0$ and $I^l_{n,m,v} = 0$, $T^l_{n,m,v}$ corresponds to the local execution delay t^{local}_v; or on the SCceNB side, when $I^l_{n,m,u} = 1$

$$T^l_{n,m} = \sum_{v\in\mathcal{V}} \sum_{\mathcal{E}_{u,v}\in\mathcal{E}} T^l_{n,m,v} = \underbrace{\sum_{v\in\mathcal{V}} (1 - I^l_{n,m,v}) \cdot t^{local}_v}_{\text{local execution delay}} + \underbrace{\sum_{\mathcal{E}_{u,v}\in\mathcal{E}} [\overbrace{(t\_s\_low^{u,v}_{n,m} + t^{low}_v \cdot K^l_{n,m,v}) \cdot I^l_{n,m,v}}^{\text{uplink transmission delay and SCceNB execution delay}}}_{\text{offloading overhead}}$$
$$+ \underbrace{\overbrace{(t\_r\_low^{u,v}_{n,m} + t\_d_{u,v}) \cdot I^l_{n,m,u}}^{\text{downlink transmission delay and decoding delay}} - (t\_s\_low^{u,v}_{n,m} + t\_r\_low^{u,v}_{n,m} + t\_d_{u,v}) \cdot I^l_{n,m,v} \cdot I^l_{n,m,u}]}_{\text{offloading overhead}}, \tag{6}$$

$$E^l_{n,m} = \sum_{v\in\mathcal{V}} \sum_{\mathcal{E}_{u,v}\in\mathcal{E}} E^l_{n,m,v} = \underbrace{\sum_{v\in\mathcal{V}} (1 - I^l_{n,m,v}) \cdot E^{local}_v}_{\text{local energy consumption}} + \underbrace{\sum_{\mathcal{E}_{u,v}\in\mathcal{E}} [\overbrace{(E\_s\_low^{u,v}_{n,m} + E^{low}_v \cdot K^l_{n,m,v}) \cdot I^l_{n,m,v}}^{\text{uplink transmission and idling energy consumption}}}_{\text{offloading overhead}}$$
$$+ \underbrace{\overbrace{(E\_r\_low^{u,v}_{n,m} + E\_d^{u,v}_{n,m}) \cdot I^l_{n,m,u}}^{\text{idling and local decoding energy consumption}} - (E\_s\_low^{u,v}_{n,m} + E\_r\_low^{u,v}_{n,m} + E\_d^{u,v}_{n,m}) \cdot I^l_{n,m,v} \cdot I^l_{n,m,u}]}_{\text{offloading overhead}}, \tag{7}$$

TABLE I
ALL THE POSSIBLE VALUES FOR $T^l_{n,m,v}$ AND $E^l_{n,m,v}$ IN (6) AND (7), RESPECTIVELLY

$T^l_{n,m,v}$	$I^l_{n,m,v}=0$	$I^l_{n,m,v}=1$
$I^l_{n,m,u}=0$	t^{local}_v	$t\_s\_low^{u,v}_{n,m}+t^{low}_v\cdot K^l_{n,m,v}$
$I^l_{n,m,u}=1$	$t\_r\_low^{u,v}_{n,m}+t\_d_{u,v}+t^{local}_v$	$t^{low}_v\cdot K^l_{n,m,v}$

(a) Low-end $T^l_{n,m,v}$ for different $I^l_{n,m,u}$ and $I^l_{n,m,v}$ in (6)

$E^l_{n,m,v}$	$I^l_{n,m,v}=0$	$I^l_{n,m,v}=1$
$I^l_{n,m,u}=0$	E^{local}_v	$E\_s\_low^{u,v}_{n,m}+E^{low}_v\cdot K^l_{n,m,v}$
$I^l_{n,m,u}=1$	$E\_r\_low^{u,v}_{n,m}+E\_d^{u,v}_{n,m}+E^{local}_v$	$E^{low}_v\cdot K^l_{n,m,v}$

(b) Low-end $E^l_{n,m,v}$ for different $I^l_{n,m,u}$ and $I^l_{n,m,v}$ in (7)

and $I^l_{n,m,v}=1$, $T^l_{n,m,v}$ corresponds to the remote execution delay t^{low}_v (if no results of v are cached in SCceNB).

Similarly, according to different values of $I^l_{n,m,u}$ and $I^l_{n,m,v}$, the total energy cost for component v in (7) is shown in Table I (b).

Note that the objective of data caching is to save CPU cycles for repeated computation, and thus the best data caching policy is to cache more popular computation results with higher ω (cpu cycles per byte). The optimal data caching policies $K^{l,*}_n$ ($n=1,2,...,N_l$) for the low-end (distributed caching) scenario is given as follows:

$$\begin{aligned} K^{l,*}_n = &\underset{K^l_{n,v}\in\mathcal{K}^l_n}{\operatorname{argmin}} K^l_{n,v}\cdot|\mathcal{E}_{v,w}|\cdot\omega(v),\\ \text{s.t.}\ &\sum_{v,w\in\mathcal{V}} K^l_{n,v}\cdot|\mathcal{E}_{v,w}|\le Q_l,\\ &K^l_{n,v}\in\{0,1\},\\ &n=1,2,...,N_l. \end{aligned} \tag{8}$$

In order to minimize the average delay for UEs, we formulate the problem as an optimal offloading strategy under given caching lists of the SCceNBs, which is a simple 0-1 programming problem that aims at selecting the optimal number of components to be offloaded at SCceNB n for UE m. This can be formulated as follows:

When UE m makes an offloading decision to SCceNB n for its current application $\mathcal{G}$, we define the optimal offloading decision $I\_low^*_{n,m}=\{I^l_{n,m,v}, v\in\mathcal{V}\}$ ($I\_low^*_{n,m}\in\mathcal{I}\_low^*_m$) under caching list $K^l_{n,v}$ as **Optimization Problem 1:**

$$I\_low^*_{n,m}(K^{l,*}_n,\mathcal{G})=\underset{I^l_{n,m,v}}{\operatorname{argmin}} T^l_{n,m} \tag{9}$$

$$\text{s.t. } T^l_{n,m}(I\_low^*_{n,m})\le\sum_{v\in\mathcal{V}} t^{local}_v, \tag{10}$$

$$E^l_{n,m}(I\_low^*_{n,m})\le\sum_{v\in\mathcal{V}} E^{local}_v, \tag{11}$$

$$\sum_{n=1}^{N_l} I^l_{n,m,v}=1,\ \ m=1,2,...,M,\ \ v\in\mathcal{V}, \tag{12}$$

where $\mathcal{I}\_low^*_m$ represents the set of the optimal offloading decisions between the m^{th} UE and all the available nearby SCceNBs in the network (i.e., the ones that meet the constraints). Note that constraint (10) indicates that the time it takes to execute the application through offloading action $I\_low^*_{n,m}$ must less than the local execution delay, while constraint (11) ensures that the total energy consumption for a feasible offloading action $I\_low^*_{n,m}$ is less than the total energy consumption of local execution.

2) Multi-user scenario: Then, we focus on the reduction of average latency for MCA collaborative execution within multiple users. Note that in the multi-user multi-small cell scenario, different SCceNBs have different data caching contents. Therefore, the caching policies $K^{l,*}_n$ are different if one user attach to different SCceNBs, which can change their offloading decisions. On the other hand, users' offloading request can affect the local caching content of its serving SCceNB, and thus affect the offloading decision of other users who attach to the same SCceNB. Therefore, when UEs make offloading decisions in a collaborative manner, and we can minimize the average delay for the UEs as **Optimization Problem 2:**

$$\min\ \frac{1}{M}\cdot\sum_{m=1}^{M} T^l_{n,m}\left(I\_low^*_{n,m},K^{l,*}_n\right) \tag{13}$$

$$\text{s.t. } m\in\mathcal{M}, \tag{14}$$

$$n\in\mathcal{N}_l, \tag{15}$$

$$I\_low^*_{n,m}\in\mathcal{I}\_low^*_m, \tag{16}$$

$$T^l_{n,m}(I\_low^*_{n,m})\le\sum_{v\in\mathcal{V}} t^{local}_v, \tag{17}$$

$$E^l_{n,m}(I\_low^*_{n,m})\le\sum_{v\in\mathcal{V}} E^{local}_v, \tag{18}$$

$$\sum_{n=1}^{N_l} I^l_{n,m,v}=1,\ \ m=1,2,...,M,\ \ v\in\mathcal{V}, \tag{19}$$

where (17) and (18) are the delay constraint and energy constraint for the optimal offloading decision, respectively. Equation (19) denotes each mobile user offloads its component to a single SCceNB.

C. Problem Formulation for High-End MEC Deployment

1) Single-user Scenario: Then, we consider the single-user high-end deployment case, where UE m ($m\in\mathcal{M}$) offloads its components to an aggregation point (i.e., the centralized high-end server) through its nearby SCeNB n ($n\in\mathcal{N}_h$). Similar with the low-end case, the total latency $T^h_{n,m}$ and energy consumption $E^h_{n,m}$ for UE m to execute the application can be expressed by (20) and (21) as shown at the bottom of the next page, respectively. Here $T^h_{n,m,v}$ (given by Table II (a)) is the execution delay of component v in high-end case, and $E^h_{n,m,v}$ (illustrated in Table II (b)) is the corresponding energy consumption at the

TABLE II
ALL THE POSSIBLE VALUES FOR $T^h_{n,m,v}$ AND $E^h_{n,m,v}$ IN (20) AND (21), RESPECTIVELLY

$T^h_{n,m,v}$	$I^h_{n,m,v}=0$	$I^h_{n,m,v}=1$
$I^h_{n,m,u}=0$	t_v^{local}	$t\_s\_high^{u,v}_{n,m}+t_v^{high}\cdot K^h_{m,v}$
$I^h_{n,m,u}=1$	$t\_r\_high^{u,v}_{n,m}+t\_d_{u,v}+t_v^{local}$	$t_v^{high}\cdot K^h_{m,v}$

(a) High-end $T^h_{n,m,v}$ for different $I^h_{n,m,u}$ and $I^h_{n,m,v}$ in (20)

$E^h_{n,m,v}$	$I^h_{n,m,v}=0$	$I^h_{n,m,v}=1$
$I^h_{n,m,u}=0$	E_v^{local}	$E\_s\_high^{u,v}_{n,m}+E_v^{high}\cdot K^h_{m,v}$
$I^h_{n,m,u}=1$	$E\_r\_high^{u,v}_{n,m}+E\_d^{u,v}_{n,m}+E_v^{local}$	$E_v^{high}\cdot K^h_{m,v}$

(b) High-end $E^h_{n,m,v}$ for different $I^h_{n,m,u}$ and $I^h_{n,m,v}$in (21)

mobile terminal side. $I^h_{n,m,v}$ and K^h_v are the indicator variables. Specifically, $I^h_{n,m,v}\in\mathcal{I}^h_{n,m}$ is the offloading decision variable, which is equal to one, if component v is processed remotely, or zero, if it is executed locally. Whereas $K^h_v\in\mathcal{K}^h$ is the computation results caching variable: which is equal to one, if the UE m can not find the computation results of component v shared in the high-end server (through local caching), and zero, if UE m can find the results in the server. The optimal data caching policies $K^{h,*}$ for the high-end (centralized caching) scenario is given as follows:

$$\begin{aligned} K^{h,*} &= \underset{K^h_v\in\mathcal{K}^h}{\operatorname{argmin}} K^h_v\cdot|\mathcal{E}_{v,w}|\cdot\omega(v),\\ \text{s.t.}\ &\sum_{v,w\in\mathcal{V}} K^h_v\cdot|\mathcal{E}_{v,w}|\le Q_h,\\ &K^h_v\in\{0,1\}. \end{aligned} \tag{22}$$

Different from the low-end case, each user in the high-end deployment can be served by one or multiple SCeNBs, depending on the employed transmission scheme (such as CoMP [44]). The optimal offloading decision problem for high-end deployment is thus given by **Optimization Problem 3:**

$$I\_high^*_{n,m}(K^{h,*},\mathcal{G})=\underset{I^h_{n,m,v}}{\operatorname{argmin}}\ T^h_{n,m} \tag{23}$$

$$\text{s.t.}\quad T^h_{n,m}(I\_high^*_{n,m})\le\sum_{v\in\mathcal{V}} t_v^{local}, \tag{24}$$

$$E^h_{n,m}(I\_high^*_{n,m})\le\sum_{v\in\mathcal{V}} E_v^{local}, \tag{25}$$

$$\sum_{n=1}^{N_h} I^h_{n,m,v}\ge 1,\ \ m=1,2,...,M,\ \ v\in\mathcal{V}, \tag{26}$$

where $I\_high^*_{n,m}$ denotes the optimal offloading decision between UE m and SCeNB n that can minimize the total delay for processing the application $\mathcal{G}$. (24) and (25) are the delay and energy consumption constraints for the optimal offloading decision, respectively.

2) Multi-user scenario: Note that in the high-end deployment, all the mobile users offload the components to the high-end server, which means that the users make offloading decisions according to the same caching factor $K^h_{m,v}$ in a non-cooperative manner. Each user can thus select its nearest SCeNB (i.e., the SceNB with highest $r^{dl}_{n,m}$ and $r^{ul}_{n,m}$) as its serving small cell. We can reduce the average delay through minimizing each single user's execution latency as formulated in **Optimization Problem 3**.

V. ALGORITHM DESIGN

In this section, we present our proposed optimal offloading with caching-enhancement scheme (OOCS). We give the details

$$\begin{aligned} T^h_{n,m} &= \sum_{v\in\mathcal{V}}\sum_{\mathcal{E}_{u,v}\in\mathcal{E}} T^h_{n,m,v} = \underbrace{\sum_{v\in\mathcal{V}}(1-I^h_{n,m,v})\cdot t_v^{local}}_{\text{local execution delay}} + \underbrace{\sum_{\mathcal{E}_{u,v}\in\mathcal{E}}[\overbrace{(t\_s\_high^{u,v}_{n,m}+t_v^{high}\cdot K^h_{m,v})}^{\text{uplink transmission delay and server execution delay}}\cdot I^h_{n,m,v}}_{\text{offloading overhead}}\\ &+\underbrace{\overbrace{(t\_r\_high^{u,v}_{n,m}+t\_d_{u,v})\cdot I^h_{n,m,u}}^{\text{downlink transmission delay and local decoding delay}} - (t\_s\_high^{u,v}_{n,m}+t\_r\_high^{u,v}_{n,m}+t\_d_{u,v})\cdot I^h_{n,m,v}\cdot I^h_{n,m,u}]}_{\text{offloading overhead}}, \end{aligned} \tag{20}$$

$$\begin{aligned} E^h_{n,m} &= \sum_{v\in\mathcal{V}}\sum_{\mathcal{E}_{u,v}\in\mathcal{E}} E^h_{n,m,v} = \underbrace{\sum_{v\in\mathcal{V}}(1-I^h_{n,m,v})\cdot E_v^{local}}_{\text{local energy consumption}} + \underbrace{\sum_{\mathcal{E}_{u,v}\in\mathcal{E}}[\overbrace{(E\_s\_high^{u,v}_{n,m}+E_v^{high}\cdot K^h_{m,v})}^{\text{uplink transmission and idling energy consumption}}\cdot I^h_{n,m,v}}_{\text{offloading overhead}}\\ &+\underbrace{\overbrace{(E\_r\_high^{u,v}_{n,m}+E\_d^{u,v}_{n,m})\cdot I^h_{n,m,u}}^{\text{idling and decoding energy consumption}} -(E\_s\_high^{u,v}_{n,m}+E\_r\_high^{u,v}_{n,m}+E\_d^{u,v}_{n,m})\cdot I^h_{n,m,v}\cdot I^h_{n,m,u}]}_{\text{offloading overhead}}, \end{aligned} \tag{21}$$

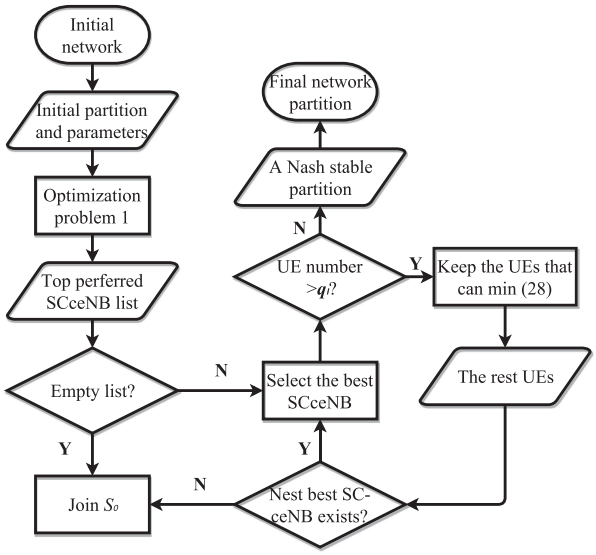


Fig. 3. The flowchart of *Algorithm 1*.

how the optimal offloading decisions for all users in the network are made by computing first a list of the optimal number of components to be offloaded to nearby small cells for each user (using **Optimization Problems 1** and **3**, respectively). Then we define the optimal partitions of users (or coalitions) for low-end deployment. In order to identify which SCceNB can serve the attached users and execute the offloading decisions, we explore the concept of coalition formation game [45], [46] to solve **Optimization Problem 2**.

Algorithm 1: Optimal Network Partition for Multi-user Low-end Deployment.

Initial network:
initial network partition for the UEs: $\{\{\emptyset\}, \{1\}, \{2\}, ..., \{M\}\}$.
Step 1: Component Offloading Decision
UEs work in a Non-cooperative manner
Input of Step 1: Parameters M, N_l, $K^l_{m,n,v}$, Q_l, t_v^{local}, $\mathcal{G}$, $t\_s\_low_{n,m}$, t_v^{low}, $t\_r\_low^{u,v}_{n,m}$, $t\_d_{u,v}$.
1) Each UE builds a top preferred SCceNB list $I\_low^*_{n,m}$ according to **Optimization Problem 1**.
2) Each UE selects its best preferred SCceNB as its serving SCceNB and submit its offloading requests.
3) For the UEs whose list is empty, they join S_0.
Output of Step 1: $I\_low^*_{n,m}$.
Step 2: Coalition Formation
UEs work in a cooperative manner
Input of Step 2: Parameters $I\_low^*_{n,m}$, q_l.
4) Each SCceNB receives the requests. Due to the limited computation capacity of SCceNBs, each SCceNB keeps the top q_l UEs that can minimize (28), and reject the rest.
5) The rejected UEs will re-apply to their next best SCceNB of their list $I\_low^*_{n,m}$, and each SCceNB updates its serving UEs list.
6) Repeat 5), until convergence to a final Nash-stable partition π^*. For the UEs who cannot be allocated to a SCceNB, they execute the application locally and join S_0.
Output of Step 2: π^*

A. Game Formulation

Indeed, in the low-end deployment case, the attachment of mobile users to a particular SCceNB can be seen as a coalition formation game in partition form with transferable utility. Specifically, let M UEs be players, and π be the set of existing users in the network. We assume that UEs in each coalition connect to a single SCceNB and form a coalition. Let S_n denote the set of UEs that are served by SCceNB n ($n \in \mathcal{N}_l$), and S_0 denote the set of UEs that execute the application locally, i.e. without offloading.

The payoff of UE m in the coalition S_n (in terms of execution latency) can be expressed as follows:

$$V_m(S_n, \pi) = \begin{cases} T^l_{n,m}, & n \neq 0, \\ \sum_{v \in \mathcal{V}} t_v^{local}, & n = 0. \end{cases} \tag{27}$$

We define $V_{S_n}(\pi)$, as the sum of the payoff of all players within the coalition, i.e. total execution delay of all UEs in the coalition S_n, and is given by:

$$V_{S_n}(\pi) = \sum_{m \subseteq S_n} V_m(S_n, \pi). \tag{28}$$

Based on this, the optimal network partition (coalition formation) for multi-user low-end deployment is given by *Algorithm 1*. The corresponding flowchart is shown in Fig. 3. Note that the UEs merge through SCceNB keeping the top q_l UEs that can minimize (28). The rejected users will re-send their location and component information to the next SCceNB of their top preferred lists. Each SCceNB updates its list of serving users, and then repeats the previous process of coalition formation until all UEs are allocated to their nearby SCceNB. Note that user m has an incentive to move from its current coalition S_a to another coalition S_b in π^* if the following split rule in (29) is satisfied.

$$v(S_a \backslash \{m\}) + v(S_b \bigcup \{m\}) < v(S_a) + v(S_b), \tag{29}$$

where $v(S_a)$ denotes the total payoff generated by a coalition S_a.

It is worth noting that UEs who can not be allocated to a SCceNB will execute their application locally and join the coalition S_0. The final network partitions will be thus given as $\pi^* = \{S_0, S_1, ..., S_{N_l}\}$.

Proposition V.1: Starting by any initial partition π_{ini} from step 4) in Algorithm 1, the coalition formation for transfers is guaranteed to converge to a final stable partition π^*.

Proof: See Appendix A. ■

B. Algorithm Description

To sum up, our algorithm consists of three main phases: (i) Initial phase, selection of possible offloaded components, along with the best preferred SCeNB for each single UE in

high-end deployment case, or a list of top preferred SCceNBs for each single UE in the low-end deployment case. In the latter case, the optimal network partition shown in *Algorithm 1* is executed, (ii) Cache placement phase, where we determine the cache results at each edge server. Specifically, in the high-end deployment case, the central high-end server make caching decision through collecting the offloading requests of all the UEs around the network. Whereas in the low-end deployment case, each SCceNB caches the results according to its nearby UEs' offloading requests in a distributed manner, and (iii) Delivery phase, which allows the edge servers to deliver the requested results to the UEs over wireless channels. In what follows, we describe these three phases.

First of all, each UE observes the current network states to identify the accessible nearby small cells over a control channel, and computes the optimal number of components to be offloaded at each identified nearby small cell by resolving **Optimization Problem 1** (low-end) or **Optimization Problem 3** (high-end). A top preferred SCeNB (high-end case) or a list of top preferred SCceNBs (low-end case) will be thus created for every user according to the server's previous caching content $K^{l,*}_{n,m,v}$ or $K^{h,*}_{m,v}$. The list is sorted in ascending order according to the total latency.

In the high-end case, each user attaches to their best preferred SCeNB and submit its location and offloading requests to the high-end server. The server makes current optimal cache placement decision $K^{h,*}_{m,v}$ (as shown in (22)) after the execution, according to its previous caching content and the current offloading requests. Whereas in the low-end case, each user first selects its best preferred SCceNB (the one from its list with minimum delay) as its serving SCceNB and upload the required information to the SCceNB. This information relates to both the offloading data as well as the mobile device and network characteristics. The information related to offloading data include the: i) application type, ii) current components that the user is executing, iii) buffer size (i.e., input data size for offloading) and iv) workload (CPU cycles) for the current components. Whereas, the mobile device and network related information contain the mobile transmission rate and processing power as well as the channel quality that they experience with respect to the SCceNB. Then, the optimal network partition is derived using *Algorithm 1*. After each user attaching to the optimal small cell, the servers deliver the requested results to the UEs over wireless channels. At last, each SCceNB makes the optimal cache placement decision $K^{l,*}_{n}$ (as shown in (8)) according to its serving users' offloading requests and previous caching content.

C. Complexity Analysis

The optimal caching placement problems formulated in (8) and (22) are the simple 0-1 Knapsack problems [47]. The problem is NP-hard [48], but the suboptimal solutions can be obtained in pseudo-polynomial time by dynamic programming approach [49]. Thus, the complexities of our suboptimal caching placement algorithms are $\mathcal{O}(|\mathcal{E}_{v,w}|Q_l)$ for the low-end case and $\mathcal{O}(|\mathcal{E}_{v,w}|Q_h)$ for the high-end case. Note that both problems are solved at the server side after the execution of MCAs. Thus, the overheads (delay) for solving both optimization problems have no impact on the execution latency in (6) and (20). In addition, Optimization Problem 1 and Optimization Problem 3 are 0-1 programming problems that can be solved through the branch and bound algorithm. Their corresponding complexities are both $\mathcal{O}(2^{(|\mathcal{V}|)})$ in the worst case, but can be reduced through pruning schemes, such as alpha-beta pruning [50]. Optimization Problem 2, on the other hand, can be viewed as a college admission game [51]. The complexity for the solution is $\mathcal{O}(N_l)$ in the worst case.

TABLE III
NETWORK PARAMETERS

Parameter	Value	Parameter	Value
λ^l_s, λ^h_s	10^{-3}	λ_u	5×10^{-3}
g_{ul}	10^{-3}	g_{dl}	10^{-3}
r	100 m	t_d	0.2 s
p_u	0.01 W	p_s	0.1 W
p_c	0.9 W	p_i	0.3 W
f_m	10^9 cyclse/s	f_s	10^{10} cyclse/s
f_c	10^{11} cyclse/s	t_e	0.02 s
Q_h	4 Gbits	Q_l	2 Gbits
q_h	50	q_l	6
N_0	5×10^{-5}	β	-2

VI. PERFORMANCE EVALUATION

In this section, we evaluate the efficiency of our proposal for both low-end and high-end deployments under single-user and multi-user scenarios. We assume that the geographical distributions of both the small cells and UEs follow an independent Homogeneous Poisson Point Process (HPPP) [52], which is suitable for describing the distribution of small devices. The geographical distributions of both small cells and UEs are thus given by:

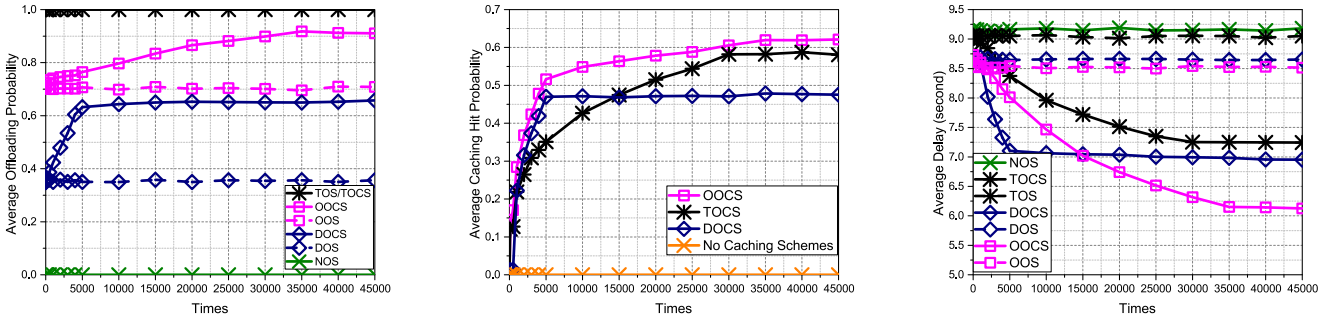
$$P(n,r) = \frac{[\lambda L(r)]^n}{n!} e^{-\lambda L(r)} \tag{30}$$

where $L(r) = r^2$ denote the area of the network of radius r. λ is the mean density (intensity) of points. We define λ_u, λ^l_s and λ^h_s as the density of UEs, SCceNBs and SCeNBs, respectively.

For the computing components, we assume that each application consists five random components as shown in Fig. 2, both data dependencies $\mathcal{E}_{u,v}$ and required number of CPU cycles per byte (cpb) for components ω follow the uniform distribution with $\mathcal{E}_{u,v} \in [100, 500]$ KB and $\omega \in [4000, 12000]$ cpb as in [38]. All random variables are independent for different components. The size of computation results library $|\mathcal{E}_{u,v}| = 1000000$.

We compare the benefits of our optimal offloading with caching-enhancement scheme (OOCS) with respect to six benchmark policies, namely:

- *No Offloading Scheme (NOS):*
  Local execution, which means that applications are executed on smartphones, by letting offloading decision variables $\mathcal{I}^l_{n,m}$ and $\mathcal{I}^h_{n,m}$ equal to 0 in (6) and (20), respectively.



(a) Average offloading probability vs. execution times

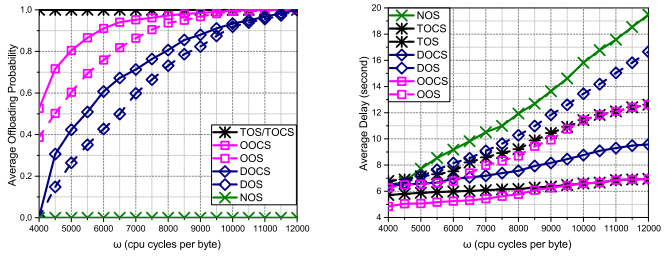
(b) Average caching hit probability vs. execution times

(c) Average application execution delay vs. execution times

Fig. 4. High-end performance vs. application execution times, $\omega = 5900$ cpb, $\delta = 1$.

- *Optimal Offloading Without Caching Scheme (OOS):* Traditional offloading strategies [15]–[17] without caching enhancement, i.e., let caching decision variables $\mathcal{K}^l_{n,m}$ and $\mathcal{K}^h$ equal to 0 in (6) and (20), respectively.
- *Total Offloading Without Caching Scheme (TOS):* Coarse offloading strategies [13], [22] without caching enhancement, which means that offload all the components to the server side, i.e., let offloading decision variables $\mathcal{I}^l_{n,m}$ and $\mathcal{I}^h_{n,m}$ equal to 1 in (6) and (20), respectively.
- *Total Offloading and Caching Scheme (TOCS):* Coarse offloading with caching enhancement.
- *D2D Offloading Without Caching Scheme (DOS):* D2D offloading strategies [12] without caching enhancement, which means resource-poor devices can utilize other users' vacant computing resources.
- *D2D Offloading and Caching Scheme (DOCS):* D2D offloading strategies with D2D caching [6], [27], [28] enhancement.

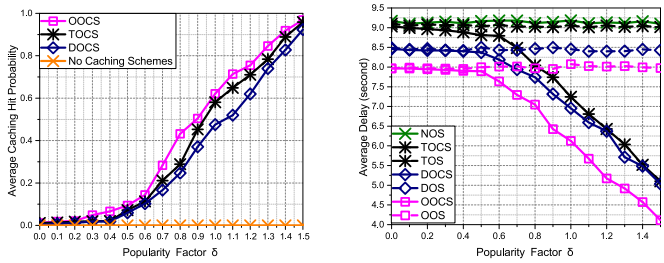
Note that in the D2D offloading scenario, we assume that the offloadee mobile devices [12] (servers) are equipped with more powerful processors than the offloader mobile devices (clients), the cpu rate of offloadee devices $f_{ms} = 1.5 \times 10^9$ cpu cycles per second. Each offloadee device allocates 0.5 Gbits of its memory to D2D caching. The parameters used in our simulations are reported in Table III.



(a) Average offloading probability vs. ω

(b) Average execution delay vs. ω

Fig. 5. High-end performance vs. ω (the required number of cpu cycles per byte for the components), $\delta = 1$.



(a) Average caching hit probability vs. δ

(b) Average execution delay vs. δ

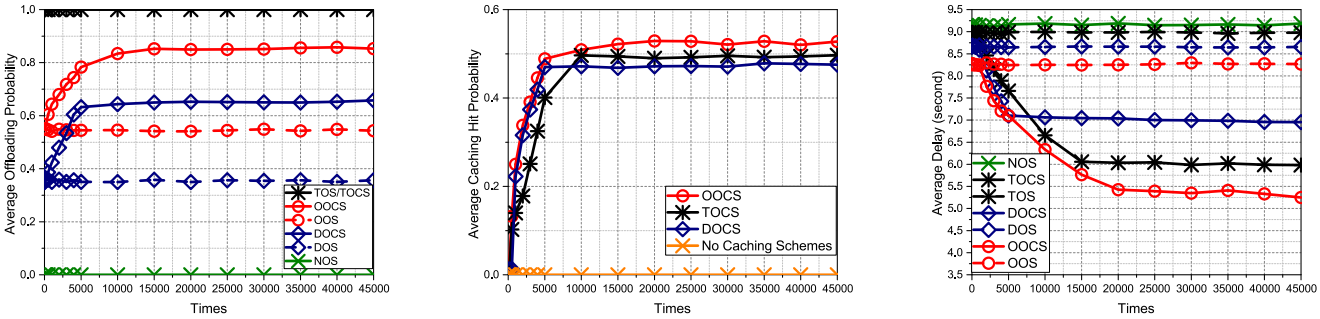
Fig. 6. High-end performance vs. popularity factor δ, $\omega = 5900$ cpb.

A. Performance Evaluation of OOCS in Single-User Scenario

We first illustrate the performance of our proposed OOCS scheme for the single-user scenario. This is done by minimizing the execution delay for a single user through **Optimization Problem 1** and **Optimization Problem 3**.

To do so, we generate 45000 independent applications and run the applications continuously. Let the required number of cpu cycles per byte for the components $\omega = 5900$ cpb, which corresponds to the workload of processing the English main page of Wikipedia [39]. Assume that offloading requests are modeled as the Zipf distribution and the popularity factor $\delta = 1$ (shown in (3)). The simulation results are averaged over 2000 Monte-Carlo topology realizations.

1) High-end deployment scenario: Fig. 4(a), 4(b), 4(c) show the curves of average offloading probability, average caching hit probability and average application execution delay versus application execution times, respectively. Several observations can be made. First of all, for all the non-caching schemes (TOS, NOS, DOS, OOS), execution times have no influence on the average offloading probability, average caching hit probability and the average execution delay. The reason is that without caching enhancements, the users make offloading decision independently at the beginning of each execution time. Secondly, for the schemes allowing caching enhancement (OOCS, DOCS), the average offloading probability and average caching hit probability increase considerable with the execution time. We observe a peak after the servers' storage capacities are achieved. Specifically, the storage capacity of high-end server is achieved after about 35000 times of execution and storage capacity of offloadee is achieved after about 5000 times of execution. This suggests that as the execution time grows, more popular computation results are cached in the server. Thus, the user prones to

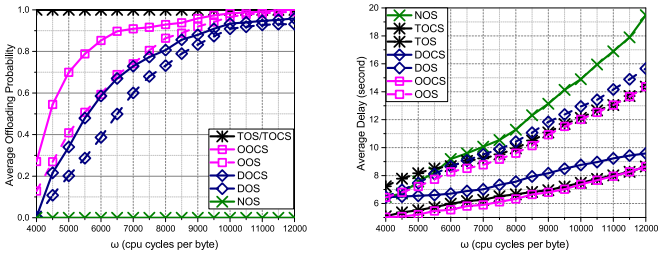


(a) Average offloading probability vs. execution times

(b) Average caching hit probability vs. execution times

(c) Average application execution delay vs. execution times

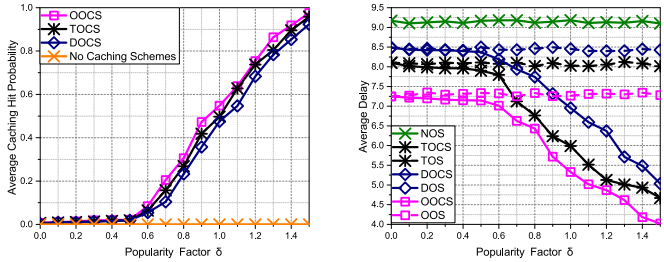
Fig. 7. Low-end performance vs. application execution times, $\omega = 5900$ cpb, $\delta = 1$.



(a) Average offloading probability vs. ω

(b) Average execution delay vs. ω

Fig. 8. Low-end performance vs. ω (required number of cpu cycles per byte for the components), $\delta = 1$.



(a) Average caching hit probability vs. δ

(b) Average execution delay vs. δ

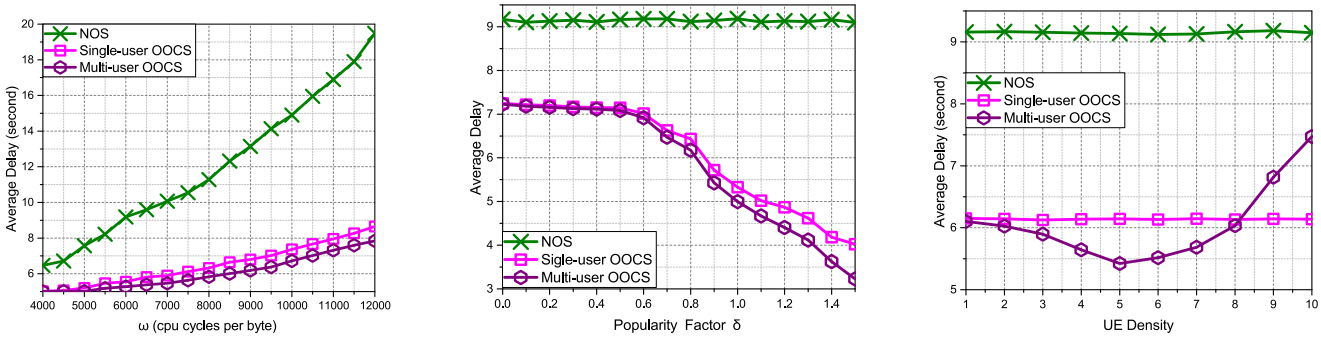
Fig. 9. Low-end performance vs. popularity factor δ, $\omega = 5900$ cpb.

offload the components to the server to reduce the execution delay. Last but not least, it can be observed from Fig. 4 (c) that after the high-end server's storage capacity is achieved, our high-end OOCS algorithm can reduces 28.04%, 33.28%, 32.31%, 15.42%, 29.19% and 11.89% execution delay compared with OOS, NOS, TOS, TOCS, DOS and DOCS, respectively.

Figs. 5 and 6 plot the high-end performance versus ω (required cpb for the components) and the popularity factor δ, respectively. It can be seen from the figures that ω has tremendous influence on the average offloading probability (as shown in Fig. 5 (a)), whereas δ plays a key role in the average caching hit probability (as shown in Fig. 6 (a)). It is evident that the local execution overhead increases as ω grows. Thus, the mobile user prefers to offload the components rather than local execution. Moreover, higher ω and δ can bring larger reduction of execution delay (as shown in Figs. 5 (b) and 6 (b)), and our OOCS algorithm still performs better than the other schemes in the delay reduction.

2) Low-end deployment scenario: Similar results are given in Figs. 7, 8 and 9 for low-end deployment scenario. Specifically, Fig. 7 reports the low-end performance versus the application execution times. Note that the storage capacity of SCceNB is achieved after about 20000 times of execution. Figs. 8 and 9 plot the low-end performance versus ω and δ, respectively. From the figures, we can see clearly that ω and δ still play key roles in the average offloading probability and average caching hit probability, respectively. Note that the average offloading probability of OOCS scheme increases considerable from 28% to 97%, when ω grows from 4000 to 9000, as shown in Fig. 8 (a). When $\omega > 9000$, the average offloading probability of our OOCS will be more than 99%. This suggests that the performance of our OOCS is similar with the performance of total offloading schemes (TOS and TOCS) when $\omega > 9000$. The situation is illustrated by Fig. 8 (b). On the other hand, when δ grows from 0 to 0.5, there was hardly any change in the value of average caching hit probability. This suggests that when $\delta < 0.5$, the schemes with caching enhancements (TOCS, OOCS, DOCS) performs similarly with respect to their non-caching schemes (TOS, OOS, DOS). The trend is given by Fig. 9 (b). In addition, it can be observed from Fig. 7 (c) that after SCceNB's storage capacity achieved, our low-end OOCS algorithm can reduce 36.49%, 42.83%, 41.51%, 12.25%, 39.33% and 24.50% execution delay compared to OOS, NOS, TOS, TOCS, DOS and DOCS, respectively.

3) High-end/Low-end performance comparison: As stated earlier, the high-end and low-end scenarios perform quasi similarly. However, there are still several differences between the two scenarios in the following aspects: (i) The high-end deployment performs better in the average offloading probability and average caching hit probability, whereas the low-end deployment performs better in the average delay reduction. The reason is that the high-end server is equipped with more powerful processors and larger storage sizes (i.e. larger caching list) than the SCceNB, and thus mobile users have more incentive to offload the components to the high-end server. On the other hand, the high-end deployment performs worse in delay reduction due to the large end-to-end delay from SCeNBs to the high-end server. (ii) The performance thresholds between schemes are different in the two deployments. For example, as shown in Figs. 5 (a) and



(a) Average application execution delay vs. ω

(b) Average application execution delay vs. δ

(c) Average application execution delay vs. UE density λ_u (10^{-3})

Fig. 10. Low-end (Femto-cloud) performance of multi-user scenario.

8 (a), the CPB thresholds between OOCS and TOCS are 7500 and 9000 for high-end and low-end deployments, respectively. In addition, the popularity factors thresholds between OOCS and OOS are 0.4 and 0.5 for high-end and low-end deployments, respectively (as shown in Figs. 6 (a) and 9 (a)).

B. Performance Evaluation of OOCS in Multi-User Scenario

We then illustrate the performance of our OOCS scheme for the multi-user scenario. This is done by minimizing the average execution delay for multiple users through **Optimization Problem 3**. Fig. 10 show the multi-user delay performance versus ω, δ and UE density λ_u, respectively. Note that our multi-user OOCS performs better in delay reduction when ω grows as shown in Fig. 10 (a). The reason is that the offloading probability increases with ω since more users can reduce their delay through joining our coalition game performed in OOCS. Similarly, as δ grows, more users can find their required results cached in their serving SCceNB, and thus reducing their execution delay, as shown in Fig. 10 (b). Finally, Fig. 10 (c) shows that our multi-user OOCS performs better as UE density grows from 10^{-3} to 8×10^{-3}. A peak is observed when $\lambda_u = 5 \times 10^{-3}$, which corresponds to 11.71% and 40.61% delay reduction, compared to single-user OOCS and NOS, respectively. When $\lambda_u > 8 \times 10^{-3}$, single-user OOCS performs better. The reason is that, when λ_u grows larger ($> 5 \times 10^{-3}$), the SCceNBs cannot afford such many UEs (we assume that each SCceNB can handle 6 UEs in our simulations), and thus a large number of UEs will be rejected and run the application locally. As a result, the average delay increases and will be close to the local execution delay.

VII. Conclusion

In this paper, we have studied the computation offloading problem with caching enhancement for mobile edge cloud networks. We propose an optimal offloading with caching-enhancement scheme (OOCS) to minimize the execution delay for mobile users in two kinds of edge cloud scenarios. First, we have proposed a concept of cooperative call graph to formulate the offloading and caching relationships within multiple components. Then, in order to minimize the overall delay for multi-user femto-cloud networks, we have formulated the problem of users' allocation as a coalition formation game. We have proved the existence of convergence. Compared to six alternative solutions in literature, our proposed approach achieves the best performance. Specifically, our high-end OOCS algorithm can reduces 28.04%, 33.28%, 32.31%, 15.42%, 29.19% and 11.89% execution delay compared with OOS, NOS, TOS, TOCS, DOS and DOCS, respectively. Whereas our low-end OOCS algorithm can reduce 36.49%, 42.83%, 41.51%, 12.25%, 39.33% and 24.50% execution delay compared to OOS, NOS, TOS, TOCS, DOS and DOCS, respectively. Moreover, when consider the multiuser scenario, our multiuser OOCS can reduce 11.71% delay comparing with the single-user OOCS.

Appendix A
Proof of the Proposition V.1

Proof: For a partition Π_{ini} from step 4) in *Algorithm 1*, the coalition formation process can be seen as a sequence of transfer operations that transform the network's partition,

$$\Pi_l = \Pi_{ini} \rightarrow \Pi_1 \rightarrow \Pi_2 \rightarrow ..., \tag{31}$$

where $\Pi_l = \{S_0, S_1 ..., S_{N_l}\}$ is a partition which composed of at most $N_l + 1$ coalitions (N_l coalitions forms at N_l given SCceNBs and one local execution coalition S_0) that is formed after l transfers. User m has an incentive to move from its current coalition S_a to another coalition S_b in π^* if

$$v(S_a \backslash \{m\}) + v(S_b \bigcup \{m\}) < v(S_a) + v(S_b). \tag{32}$$

As a transfer between two SCceNBs a and b in a partition in a partition Π_l, does not affect the total utility generated by the other coalitions in $\Pi_l \setminus \{S_a, S_b\}$ (since UEs use orthogonal channels, we do not consider intra-cell interference in this work), every transfer $\Pi_l \rightarrow \Pi_k$ forms an order such that

$$\Pi_l \rightarrow \Pi_k \Leftrightarrow \sum_{S_m \in \Pi_k} v(S_m) < \sum_{S_m \in \Pi_l} v(S_p) \tag{33}$$

which is transitive and irreflexive. Therefore, for any two partitions, we have $\Pi_l \neq \Pi_k$, $l \neq k$. As the number of partitions of a set is finite and equal to the Bell number, the sequence in (31) is guaranteed to converge to a final partition π^*.

■

Acknowledgment

The authors would like to thank the anonymous reviewers for their comments.

REFERENCES

[1] S. Yu, R. Langar, W. Li, and X. Chen, "Coalition-based energy efficient offloading strategy for immersive collaborative applications in femto-cloud," in *Proc. IEEE Int. Conf. Commun.*, Kuala Lumpur, Malaysia, 2016, pp. 1–6.

[2] T. Verbelen, P. Simoens, F. D. Turck, and B.Dhoedt,"Leveraging cloudlets for immersive collaborative applications," *IEEE Pervasive Comput.*, vol. 12, no. 4, pp. 30–38, Oct. 2013.

[3] T. Langlotz, D. Wagner, A. Mulloni, and D. Schmalstieg, "Online creation of panoramic augmented reality annotations on mobile phones," *IEEE Pervasive Comput.*, vol. 11, no. 2, pp. 56–63, Feb. 2012.

[4] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: An efficient code partition algorithm for mobile cloud computing," in *Proc. IEEE 1st Int. Conf. Cloud Netw.*, Paris, France, 2012, pp. 80–86.

[5] M. Maier, M. Chowdhury, B. P. Rimal, and D. P. Van, "The tactile internet: Vision, recent progress, and open challenges," *IEEE Commun. Mag.*, vol. 54, no. 5, pp. 138–145, May 2016.

[6] L. Wang, H. Wu, Y. Ding, W. Chen, and H. V. Poor, "Hypergraph-based wireless distributed storage optimization for cellular d2d underlays," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 10, pp. 2650–2666, Oct. 2016.

[7] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, Sep. 2013.

[8] P. Gupta, *Mobile Edge Computing (MEC)—A Step Towards 5G*. [Online]. Available: http://flarrio.com/mobile-edge-computing-mec-a-step-towards-5g/

[9] *Distributed Computing, Storage and Radio Resource Allocation Over Cooperative Femtocells (Tropic*). [Online]. Available: http://www.ict-tropic.eu

[10] ETSI, *Mobile Edge Computing (MEC)*. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing

[11] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tut.*, vol. 19, no. 4, pp. 2322–2358, Aug. 2017.

[12] L. Pu, X. Chen, J. Xu, and X. Fu, "D2d fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted d2d collaboration," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3887–3901, Nov. 2016.

[13] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-efficient mobile cloud computing under stochastic wireless channel," *IEEE Wireless Commun.*, vol. 12, no. 9, pp. 4569–4581, Sep. 2013.

[14] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative task execution in mobile cloud computing under a stochastic wireless channel," *IEEE Wireless Commun.*, vol. 14, no. 1, pp. 81–93, Jan. 2015.

[15] S. Barbarossa, S. Sardellitti, and P. D. Lorenzo, "Communicating while computing: Distributed cloud computing over 5g heterogeneous networks," *IEEE Signal Process. Mag.*, vol. 31, no. 6, pp. 45–55, Nov. 2014.

[16] S. Sardellitti, S. Barbarossa, and G. Scutari, "Distributed mobile cloud computing: Joint optimization of radio and computational resources," in *Proc. IEEE Globecom*, Austin, TX, USA, 2014, pp. 1505–1510.

[17] P. D. Lorenzo, S. Barbarossa, and S. Sardellitti, "Joint optimization of radio resources and code partitioning in mobile cloud computing," unpublished paper, 2013. [Online]. Available: https://arxiv.org/abs/1307.3835v1

[18] 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.

[19] S. Tamoor-ul-Hassan, M. Bennis, P. H. J. Nardelli, and M. Latva-aho, "Caching in wireless small cell networks: A storage-bandwidth tradeoff," *IEEE Wireless Commun. Lett.*, vol. 20, no. 6, pp. 1175–1178, Mar. 2016.

[20] B. G. Ryder, "Constructing the call graph of a program," *IEEE Trans. Softw. Eng.*, vol. SE-5, no. 3, pp. 216–226, May 1979.

[21] X. D. Foy, *Mobile Edge Computing: Putting the Network on the Path to 5G*. [Online]. Available: http://the-mobile-network.com/2015/09/mobile-edge-computing-putting-the -network-on-the-path-to-5g/

[22] E. Cuervo *et al.*, "Maui: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst., Appl., Services*, New York, NY, USA, 2010, pp. 49–62.

[23] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proc. EuroSys*, Salzburg, Austria, 2011, pp. 301–314.

[24] R. Kemp, N. Palmer, T. Kielmann, and H. E. Bal, "Cuckoo: A computation offloading framework for smartphones," in *Proc. 2nd Int. Conf. Mobile Comput., Appl., Services, Santa Clara, CA, USA*, 2010, pp. 59–79.

[25] K. Sokol, A. Andrius, H. Pan, M. Richard, and Z. Xinwen, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE Int. Conf. Comput. Commun.*, Orlando, FL, USA, 2012, pp. 945–953.

[26] C. Shi, K. Habak, P. Pandurangan, M. H. Ammar, M. Naik, and E. W. Zegura, "COSMOS: Computation offloading as a service for mobile devices," in *Proc. ACM 15th Int. Symp. Mobile Ad Hoc Netw. Comput.*, Philadelphia, PA, USA, 2014, pp. 287–296.

[27] L. Wang, H. Wu, W. Wang, and K.-C. Chen, "Socially enabled wireless networks: Resource allocation via bipartite graph matching," *IEEE Commun. Mag.*, vol. 53, no. 10, pp. 128–135, Oct. 2015.

[28] B. Bai, L. Wang, Z. Han, W. Chen, and T. Svensson, "Caching based socially-aware d2d communications in wireless content delivery networks: A hypergraph framework," *IEEE Wireless Commun.*, vol. 23, no. 4, pp. 74–81, Aug. 2016.

[29] F. Chi, X. Wang, W. Cai, and V. Leung, "Ad-hoc cloudlet based cooperative cloud gaming," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 625–639, Jul./Sept. 2018.

[30] S. Yu, B. Dab, Z. Movahedi, R. Langar, and L. Wang, "A socially-aware hybrid computation offloading framework for mobile edge computing," submitted for publication.

[31] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Commun. Lett.*, vol. 6, no. 3, pp. 398–401, Jun. 2017.

[32] G. Merlino, S. Arkoulis, S. Distefano, C. Papagianni, A. Puliafito, and S. Papavassiliou, "Mobile crowdsensing as a service," *Future Gener. Comput. Syst.*, vol. 56, pp. 623–639, Mar. 2016. [Online]. Available: https://doi.org/10.1016/j.future.2015.09.017

[33] X. Zhang *et al.*, "Incentives for mobile crowd sensing: A survey," *IEEE Commun. Surv. Tut.*, vol. 18, no. 1, pp. 54–67, Mar. 2015.

[34] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: Current state and future challenges," *IEEE Commun. Mag.*, vol. 49, no. 11, pp. 32–39, Nov. 2011.

[35] S. Wang, Y.-C. Wu, and M. Xia, "Achieving global optimality for wirelessly-powered multi-antenna TWRC with lattice codes," in *Proc. IEEE Int. Conf. Acoustics, Speech Signal Process.*, Shanghai, China, 2016, pp. 3556–3560.

[36] S. Wang, M. Xia, and Y.-C. Wu, "Multipair two-way relay network with harvest-then-transmit users: Resolving pairwise uplink-downlink coupling," *IEEE J. Sel. Topics Signal Process.*, vol. 10, no. 8, pp. 1506–1521, Dec. 2016.

[37] A. Goldsmith, *Wireless Communications*. Cambridge, U.K.: Cambridge University Press, 2005.

[38] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Dec. 2016.

[39] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Sep. 2016.

[40] I. K. Center, *Processor Allocation*. [Online]. Available: http://www.ibm.com/support/knowledgecenter/POWER6/iphb1/iphb1_vios_planning_sea_procs.htm

[41] W. H. Yuan and K. Nahrstedt, "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems," in *Proc. 19th ACM Symp. Operating Syst. Princ.*, New York, NY, USA, Oct. 2003, pp. 149–163.

[42] M. Yang, Y. Wen, J. Cai, and C. H. Foh, "Energy minimization via dynamic voltage scaling for real-time video encoding on mobile devices," in *Proc. IEEE Int. Conf. Commun.*, Ottawa, ON, Canada, 2012, pp. 2026–2031.

[43] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[44] S. Fu, B. Wu, H. Wen, P.-H. Ho, and G. Feng, "Transmission scheduling and game theoretical power allocation for interference coordination in comp," *IEEE Trans. Wireless Commun.*, vol. 50, no. 11, pp. 44–50, Nov. 2012.

[45] Y. Li, D. Jin, J. Yuan, and Z. Han, "Coalitional games for resource allocation in the device-to-device uplink underlaying cellular networks," *IEEE Trans. Wireless Commun.*, vol. 13, no. 1, pp. 112–123, Jan. 2014.

[46] R. Langar, S. Secci, R. Boutaba, and G. Pujolle, "An operations research game approach for resource and power allocation in cooperative femtocell networks," *IEEE Trans. Mobile Comput.*, vol. 14, no. 4, pp. 675–687, Apr. 2015.

[47] P. Chu and J. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *J. Heuristics*, vol. 4, no. 1, pp. 63–86, Jun. 1998.

[48] S. Martello, D. Pisinger, and P. Toth, "Dynamic programming and strong bounds for the 0-1 knapsack problem," *Manage. Sci.*, vol. 45, no. 3, pp. 414–424, Mar. 1999. [Online]. Available: http://dx.doi.org/10.1287/mnsc.45.3.414
[49] C. H. Papadimitriou, "On the complexity of integer programming," *J. ACM*, vol. 28, no. 4, pp. 765–768, Oct. 1981. [Online]. Available: http://doi.acm.org/10.1145/322276.322287
[50] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artif. Intell.*, vol. 6, no. 4, pp. 293–326, 1975.
[51] W. Saad, Z. Han, R. Zheng, M. Debbah, and H. V. Poor, "A college admissions game for uplink user association in wireless small cell networks," in *Proc. IEEE Int. Conf. Comput. Commun.*, Toronto, ON, Canada, Apr. 2014, pp. 1096–1104.
[52] S. Lee, R. Zhang, and K. Huang, "Opportunistic wireless energy harvesting in cognitive radio networks," *IEEE Trans. Wireless Commun.*, vol. 12, no. 9, pp. 4788–4799, Sep. 2013.

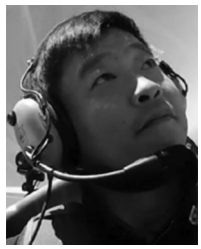
Shuai Yu (S'06–M'08) received B.S. degree from Nanjing University of Post and Telecommunications (NJUPT), Nanjing, China, in 2009, the M.S. degree from Beijing University of Post and Telecommunications (BUPT), Beijing, China, in 2014, and the Ph.D. degree from University Pierre and Marie Curie (now Sorbonne Université), Paris, France, in 2018. He is now a Postdoctoral Research Fellow with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His research interests include wireless communications, mobile computing and machine learning.

Rami Langar received the M.Sc. degree in network and computer science from University Pierre and Marie Curie, Paris, France, in 2002 and the Ph.D degree in network and computer science from Telecom ParisTech, Paris, France, in 2006. He was an Associate Professor with Laboratoire d'Informatique de Paris 6 (LIP6), University Pierre and Marie Curie (now Sorbonne Université) between 2008 and 2016, and a Postdoctoral Research Fellow with the School of Computer Science, University of Waterloo, Waterloo, ON, Canada, between 2006 and 2008. He is currently a Full Professor with University of Paris Est Marne-la-Vallée (UPEM), France. He is involved in many European and National French research projects, such as MobileCloud (FP7), GOLDFISH (FP7), ANR ABCD, FUI PODIUM, FUI ELASTIC, FUI SCORPION. His research interests include resource management in future wireless systems, cloud-RAN, network slicing in 5G, software-defined wireless networks, and mobile cloud offloading. Dr. Langar is Chair of IEEE ComSoc Technical Committee on Information Infrastructure and Networking (TCIIN). He was the corecipient of the IEEE/IFIP International Conference on Network and Service Management 2014 (IEEE/IFIP CNSM 2014) Best Paper Award.

Xiaoming Fu received the Ph.D. degree from Tsinghua University, Beijing, China, in 2000. He was a Research Staff with Technical University of Berlin, Berlin, Germany, before moving to Göttingen in 2002, where he has been a Professor and Head of computer networks group, since 2007. He is a currently a Full Professor of computer science with the University of Göttingen, Göttingen, Germany. His research interests include networked systems and applications, including mobile and cloud computing, social networks, and big data analysis. Dr. Fu is an IEEE Senior Member and an IET Fellow. He currently serves on the editorial boards of IEEE COMMUNICATIONS MAGAZINE, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, and Computer Communications. From 2010 to 2012, he served as Vice Chair of IEEE Technical Committee of Computer Communications (TCCC). From 2011 to 2013, he was Chair of IEEE Technical Committee on Internet (ITC), and has been the coordinator of EU FP7 MobileCloud, GreenICN and CleanSky projects, as well as H2020 ICN2020 projects.

Li Wang (S'08–M'14–S M'16) received the Ph.D. degree from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2009. She held Visiting Professor positions with the School of Electrical and Computer Engineering, Georgia Tech, Atlanta, GA, USA, from December 2013 to January 2015, and with the Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden, from August to November, 2015. She is a Full Professor with the School of Electronic Engineering, BUPT, where she heads the High Performance Computing and Networking Laboratory. She is also with the Key Laboratory of the Universal Wireless Communications, Ministry of Education, China.

Her research interests include wireless communications, secure communications, cooperative networking, and distributed networking and storage. She has authored/coauthored two books: *Device-to-Device Communications* (Springer, 2016) and *Physical Layer Security* (Springer, 2017). Dr. Wang is the Symposium Chair of IEEE ICC 2019 on Cognitive Radio and Networks Symposium, and chairs the special interest group (SIG) on Social Behavior Driven Cognitive Radio Networks for the IEEE Technical Committee on Cognitive Networks. She also served technical program committees of multiple IEEE conferences, including IEEE GLOBECOM, ICC, WCNC, and VTC over the years. She has been an Editor for the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY and an Associate Editor for the IEEE ACCESS, since June 2016.



Zhu Han (S'01–M'04–S M'09–F'14) received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 1997, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Maryland, College Park, MD, USA, in 1999 and 2003, respectively.

From 2000 to 2002, he was an R&D Engineer with JDSU, Germantown, Maryland. From 2003 to 2006, he was a Research Associate with the University of Maryland. From 2006 to 2008, he was an Assistant Professor with Boise State University, Idaho, USA. Currently, he is a Professor with the Electrical and Computer Engineering Department as well as the Computer Science Department at the University of Houston, TX, USA.His research interests include wireless resource allocation and management, wireless communications and networking, game theory, big data analysis, security, and smart grid. Dr. Han was the recipient of an NSF Career Award in 2010, the Fred W. Ellersick Prize of the IEEE Communication Society in 2011, the EURASIP Best Paper Award for the Journal on Advances in Signal Processing in 2015, IEEE Leonard G. Abraham Prize in the field of Communications Systems (best paper award in IEEE JSAC) in 2016, and several best paper awards in IEEE conferences. Currently, he is an IEEE Communications Society Distinguished Lecturer.