Online Learning for Edge Node Program Placement in Mobile Edge Computing Networks

Mingjie Feng *Member, IEEE* and Marwan Krunz *Fellow, IEEE*

***Abstract*—Mobile edge computing (MEC) is a key technology to support computationally intensive mobile applications with stringent latency requirements. With MEC servers deployed at network edge (e.g., base stations), the computational tasks generated by various applications can be offloaded to nearby edge nodes (ENs) and timely processed there. Meanwhile, future mobile applications will be more diverse and complex, which will need to be supported by a large number of complicated programs. As the storage space of ENs is limited, it is infeasible for each EN to store the program codes of all applications. Thus, it is necessary to optimize program placement at ENs to fully harvest the potential of MEC. In this paper, we investigate the problem of program placement and user association in storage-limited MEC networks. Such a problem is formulated as a sequential decision-making problem. We first consider the single EN scenario and propose an online learning-based solution. We then propose a solution framework for the multi-EN scenario, where we decompose the original problem into three subproblems and iteratively solve them with low-complexity approaches. Simulation results show that the average latency achieved by our proposed schemes is 30% to 70% lower than two benchmark schemes and is on average less than 10% higher than a lower bound.**

***Index Terms*—Mobile edge computing; low-latency applications; storage-limited systems; program placement optimization; multi-armed bandit; Thompson sampling.**

I. Introduction

Mobile Internet of Things (IoT) applications (e.g., connected and autonomous vehicles, augmented/virtual reality, etc.) often require executing delay-sensitive yet computationally intensive tasks [2], [3]. With limited computational capability, it is quite challenging for mobile devices to execute these tasks in a timely manner. Mobile edge computing (MEC) provides an effective solution to this challenge. By deploying MEC servers within the radio access network, e.g., close to base stations (BSs) or access points (APs), the computational tasks generated by users can be offloaded to and executed by nearby *edge nodes* (ENs)[1] [4]. Due to their proximity to end users, ENs can deliver low-latency services.

This research was supported in part by NSF (awards # 2434021, 2413009, 2229386, 2425535, and 2432139) and by the WISPER Center, and in part by the NSFC under Grant 62101202. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the author(s) and do not necessarily reflect the views of NSF.

M. Feng is with Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, 430074 China. M. Krunz is with the Department of Electrical and Computer Engineering, The University of Arizona, Tucson, AZ, 85721 USA. Email: mingjiefeng@hust.edu.cn, krunz@email.arizona.edu.

An abridged version of this paper was presented at IEEE WoWMoM 2021, Online, June 2021 [1].

[1]An EN refers to a combination of an MEC server and a BS/AP.

To execute a task at an EN, the input data (e.g., video clips taken by the mobile device) and the program that executes the task (e.g., object recognition software) are required. Many research papers assume that the programs of all tasks are stored at each EN. This way, users can always offload any task to a nearby EN. It is also often assumed that the programs are always loaded in the random-access memory (RAM) of each EN, so that a task can be immediately executed once the input data has been uploaded by the user, without having to wait for program loading. However, these assumptions are not practical for future MEC systems. Specifically, mobile IoT applications will become more diverse (e.g., activity monitoring and anomaly detection in different scenarios), and their complexity will increase (e.g., going from Level 1 autonomous driving to Level 5 autonomous driving), with a corresponding increase in the sizes of programs associated with these applications. On the other hand, with the projected massive deployment of ENs, cheap hardware with relatively small storage will likely be used to avoid high capital expenditure (CAPEX). Thus, it is quite unlikely that any EN will need to store all programs generated by different users. When the program of a requested task is not stored in a given EN, the task has to be executed at the user's own device or handed over to another EN that has the program. Both options increase the overall latency. To this end, storage utilization at an EN needs to be optimized in a way that *minimizes the average latency*.

Meanwhile, the patterns of task requests received by ENs vary in time and space, depending on the specific application. For example, ENs located at roadside are expected to receive more tasks related to smart transportation, especially during the rush hours. This means that ENs must learn to optimize their own *program placement* strategy, which includes program storage and preloading. Intuitively, frequently requested programs should have a higher priority to be stored and preloaded. However, other factors need to be taken into account, such as program file size, computational complexity, and loading time of each program. In an MEC system with multiple users and ENs, user association is a design problem that is coupled with program placement. On the one hand, users in overlapping coverage areas of neighboring ENs have multiple choices of ENs for task offloading. As the ENs vary in program availability, communication link quality, and computational capability, the overall latency of a user is dictated by its EN selection. On the other hand, user association determines the traffic load at each EN and the associated communication and computational latencies. Thus, program placement strategies of neighboring ENs are coupled via user association.

In this paper, we investigate the optimization of program placement and user association in storage-constrained MEC systems, aiming to minimize the average latency of all users over a finite time interval. We develop efficient solutions based on an online learning framework called multi-armed bandit (MAB). In particular, we employ an online learning framework called multi-armed bandit (MAB) to model our problem and apply Discounted Thompson Sampling (DTS)-based approaches to solve the formulated problems. The main contributions of this paper are as follows:

- We formulate the problem of program placement (including program storage and preloading) and user association as a non-stationary sequential decision-making problem with coupled variables.
- We propose an online learning-based solution for the single-EN scenario. First, we transform the formulated problem into a non-stationary MAB problem in which each EN acts as an agent that optimizes its program placement strategy. Then, instead of directly applying an existing algorithm to solve the MAB problem, we reduce the complexity of the MAB problem by excluding the strategies that are impossible to be optimal and model the remaining strategies as the arms to be played. We also quantify the complexity reduction by deriving an upper bound on the number of arms in the reduced MAB problem. Finally, we solve the reduced MAB problem using a DTS algorithm and provide regret analysis with mathematical proof.
- We develop a solution framework for the multi-EN scenario by decomposing the original problem into three subproblems that are solved at each time slot using low-complexity schemes. The first subproblem targets learning task popularity (i.e., the probabilities that various tasks will be requested by users). This subproblem is formulated as a non-stationary MAB problem and solved by a DTS algorithm that only requires ENs to perform a simple parameter update at each round of learning. We also provide a regret analysis for this DTS algorithm. The second subproblem focuses on optimizing program placement under a given user association. This subproblem is a Knapsack problem, which we solve via a low-complexity greedy algorithm. We provide a lower bound on the performance of this algorithm. The last subproblem is user association, for which we propose a dual decomposition-based approach to derive a near-optimal solution. The proposed user association solution is implemented in a distributed manner, requiring a few information exchanges between users and ENs. We provide complexity analysis, which shows that the number of iterations is a function of only the convergence threshold. This allows for low-complexity implementation.
- We evaluate the performance of the proposed schemes using simulations based on a cellular network setup, in which the latency performance under different task profiles (popularity, complexity, and program file size of tasks) and network scenarios (varying numbers of ENs and users) are compared. The results show that the proposed schemes can reduce the average latency by $30\%\sim70\%$ compared to two benchmark schemes. To demonstrate that near-optimal performance can be achieved, we derive a lower bound on the latency. The results show that, on average, the latency achieved by our solution is less than 10% higher than the lower bound.

In the remainder of this paper, we overview related work in Section II. We introduce the system model and problem formulation in Sections III and IV, respectively. The solution algorithms for the single-EN and multi-EN scenarios are presented in Sections V and VI, respectively. We present our simulation results in Section VII and conclude the paper in Section VIII.

II. Related Work

As an enabling technology for IoT applications, MEC has attracted significant attention from both industry and academia. Early standardization efforts were initiated by the Industry Specification Group (ISG) of the European Telecommunications Standards Institute (ETSI) [2]. A literature overview of MEC can be found in [4]. Recently, an analytical framework for the fundamental aspects of MEC, including communication, computation, caching, and control, was introduced in [5].

The problem of program placement at ENs has some similarities with content caching/prefetching in content delivery networks (CDNs), where popular content is cached at stations that are close to end users (e.g., small BSs), allowing such content to be quickly delivered to these users [9]–[11]. Because the content popularity profile is often unknown, machine learning (ML) algorithms have been developed to predict the content request pattern and optimize caching strategies (e.g., [12]–[14]). In particular, an MAB-based algorithm was employed in [14] to learn the file request pattern at a small BS, and a greedy file placement scheme was proposed based on the predicted popularity profile. MEC-supported content caching was considered in some recent works [15]–[18], where the computing capability of MEC servers was utilized to improve CDN performance. For example, the MEC server can preprocess certain files (e.g., perform video transcoding) to reduce the processing time for users. It can also help compress files to save storage space. In contrast to these works that optimize content placement for fast content delivery in CDNs, we consider optimizing the placement of *user programs* at ENs, which determines the task offloading availability, aiming to fully harvest the benefit of MEC systems in providing low-latency computing services. State-of-the-art techniques for content placement cannot be directly applied to the program placement problem for the following reasons. First, compared to content placement problems, which involve a single optimization variable, we optimize both program storage and program preloading strategies, resulting in two sets of coupled decision variables. Second, given the trend in network densification, the coverage areas of neighboring ENs are likely to overlap, making the program placement strategy highly coupled with the user association strategy. Third, users' requests for different contents (e.g., videos) are driven by their interests/preferences, which are relatively stable and easy to capture. In contrast, users' requests for various IoT applications (and the corresponding programs) are demand-driven. Such demands fluctuate rapidly in time and space according to certain patterns. Thus, new methods are needed to capture the temporal and spatial patterns of users' requests for various programs, which is critical for improving the storage utilization of ENs.

User association in MEC systems has been widely investigated. In [20]–[23], user association was jointly optimized with computational resource allocation, task partitioning, and power control, respectively, aiming to minimize the total energy consumption and task completion latency. In [7], joint optimization of EN selection and computing resources was considered to improve the quality of experience (QoE) of users. In [24], user association was jointly optimized with content caching to minimize the latency caused by content acquisition and handover. Recently, user association was investigated in the contexts of drone-based MEC systems [25] and satellite-based MEC systems [26], [27], in which machine learning algorithms were proposed to capture system dynamics. In our problem, user association is coupled with the program placement strategy of ENs, where we jointly optimize the two to minimize the average latency of all users.

III. System Model

A. Problem Setup

We consider an MEC system with J ENs, indexed by $j \in \{1,\dots,J\} \triangleq \mathcal{J}$. These ENs provide task offloading services to K mobile user equipments (UEs), indexed by $k \in \{1,\dots,K\} \triangleq \mathcal{K}$. Each EN has a storage space (e.g., disk) with capacity E_{H} (in bytes) that stores the program codes of different tasks. Each EN also has a RAM of capacity E_{R} (in bytes) that loads the programs from disk, allowing the CPU to read and execute these programs.

We consider a finite time interval that is divided into T slots indexed by $t = 1,\dots,T$. In each time slot t, each UE randomly requests one of the N possible computational tasks, following the task request pattern at that time slot. Each task is executed by a unique program. The tasks and their associated programs are indexed by $i \in \{1,\dots,N\} \triangleq \mathcal{N}$. As mentioned before, the task requests received by various ENs vary in time and space. For analytical tractability, we use the logical coverage area of ENs [2] to model the location-specific task request patterns of UEs[3]. Specifically, UEs within the logical coverage of each EN follow the same time-varying task popularity profile at each time slot (but these UEs are not necessarily served by the EN). We also assume that the task generation processes of various UEs are mutually independent, due to the fact that different UEs act independently of each other. Let $\theta_{i,j}^{[t]}$ be the probability that task i is generated by any arbitrary UE within the coverage of EN j at time t. Then, $\sum_{i=1}^{N} \theta_{i,j}^{[t]} = 1$ for $j \in \mathcal{J}, t = 1,\dots,T$. The *task popularity file* for EN j at time t, denoted by $\boldsymbol{\theta}_j^{[t]} = \left[\theta_{1,j}^{[t]},\dots,\theta_{N,j}^{[t]}\right]$, is determined by the statistical behavior of UEs near EN j at time t (e.g., roadside ENs receive more vehicle-generated tasks during rush hours), rather than the instantaneous task requests of UEs associated with EN j. Before the learning process starts, $\boldsymbol{\theta}_j^{[t]}$ is unknown to each EN j.

For analytical tractability, we consider a finite number of task request patterns for UEs served by each EN. Specifically, we assume that at each time t, $\{\theta_{1,j}^{[t]},...,\theta_{N,j}^{[t]}\}$ take values from a set that satisfies $\sum_{i=1}^{N} \theta_{i,j}^{[t]} = 1$. Then, there are $N!$ possible $\boldsymbol{\theta}_j^{[t]}$. At each time t, each $\boldsymbol{\theta}_j^{[t]}$ is in one of the $N!$ patterns. Considering that the task request pattern usually exhibits a periodic pattern (e.g., roadside ENs receive more vehicle-generated tasks during the rush hours of each day), we assume that $\boldsymbol{\theta}_j^{[t]}$ is updated periodically. Besides periodicity, the task request pattern is expected to vary gradually. Thus, we assume that $\boldsymbol{\theta}_j^{[t]}$ stays at the same pattern for multiple time slots and updates to the next pattern, then stays at that pattern for multiple time slots and updates again.

At each time slot t, the programs that are stored on the disk of EN j and loaded to the RAM are specified by two sets of binary variables $a_{i,j}^{[t]}$ and $b_{i,j}^{[t]}$, defined by:

$$a_{i,j}^{[t]} = \begin{cases} 1, & \text{if program } i \text{ is stored on disk of EN } j \\ 0, & \text{otherwise} \end{cases} \quad i = 1,\dots,N,\ j \in \mathcal{J},\ t = 1,\dots,T. \tag{1}$$

$$b_{i,j}^{[t]} = \begin{cases} 1, & \text{if program } i \text{ is preloaded to the RAM of EN } j \\ 0, & \text{otherwise} \end{cases} \quad i = 1,\dots,N,\ j \in \mathcal{J},\ t = 1,\dots,T. \tag{2}$$

Because a program must be stored on the disk before being loaded into the RAM, $\{a_{i,j}^{[t]}\}$ and $\{b_{i,j}^{[t]}\}$ must satisfy $b_{i,j}^{[t]} \le a_{i,j}^{[t]},\ i \in \mathcal{N},\ j \in \mathcal{J},\ t = 1,\dots,T$. Let s_i be the size of program i (in bytes) and q_i be the storage space occupied by this program (in bytes) when loaded to RAM, $q_i > s_i$. Then, for all $i \in \mathcal{N}$, $a_{i,j}^{[t]}$ and $b_{i,j}^{[t]}$ should satisfy:

$$\begin{cases} \sum_{i=1}^{N} a_{i,j}^{[t]} s_i \le E_{\mathrm{H}},\ \forall j \in \mathcal{J},\ t = 1,\dots,T \\ \sum_{i=1}^{N} b_{i,j}^{[t]} q_i \le E_{\mathrm{R}},\ \forall j \in \mathcal{J},\ t = 1,\dots,T. \end{cases} \tag{3}$$

To update the program storage, each EN downloads the programs to be added from the core network via backhaul and deletes the programs to be removed. Each EN must also decide the set of UEs associated with it. This is indicated by the following binary variables:

$$x_{k,j}^{[t]} = \begin{cases} 1, & \text{if UE } k \text{ is associated with EN } j \\ 0, & \text{otherwise,} \end{cases} \quad k \in \mathcal{K},\ j \in \mathcal{J},\ t = 1,\dots,T. \tag{4}$$

When UE k is associated with EN j (i.e., $x_{k,j}^{[t]} = 1$) and it generates task i at time t, the task will be executed by EN j only if the program codes of task i are stored at EN j.

[2]The logical coverage area of an EN j is defined as follows. When any UE is located in the logical coverage area of EN j, its nearest EN is EN j. Obviously, the logical coverage areas of different ENs are non-overlapping. In contrast, the physical coverage area of each EN j is the area in which UEs can communicate with EN j.

[3]Since the physical coverage areas of neighboring ENs may overlap, if we use the physical coverage areas of ENs to characterize the location-specific task request patterns, the UEs in the overlapping areas would be counted multiple times in the objective function. Thus, we use logical coverage areas to characterize the spatially varying task request patterns, which ensures that each UE is in the logical coverage area of *only one* EN.

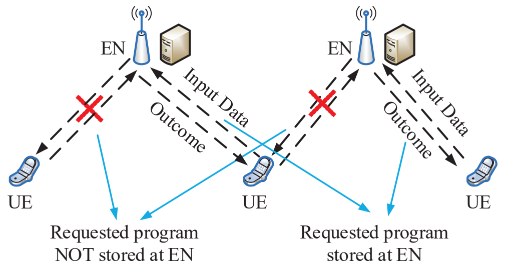
EN
EN
Input Data
Outcome
Input Data
Outcome
UE
UE
UE
Requested program NOT stored at EN
Requested program stored at EN

Fig. 1. Example of task offloading in a multi-EN system with limited storage. A UE can offload a computational task to its associated EN only when the task program is stored at the EN.

Otherwise, the task can only be executed by UE k. Fig. 1 illustrates the feasibility of task offloading under different program availability scenarios.

The sequence of program placement and user association strategies of the J ENs for $t = 1, \ldots, T$ is called a policy, which is denoted by $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_J\}$.

B. Communication Model

We consider a generic cellular network (e.g., LTE or 5G) in which UEs are served by BSs acting as ENs. UEs served by the same EN are assumed to have the same priority, hence the communication resource is equally allocated to them [19]. At time t, the data rate of UE k when associated with EN j is given by:

$$R_{k,j}^{[t]} = \frac{W \log\left(1 + \gamma_{k,j}^{[t]}\right)}{\sum_{k=1}^{K} x_{k,j}^{[t]}} \tag{5}$$

where W is the available bandwidth for each EN, and $\gamma_{k,j}^{[t]}$ is the uplink signal-to-interference-plus-noise ratio (SINR) for the transmission from UE k to EN j, which can be measured by EN j. Let $\tilde{s}_{k,i}$ be the size of input data (in bits) for task i generated by UE k, the offloading time of task i from UE k to EN j is $\frac{\tilde{s}_{k,i}}{R_{k,j}^{[t]}}$. To successfully receive UE data, we must have $\gamma_{k,j}^{[t]} \geq \gamma_{\text{th}}$, where γ_{th} is the SINR threshold. For simplicity, we do not consider multi-rate communications in this paper. Let $\Omega_k^{[t]}$ denote the set of candidate ENs that can be used by UE k for task offloading at time t, $\Omega_k^{[t]} = \left\{ j \in \mathcal{J} \,\middle|\, \gamma_{k,j}^{[t]} \geq \gamma_{\text{th}} \right\}$. Then, $x_{k,j}^{[t]}$ must satisfy:

$$x_{k,j}^{[t]} = 0, \forall j \notin \Omega_k^{[t]}. \tag{6}$$

C. Computational Model

The computational resource required for executing a task is determined by the size of input data (in bits) and the task's computational complexity, represented in the number of CPU cycles needed to execute one bit of the task. Let z_i be the computational complexity of task i. Then, the number of CPU cycles required to complete task i is $\tilde{s}_{k,i} z_i$.

1) UE Computing: Let $c_k^{(\mathrm{L})}$ be the computational capability of UE k, measured in CPU cycles per second. Then, the execution time for task i at UE k is $\frac{\tilde{s}_{k,i} z_i}{c_k^{(\mathrm{L})}}$. Because UE k may generate any one of the N tasks with EN-specific and time-dependent probabilities, the expected task execution time (in seconds) by UE k when it is within the logical coverage area of EN j at time t is given by:

$$D_{k,j}^{(\mathrm{L})\,[t]} = \sum_{i=1}^{N} \theta_{i,j}^{[t]} \frac{\tilde{s}_{k,i} z_i}{c_k^{(\mathrm{L})}}. \tag{7}$$

2) MEC Server Computing: Let $c_j^{(\mathrm{E})}$ be the computational capability of EN j. We assume that $c_j^{(\mathrm{E})}$ is equally split between all UEs associated with EN j during each time slot t. Then, the computational capability allocated to UE k by EN j at time t is $c_{k,j}^{(\mathrm{E})\,[t]} = \frac{c_j^{(\mathrm{E})}}{\sum_{k=1}^{K} x_{k,j}^{[t]}}$. Similar to (7), the expected computing time for executing the task of UE k at EN j is given by:

$$D_{k,j}^{(\mathrm{E})\,[t]} = \sum_{i=1}^{N} \theta_{i,j}^{[t]} \frac{\tilde{s}_{k,i} z_i}{c_{k,j}^{(\mathrm{E})\,[t]}} = \sum_{i=1}^{N} \theta_{i,j}^{[t]} \frac{\tilde{s}_{k,i} z_i \sum_{k=1}^{K} x_{k,j}^{[t]}}{c_j^{(\mathrm{E})}}. \tag{8}$$

D. Latency Analysis

Without loss of generality, suppose that at time t, UE k is within the logical coverage area of EN j. The latency for completing the task of UE k is calculated by combining two cases: UE k is associated with EN j ($x_{k,j}^{[t]} = 1$) and UE k is not associated with EN j ($\sum_{j=1}^{J} x_{k,j}^{[t]} = 0$).

For the first case, depending on the program availability at EN j, the task may be executed by EN j (if requested program i is stored at EN j, i.e., $a_{i,j}^{[t]} = 1$) or by UE k (if requested program i is not stored at EN j, i.e., $a_{i,j}^{[t]} = 0$). If the task is executed by EN j, the expected task completion latency is the sum of expected task offloading latency $\sum_{i=1}^{N} \theta_{i,j}^{[t]} \frac{\tilde{s}_{k,i}}{R_{k,j}^{[t]}}$ and the expected task execution latency $D_{k,j}^{(\mathrm{E})\,[t]}$; if the task is executed by UE k, the task completion latency is the expected local execution latency $D_{k,j}^{(\mathrm{L})\,[t]}$. Similarly, the program loading status of the requested program i at EN j (indicated by $b_{i,j}^{[t]}$) determines if the loading time will be incurred. Thus, the expected latency for completing the task of UE k at time t is given by:

$$\begin{aligned} D_{k,j}^{(\mathrm{A})\,[t]} &= \left(\sum_{i=1}^{N} \theta_{i,j}^{[t]} \frac{\tilde{s}_{k,i}}{R_{k,j}^{[t]}} + D_{k,j}^{(\mathrm{E})\,[t]} \right) a_{i,j}^{[t]} \\ &\quad + D_{k,j}^{(\mathrm{L})\,[t]} \left(1 - a_{i,j}^{[t]}\right) + \sum_{i=1}^{N} \theta_{i,j}^{[t]} l_{i,j} \left(1 - b_{i,j}^{[t]}\right) \\ &= \sum_{i=1}^{N} \theta_{i,j}^{[t]} \left[\left(\frac{\tilde{s}_{k,i} z_i}{c_{k,j}^{(\mathrm{E})\,[t]}} + \frac{\tilde{s}_{k,i}}{R_{k,j}^{[t]}} \right) a_{i,j}^{[t]} \right. \\ &\quad \left. + \left(1 - a_{i,j}^{[t]}\right) \frac{\tilde{s}_{k,i} z_i}{c_k^{(\mathrm{L})}} + \left(1 - b_{i,j}^{[t]}\right) l_{i,j} \right] \end{aligned} \tag{9}$$

where $l_{i,j}$ is the loading time of program i at EN j. For simplicity, the latency for downloading the result of an executed task is ignored due to its small size [7], [8]. In case the downloading time is non-negligible, it can be calculated in the same way as the uploading time, and the expression of ${D_{k,j}^{(\mathrm{A})}}^{[t]}$ can be modified accordingly.

We then consider the case in which UE k is not associated with any EN at time t (although UE k is within the logical coverage area of EN j). In this case, the UE has to execute the task by itself, with expected latency of ${D_{k,j}^{(\mathrm{L})}}^{[t]}$. Combining the two cases, the expected latency for completing the task of UE k when it is within the logical coverage area of EN j is given by:

$$D_{k,j}^{[t]} = x_{k,j}^{[t]} {D_{k,j}^{(\mathrm{A})}}^{[t]} + \left(1 - x_{k,j}^{[t]}\right) {D_{k,j}^{(\mathrm{L})}}^{[t]}. \tag{10}$$

IV. Problem Formulation

In this paper, we aim to find the optimal policy $\boldsymbol{\pi}$ (i.e., the sequence of program placement and user association strategies) that minimizes the accumulated average latency of all UEs over time slots $t = 1, \ldots, T$. With $D_{k,j}^{[t]}$ given in (10), the sum latency of all UEs at time t is given by:

$$\begin{aligned}\sum_{k=1}^{K}\sum_{j=1}^{J} D_{k,j}^{[t]} = &\sum_{k=1}^{K}\sum_{j=1}^{J} {D_{k,j}^{(\mathrm{L})}}^{[t]} \\ &+ \sum_{k=1}^{K}\sum_{j=1}^{J} x_{k,j}^{[t]} \left({D_{k,j}^{(\mathrm{A})}}^{[t]} - {D_{k,j}^{(\mathrm{L})}}^{[t]}\right).\end{aligned} \tag{11}$$

Let $\Delta_{k,j}^{[t]} = {D_{k,j}^{(\mathrm{L})}}^{[t]} - {D_{k,j}^{(\mathrm{A})}}^{[t]}$. Obviously, $\Delta_{k,j}^{[t]}$ is the expected latency reduction of UE k when associated with EN j compared to executing the task by itself. It can be seen from (11) that minimizing $\sum_{k=1}^{K}\sum_{j=1}^{J} D_{k,j}^{[t]}$ is equivalent to maximizing $\sum_{k=1}^{K}\sum_{j=1}^{J} \Delta_{k,j}^{[t]}$. Thus, we define the system reward at time t as follows:

$$\mathcal{R}_{\boldsymbol{\pi}}^{[t]} = \sum_{k=1}^{K}\sum_{j=1}^{J} x_{k,j}^{[t]} \Delta_{k,j}^{[t]} \tag{12}$$

where $\Delta_{k,j}^{[t]}$ is given by:

$$\begin{aligned}\Delta_{k,j}^{[t]} = \sum_{i=1}^{N} \theta_{i,j}^{[t]} \Bigg[& a_{i,j}^{[t]} \left(\frac{\tilde{s}_{k,i} z_i}{c_k^{(\mathrm{L})}} - \frac{\tilde{s}_{k,i} z_i}{{c_{k,j}^{(\mathrm{E})}}^{[t]}} - \frac{\tilde{s}_{k,i}}{R_{k,j}^{[t]}} \right) \\ &+ (1 - b_{i,j}^{[t]}) l_{i,j} \Bigg].\end{aligned} \tag{13}$$

Then, the problem of finding the optimal policy that maximizes the accumulated reward is formulated as:

$$\mathbf{P1}: \max_{\boldsymbol{\pi}} \textstyle\sum_{t=1}^{T} \mathcal{R}_{\boldsymbol{\pi}}^{[t]} \tag{14}$$

$$\text{s.t.} \quad \textstyle\sum_{i=1}^{N} a_{i,j}^{[t]} s_i \le E_{\mathrm{H}},\ j \in \mathcal{J},\ t = 1, \ldots, T, \tag{15}$$

$$\textstyle\sum_{i=1}^{N} b_{i,j}^{[t]} q_i \le E_{\mathrm{R}},\ j \in \mathcal{J},\ t = 1, \ldots, T, \tag{16}$$

$$b_{i,j}^{[t]} \le a_{i,j}^{[t]},\ i \in \mathcal{N},\ j \in \mathcal{J},\ t = 1, \ldots, T, \tag{17}$$

$$\textstyle\sum_{j=1}^{J} x_{k,j}^{[t]} \le 1,\ k \in \mathcal{K},\ t = 1, \ldots, T, \tag{18}$$

$$\textstyle\sum_{k=1}^{K} x_{k,j}^{[t]} \le U_j,\ j \in \mathcal{J},\ t = 1, \ldots, T, \tag{19}$$

$$x_{k,j}^{[t]} = 0,\ k \in \mathcal{K},\ \forall j \notin \Omega_k^{[t]} \tag{20}$$

$$a_{i,j}^{[t]},\ b_{i,j}^{[t]} \in \{0, 1\},\ i \in \mathcal{N},\ j \in \mathcal{J},\ t = 1, \ldots, T, \tag{21}$$

$$x_{k,j}^{[t]} \in \{0, 1\},\ k \in \mathcal{K},\ j \in \mathcal{J},\ t = 1, \ldots, T. \tag{22}$$

In $\mathbf{P1}$, Constraints (15) and (16) reflect the storage capacities of disk and RAM, respectively. Constraint (17) is due to the fact that a program must be stored on the disk before being loaded into the RAM. Constraint (18) indicates that each UE can be associated with at most one EN. Constraint (19) specifies an upper bound on the number of UEs that can be served by EN j. Constraint (20) is obtained from Eq. (6), and it reflects the limited communication range of UEs.

V. Solution for The Single-EN Scenario

We first consider the scenario in which ENs are not densely deployed, hence the physical coverage areas of neighbor ENs do not overlap. In the single-EN scenario, the user association strategies of different ENs are mutually independent. Thus, the program placement strategy of each EN can be optimized independently. Because each UE only has one option of EN to associate with, and UEs within the logical coverage area of an EN follow the same task request pattern, serving different sets of UEs results in the same latency performance. Thus, optimizing user association would not bring any performance gain. Without loss of generality, we consider the program placement optimization at EN j:

$$\begin{aligned}\mathbf{P2}: &\max_{\pi_j} \textstyle\sum_{t=1}^{T} \sum_{k=1}^{K} x_{k,j}^{[t]} \Delta_{k,j}^{[t]} \\ &\text{s.t.:}\ (15) - (17) \text{ and } (21).\end{aligned} \tag{23}$$

In $\mathbf{P2}$, user association variables $\{x_{1,j}^{[t]}, \ldots, x_{K,j}^{[t]}\}$ are given for $t = 1, ..., T$, and the constraints related to $\{x_{1,j}^{[t]}, \ldots, x_{K,j}^{[t]}\}$ are omitted since user association is not opitmized. π_j is specified by $\{a_{i,j}^{[t]}, b_{i,j}^{[t]}\}$ for $i \in \mathcal{N}, t = 1, ..., T$. $\mathbf{P2}$ is a non-stationary time series decision-making problem with two sets of coupled decision variables. To obtain an efficient solution, we first transform $\mathbf{P2}$ into a non-stationary MAB problem. Then, we apply a Discounted Thompson Sampling (DTS) algorithm to obtain a policy that approaches the optimal program placement strategy.

The program placement strategy at time t can be expressed by a $2N \times 1$ vector $[a_{1,j}^{[t]}, \ldots, a_{N,j}^{[t]}, b_{1,j}^{[t]}, \ldots, b_{N,j}^{[t]}]$. As the elements of this vector are binary variables, the total number of possible values (strategies) is 2^{2N}. We denote the various possible strategies by $\mathbf{z}_m$, $m = 1, \ldots, 2^{2N}$. If we model all 2^{2N} program placement strategies as arms in an MAB problem, the complexity of the problem would be prohibitively high when N is large. To this end, we reduce the dimensionality of the problem by treating the feasible program placement strategies that are possibly optimal as arms. These feasible strategies are obtained with the following steps:

(a) Among all vectors $\mathbf{z}_m$, $m = 1, \ldots, 2^{2N}$, we select the ones that satisfy all the constraints (15)-(17).

(b) Among all the vectors selected by step (a), we keep the ones that satisfy the following conditions: if the value of any $a_{i,j}^{[t]}$ or $b_{i,j}^{[t]}$ is changed from 0 to 1, one of the storage constraints in (15) and (16) would be violated. The other vectors are removed since they cannot be the optimal program placement strategy, i.e., the optimal strategy must be among the remaining vectors.[4]

(c) Let M be the number of the remaining program placement strategies after steps (a) and (b). We denote the set of these strategies as $\mathcal{F} = \{\mathbf{z}_1, \dots, \mathbf{z}_M\}$. Then, the M strategies in $\mathcal{F}$ are the arms to be played and learned. The objective of the MAB problem is to find the arm with the largest mean reward.

After the set of arms has been determined, we apply the DTS algorithm proposed in [30] to solve the MAB problem. In the MAB problem, the reward for playing each arm m at time t is a random variable that takes values in $[0, 1]$, which can be generated from any arbitrary distribution with mean $\rho_m^{[t]}$. Here, $\rho_m^{[t]}$ is defined as the *normalized* expected latency reduction that results from applying the mth program placement strategy, given by:

$$\rho_m^{[t]} = \frac{\overline{\Delta}_j \left(\mathbf{a}_j(m), \mathbf{b}_j(m)\right)^{[t]}}{\Delta_j{}'} \tag{24}$$

where $\overline{\Delta}_j \left(\mathbf{a}_j(m), \mathbf{b}_j(m)\right)^{[t]}$ is the expected latency reduction for UEs associated with EN j under program placement strategy m at time t, denoted by $\mathbf{a}_j(m) = [a_{1,j}(m), \dots, a_{N,j}(m)]$ and $\mathbf{b}_j(m) = [b_{1,j}(m), \dots, b_{N,j}(m)]$; $\Delta_j{}'$ is the reference latency reduction, which is calculated by setting all elements in $\mathbf{a}_j(m)$ and $\mathbf{b}_j(m)$ to 1. Note that $\overline{\Delta}_j \left(\mathbf{a}_j(m), \mathbf{b}_j(m)\right)^{[t]}$ and $\Delta_j{}'$ are calculated based on the long-term average value of the per-bit offloading time, i.e., the expected value of $\frac{1}{R_{k,j}}$ for UEs served by EN j.

Since the mean rewards of playing different arms ($\rho_m^{[t]}$, $m = 1, \dots, M$) are not known a priori, EN j has a "belief" for the distribution of each $\rho_m^{[t]}$, which is updated at each iteration. In the DTS algorithm [30], the prior distributions of arms are initialized as beta distributions and are updated with a certain pattern at each iteration, allowing the agent to "forget" past observations and accommodate the non-stationary environment. In addition to the updates applied to all arms, the prior distribution of the played arm is updated based on the outcome of a Bernoulli trial. The reason for using Beta distribution as the prior distribution of each $\rho_m^{[t]}$ is that the posterior distribution is still a Beta distribution after each update, i.e., Beta distribution is a conjugate prior of the distribution of $\rho_m^{[t]}$. Let $\eta_m^{[t]}$ be the prior distribution for ρ_m at time t, it follows beta distribution with parameters $\alpha_m^{[t]}$ and $\beta_m^{[t]}$, i.e., $\eta_m^{[t]} \sim \mathrm{Beta}(\alpha_m^{[t]}, \beta_m^{[t]})$, $m = 1, \dots, M$, $t = 1, \dots, T$. For a beta distribution $\mathrm{Beta}(\alpha, \beta)$ that is conjugated with a Bernoulli trial, α and β are the numbers of observed successes and failures of the Bernoulli trial, respectively. The mean of $\mathrm{Beta}(\alpha, \beta)$ is $\alpha/\alpha + \beta$, and the higher α and β, the tighter the distribution is concentrated around its mean. In our problem, the initial prior distribution of each $\eta_m^{[t]}$ is set to be $\eta_m^{[1]} \sim \mathrm{Beta}(1, 1)$, $m = 1, \dots, M$, which corresponds to a uniform distribution.

[4]Proof (by contradiction): Suppose that the optimal program placement strategy is within the vectors removed in step (b). Then, when one more program is added to the optimal strategy, both storage constraints in (15) and (16) will not be violated, meaning that the resulting new strategy is a feasible strategy. With the additional program, the new strategy must outperform the optimal strategy by allowing more tasks executed by ENs (if added to the disk) or reducing the loading time (if added to the RAM), which is a contradiction.

Algorithm 1: DTS Algorithm for Program Placement Optimization in Single-EN Scenario

```
1  Initialize: $\eta_m^{[1]} \sim \mathrm{Beta}(1,1)$, $m = 1, \dots, M$ ;
2  for $t = 1 : T$ do
3      for $m = 1 : M$ do
4          Draw a sample $\hat{\eta}_m^{[t]}$ from the distribution
             $\eta_m^{[t]} \sim \mathrm{Beta}(\alpha_m^{[t]}, \beta_m^{[t]})$ ;
5      end
6      Play arm $m^{*[t]} = \arg\max_m \hat{\eta}_m^{[t]}$ and observe reward $r^{[t]}$ ;
7      Perform a Bernoulli trial with success probability $r^{[t]}$
         and record output $\tilde{r}^{[t]}$ ;
8      $\alpha_{m^*}^{[t+1]} = \gamma\alpha_{m^*}^{[t]} + \tilde{r}^{[t]}$ and $\beta_{m^*}^{[t+1]} = \gamma\beta_{m^*}^{[t]} + (1 - \tilde{r}^{[t]})$ ;
9      $\alpha_m^{[t+1]} = \gamma\alpha_m^{[t]}$ and $\beta_m^{[t+1]} = \gamma\beta_m^{[t]}$, $\forall m \neq m^*$ ;
10 end
```

At each time step t, EN j draws a sample $\hat{\eta}_m^{[t]}$ from the distribution $\eta_m^{[t]} \sim \mathrm{Beta}(\alpha_m^{[t]}, \beta_m^{[t]})$ and records the sampled values $\hat{\eta}_m^{[t]}$, $m = 1, \dots, M$. Then, EN j plays the arm $m^{*[t]} = \arg\max_m \hat{\eta}_m^{[t]}$ and observes the reward $r^{[t]}$. The observed reward is the normalized latency reduction achieved by all UEs that are associated with EN j, given by:

$$r^{[t]} = \frac{\overline{\Delta}_j \left(\mathbf{a}_j(m^*), \mathbf{b}_j(m^*)\right)^{[t]}}{\Delta_j{}'}. \tag{25}$$

Based on $r^{[t]}$, EN j performs a Bernoulli trial with success probability $r^{[t]}$ and observes the outcome $\tilde{r}^{[t]}$. Then, it updates the parameters of the distribution of $\eta_{m^*}^{[t]}$ by:

$$(\alpha_{m^*}^{[t+1]}, \beta_{m^*}^{[t+1]}) = \begin{cases} (\alpha_{m^*}^{[t]} + 1, \beta_{m^*}^{[t]}), & \text{if } \tilde{r}^{[t]} = 1 \\ (\alpha_{m^*}^{[t]}, \beta_{m^*}^{[t]} + 1), & \text{otherwise.} \end{cases} \tag{26}$$

The distributions of other arms ($m \neq m^*$) are updated by:

$$(\alpha_m^{[t+1]}, \beta_m^{[t+1]}) = (\gamma\alpha_m^{[t]}, \gamma\beta_m^{[t]}). \tag{27}$$

Where $\gamma \in (0, 1]$ is the discount factor used to "forget" past observations. Specifically, when $\gamma < 1$, the values of $\alpha_m^{[t]}$ and $\beta_m^{[t]}$ are decreased for $m \neq m^*$ after the tth iteration, resulting in higher variance of $\eta_m^{[t]}$. This way, the probability of selecting unplayed arms increases after each update, which reduces the impact of past observations and makes the algorithm applicable to non-stationary environments.

The sampling and update processes are repeated from time $t = 1$ to $t = T$. Note that when trying different arms during the operation of the DTS algorithm, it is necessary to load new programs into the memory of EN j, which incurs additional latency. However, for any algorithmic solution of online decision-making problems, the latency for trying strategies is inevitable and uncontrollable due to the random actions in the algorithm. Thus, such latency is not incorporated in the reward function of the MAB model and the objective function of **P2**. The DTS algorithm for program placement optimization in the single-EN scenario is summarized in Algorithm 1.

When the context of the program placement problem (e.g., the set of programs and task popularity profile) is switched, the set of arms in Algorithm 1 needs to be regenerated according to steps (a)-(c). This incurs a certain computational cost.

However, as in many optimization problems, such regeneration is typically performed offline, not during the online operation of the DTS algorithm. Thus, the cost of switching context does not impact the operation of the DTS algorithm, making the regeneration of arms practical for real-world implementation.

A. Complexity Analysis

Theorem 1. *The number of remaining program placement strategies, which is also the number of arms in Algorithm 1, is upper bounded by:*

$$M \leq \left(\sum_{y=\left\lceil \frac{E_{\mathrm{H}}-\min\{s_i\}}{\max\{s_i\}} \right\rceil}^{\left\lfloor \frac{E_{\mathrm{H}}}{\min\{s_i\}} \right\rfloor} \binom{N}{y} \right) \cdot \left(\sum_{y=\left\lceil \frac{E_{\mathrm{R}}-\min\{q_i\}}{\max\{q_i\}} \right\rceil}^{\left\lfloor \frac{E_{\mathrm{R}}}{\min\{q_i\}} \right\rfloor} \binom{\left\lfloor \frac{E_{\mathrm{H}}}{\min\{s_i\}} \right\rfloor}{y} \right), \tag{28}$$

The proof of Theorem 1 is presented in the first appendix.

In each iteration of Algorithm 1, EN j performs sampling from the distributions of all the M arms, plays the arm with the maximum sampled value, and updates the prior distribution of the selected arm. Thus, the worst-case complexity of Algorithm 1 is $\mathcal{O}\left(Z \cdot T\right)$, where Z is the right-hand-side of (28).

B. Regret Analysis

To quantify the performance of the proposed solution, we derive an upper bound of the expected total regret for Algorithm 1. Such regret is defined as the accumulated expected penalty for not playing the optimal arm over $t = 1, \ldots, T$, which is given by:

$$\mathbb{E}[\mathcal{R}(T)] = \mathbb{E}\left[\sum\nolimits_{t=1}^{T} \left(\rho_{m^{*[t]}} - \rho_{m^{[t]}}\right)\right] = \sum_{m} \delta_m \mathbb{E}\left[\kappa_m(T)\right] \tag{29}$$

where $m^{[t]}$ is the arm played at time t and $m^{*[t]}$ is the optimal arm at time t (i.e., $\rho_{m^{*[t]}} = \arg\min_{m} \rho_m^{[t]}$); $\delta_m = \rho_{m^{*[t]}} - \rho_{m^{[t]}}$, and $\kappa_m(T)$ is the number of times arm m is played over $t = 1, \ldots, T$. Let $C(T)$ be the maximum number of times of environment changes in $t = 1, ..., T$.

Theorem 2. *The expected regret of Algorithm 1 is upper bounded by:*

$$\mathbb{E}[\mathcal{R}(T)] \leq \mathcal{O}\left(\sqrt{\frac{MT\log T}{1-\gamma}} + \frac{C(T)\log T}{1-\gamma}\right). \tag{30}$$

The proof of Theorem 2 is presented in the second appendix.

VI. Solution for The Multi-EN Scenario

With the trend of *network densification* in cellular networks, the ENs are expected to be densely deployed with overlapping physical coverage areas. As a result, a UE may have multiple choices of ENs for task offloading, and it can optimize its EN selection depending on the program availability, communication link, and computational capability of nearby ENs. On the other hand, user association determines the load distribution among ENs and the latency performance of users associated with each EN. Additionally, a dynamic user association strategy yields time-varying task requests, necessitating the learning of task request patterns over time (i.e., task popularity profiles) to capture these dynamics. Thus, joint optimization of program placement, task popularity profile acquisition, and user association is necessary to achieve the full potential of latency reduction.

It is worth mentioning that Algorithm 1 cannot be applied to the multi-EN scenario. To illustrate the reasons, we consider two cases: (i) each EN acts as an agent that independently optimizes its program placement strategy using Algorithm 1, and (ii) a centralized agent optimizes the joint program placement strategy of all ENs using Algorithm 1. For case (i), since the physical coverage areas of neighboring ENs overlap, users in overlapping coverage areas have multiple candidate ENs for task offloading, and all the programs stored in these ENs can be employed to execute the tasks of these users. If each EN independently optimizes its program placement strategy, neighboring ENs would store the same (or highly similar) set of programs since their task request patterns are highly correlated. As a result, the number of programs available to users in overlapping coverage areas is limited. Conversely, if neighboring ENs store partially different programs, an increased number of programs would be available to users in overlapping coverage areas, resulting in improved latency performance. Thus, the potential of employing the diversified programs of neighboring ENs for latency reduction is not harnessed if Algorithm 1 is applied. For case (ii), since the program placement strategies across ENs are coupled with the user association strategy, the program placement strategies of neighboring ENs are coupled with each other. Because the program placement decision variables are binary, the number of feasible strategies of all ENs grows exponentially with the number of ENs. Accordingly, applying Algorithm 1 in this case would result in a prohibitively large number of arms.

A. Solution Overview

To derive an effective solution, we decompose the original problem into three subproblems to be solved at each time t and iteratively solve them to obtain a near-optimal solution. Specifically, we first apply a DTS algorithm to learn the task popularity profile of each EN at time t, i.e., $\boldsymbol{\theta}_j^{[t]}$. With the task popularity profile of all ENs, the optimization of program placement and user association is formulated as follows:

$$\mathbf{P3}: \max_{\{\mathbf{a}^{[t]},\mathbf{b}^{[t]},\mathbf{x}^{[t]}\}} \sum_{k=1}^{K}\sum_{j=1}^{J} x_{k,j}^{[t]}\Delta_{k,j}^{[t]} \tag{31}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} a_{i,j}^{[t]} s_i \le E_{\mathrm{H}},\ j\in\mathcal{J}, \tag{32}$$

$$\sum_{i=1}^{N} b_{i,j}^{[t]} q_i \le E_{\mathrm{R}},\ j\in\mathcal{J}, \tag{33}$$

$$b_{i,j}^{[t]} \le a_{i,j}^{[t]},\ i\in\mathcal{N},\ j\in\mathcal{J}, \tag{34}$$

$$\sum_{j=1}^{J} x_{k,j}^{[t]} \le 1,\ k\in\mathcal{K}, \tag{35}$$

$$\sum_{k=1}^{K} x_{k,j}^{[t]} \le U_j,\ j\in\mathcal{J}, \tag{36}$$

$$x_{k,j}^{[t]} = 0,\ k\in\mathcal{K},\ \forall j\notin\Omega_k^{[t]}, \tag{37}$$

$$x_{k,j}^{[t]} \in \{0,1\},\ k\in\mathcal{K},\ j\in\mathcal{J}, \tag{38}$$

$$a_{i,j}^{[t]},\ b_{i,j}^{[t]} \in \{0,1\},\ i\in\mathcal{N},\ j\in\mathcal{J}. \tag{39}$$

where $\mathbf{a}^{[t]}$, $\mathbf{b}^{[t]}$, and $\mathbf{x}^{[t]}$ are the matrices $[a_{i,j}^{[t]}]_{i\in\mathcal{N},j\in\mathcal{J}}$, $[b_{i,j}^{[t]}]_{i\in\mathcal{N},j\in\mathcal{J}}$, and $[x_{k,j}^{[t]}]_{k\in\mathcal{K},j\in\mathcal{J}}$. We decompose $\mathbf{P3}$ into two levels of subproblems. The lower-level subproblem is program placement at each EN under a given user association, and we propose a greedy algorithm to solve it. The higher-level subproblem is user association given that the program placement strategy has been applied, we propose a dual decomposition-based approach to solve it.

B. Task Popularity Profile Acquisition

To obtain $\boldsymbol{\theta}_j^{[t]}$ for $j\in\mathcal{J}$, we formulate a non-stationary Bernoulli bandit problem for each EN. Specifically, each EN serves as an agent, which consistently records the task request it receives and learns the task popularity profile accordingly. For EN j, the N programs are regarded as the arms to be played, and the mean reward for playing arm i at time t is $\theta_{i,j}^{[t]}$. Without loss of generality, we consider the task popularity profile acquisition at EN j. The values of $\theta_{i,j}^{[t]}(j\in\mathcal{J})$ are learned via a DTS algorithm similar to the one presented in Section V. Let $\phi_{i,j}^{[t]}$ be the prior distribution of $\theta_{i,j}^{[t]}$ at time t, it follows beta distribution with parameters $\alpha_{i,j}^{[t]}$ and $\beta_{i,j}^{[t]}$, i.e., $\phi_{i,j}^{[t]} \sim \mathrm{Beta}(\alpha_{i,j}^{[t]},\beta_{i,j}^{[t]})$, $i\in\mathcal{N}$. The initial distribution of each $\phi_{i,j}^{[t]}$ is set to be $\phi_{i,j}^{[t]}\sim\mathrm{Beta}(1,1)$, $i\in\mathcal{N}$. At time t, EN j draws a sample $\hat{\phi}_{i,j}^{[t]}$ from the distribution $\phi_{i,j}^{[t]}\sim\mathrm{Beta}(\alpha_{i,j}^{[t]},\beta_{i,j}^{[t]})$ and records the sampled values $\hat{\phi}_{i,j}^{[t]}$, $i\in\mathcal{N}$. Then, it plays the arm $i^{*[t]}=\max_i \hat{\phi}_{i,j}^{[t]}$ by placing the program of task i^* at EN j and observes the reward. We set the reward received by EN j as the proportion of UEs that have requested task i at time t, given by:

$$r_j^{[t]} = \frac{\Theta_{i,j}^{[t]}}{\sum_{i=1}^{N}\Theta_{i,j}^{[t]}} \tag{40}$$

where $\Theta_{i,j}^{[t]}$ is the number of requests for task i at time t. Based on the observed reward, the agent performs a Bernoulli trial

Algorithm 2: DTS Algorithm for Obtaining the Task Popularity Profile of EN j

```
1  Initialize: $\phi_{i,j}^{[1]} \sim \mathrm{Beta}(1,1)$, $i\in\mathcal{N}$ ;
2  for $t = 1 : T$ do
3      for $i = 1 : N$ do
4          Draw a sample $\hat{\phi}_{i,j}^{[t]}$ from the distribution
             $\phi_{i,j}^{[t]} \sim \mathrm{Beta}(\alpha_{i,j}^{[t]},\beta_{i,j}^{[t]})$ ;
5      end
6      Play arm $i^{*[t]} = \arg\max_i \hat{\phi}_{i,j}^{[t]}$ and observe reward $r_j^{[t]}$ ;
7      Perform a Bernoulli trial with success probability $r_j^{[t]}$
         and record outcome $\tilde{r}_j^{[t]}$ ;
8      $\alpha_{i^*,j}^{[t+1]} = \gamma\alpha_{i^*,j}^{[t]} + \tilde{r}_j^{[t]}$ and $\beta_{i^*,j}^{[t+1]} = \gamma\beta_{i^*,j}^{[t]} + (1-\tilde{r}_j^{[t]})$ ;
9      $\alpha_{i,j}^{[t+1]} = \gamma\alpha_{i,j}^{[t]}$ and $\beta_{i,j}^{[t+1]} = \gamma\beta_{i,j}^{[t]}$, $\forall i \ne i^*$ ;
10 end
```

with probability $r_j^{[t]}$ and observes the outcome $\tilde{r}_j^{[t]}$. Finally, the distribution of $\phi_{i,j}^{[t]}$ is updated by:

$$(\alpha_{i,j}^{[t+1]},\beta_{i,j}^{[t+1]}) = \begin{cases} (\alpha_{i,j}^{[t]}+1,\beta_{i,j}^{[t]}), & \text{if } i=i^* \,\&\, \tilde{r}_j^{[t]}=1 \\ (\alpha_{i,j}^{[t]},\beta_{i,j}^{[t]}+1), & \text{if } i=i^* \,\&\, \tilde{r}_j^{[t]}=0 \\ (\alpha_{i,j}^{[t]},\beta_{i,j}^{[t]}), & \text{if } i\ne i^* \end{cases} \tag{41}$$

The process for learning $\boldsymbol{\theta}_j^{[t]}$ is summarized in Algorithm 2.

Same as Algorithm 1, we derive an upper bound for the expected total regret of Algorithm 2. The total regret is given by:

$$\mathbb{E}[\mathcal{R}'(T)] = \mathbb{E}\left[\sum_{t=1}^{T}\left(\theta_{i^{*[t]},j} - \theta_{i^{[t]},j}\right)\right] = \sum_i \delta_i'\mathbb{E}\left[\kappa_i'(T)\right] \tag{42}$$

where $i^{[t]}$ is the arm played at time t and $i^{*[t]}$ is the optimal arm at time t; $\delta_i' = \theta_{i^{*[t]},j} - \theta_{i^{[t]},j}$, and $\kappa_i'(T)$ is the number of times that arm i is played over $t=1,\ldots,T$. Let $C(T)$ be the maximum number of times of environment changes in $t=1,...,T$.

Theorem 3. *The expected regret of Algorithm 2 is upper bounded by:*

$$\mathbb{E}[\mathcal{R}'(T)] \le \mathcal{O}\left(\sqrt{\frac{NT\log T}{1-\gamma}} + \frac{C(T)\log T}{1-\gamma}\right). \tag{43}$$

The proof of Theorem 3 is in the same way as Theorem 2, which is presented in the second appendix.

C. Program Placement with Given User Association

With given $x_{k,j}^{[t]}$ $(k\in\mathcal{K})$ and $\phi_{i,j}^{[t]}$ $(i\in\mathcal{N})$, the program placement optimization of each EN j at time t is a Knapsack problem, which is generally NP-hard. Since such a problem needs to be solved at each time step, only a low-complexity solution can be implemented. To this end, we propose a heuristic program placement strategy that greedily allocates storage space to the programs with the highest request probability per occupied space. Specifically, we first sort the programs by the descending order of $\phi_{i,j}^{[t]}/(s_i+q_i)$ and put them into the RAM

Algorithm 3: Greedy Program Placement Algorithm

```
1  Initialize: $\Omega = \{1, \dots, N\}$; $a_i^{[t]} = b_i^{[t]} = 0$, $i = 1, \dots, N$ ;
2  while $\sum_{i=1}^{N} a_i^{[t]} s_i \le E_{\mathrm{H}}$ do
3      while $\sum_{i=1}^{N} b_i^{[t]} s_i \le E_{\mathrm{R}}$ do
4          $i_{\max} = \arg\max_{i\in\Omega} \theta_i^{[t]}/(s_i + q_i)$ ;
5          Set $a_{i_{\max}}^{[t]} = b_{i_{\max}}^{[t]} = 1$ ;
6          $\Omega = \Omega - \{i_{\max}\}$ ;
7      end
8      $i_{\max} = \arg\max_{i\in\Omega} \phi_i^{[t]}/s_i$ ;
9      Set $a_{i_{\max}}^{[t]} = 1$ ;
10     $\Omega = \Omega - \{i_{\max}\}$ ;
11 end
```

according to such order until the storage space of RAM is full, i.e., $\sum_{i=1}^{N} b_{i,j}^{[t]} q_i > E_{\mathrm{R}}$. Then, we sort the remaining programs by the descending order of $\phi_{i,j}^{[t]}/s_i$ and put them into the disk according to such order until the storage space of disk is full, i.e., $\sum_{i=1}^{N} a_{i,j}^{[t]} s_i > E_{\mathrm{H}}$. The procedure of the greedy program placement strategy of EN j is summarized in Algorithm 3.

The low complexity of Algorithm 3 comes at a price in terms of its performance compared to the optimal solution. However, the performance loss is limited, and is at most 50% of the optimal performance in the worst-case scenario [28]. Thus, the performance lower bound of Algorithm 3 is 50% of the optimal performance. Note that Algorithm 3 can also be applied to the program placement optimization in the single-EN scenario, especially in the case with large N. Specifically, at each time t, the EN learns the task popularity profile with Algorithm 2 and applies the greedy program placement strategy with Algorithm 3. Compared to the solution presented in Algorithm 1, the solution based on Algorithm 3 can achieve faster convergence, since the number of arms in the corresponding MAB problem is significantly reduced. Thus, such a solution is preferred in scenarios with large N and fast-varying task request patterns.

D. Dual Decomposition-Based User Association

Let $\tilde{\Delta}_{k,j}(\mathbf{x}^{[t]})$ be the value of $\Delta_{k,j}^{[t]}$ if the greedy program placement strategy in Section VI-C is applied under $\mathbf{x}^{[t]}$. The user association problem is formulated as:

$$\mathbf{P4}: \max_{\{\mathbf{x}^{[t]}\}} \sum_{k=1}^{K}\sum_{j=1}^{J} x_{k,j}^{[t]} \tilde{\Delta}_{k,j}(\mathbf{x}^{[t]}) \quad (44)$$
$$\text{s.t.:} \ (35) - (38)$$

Problem **P4** is an integer programming problem, which is generally NP-hard. To develop an effective solution algorithm, we first relax the integer constraint by allowing all $\mathbf{x}^{[t]}$ to take values in $[0, 1]$. Let **P4-Relaxed** be the relaxed problem, it can be verified that the objective function of **P4-Relaxed** is concave and the feasible region defined by all constraints is convex. Thus, **P4-Relaxed** is a convex optimization problem, and we apply a dual decomposition approach to obtain its optimal solution.

The decision variables of **P4-Relexted** are coupled with each other via the quadratic terms in the objective function. To this end, we introduce a set of auxiliary variables $V_j^{[t]} = \sum_{k=1}^{K} x_{k,j}^{[t]}$, $j \in \mathcal{J}$, which are the traffic loads of ENs. At each iteration of the dual decomposition algorithm, we first fix the traffic loads and find the optimal solution of user association. The solution is then used to update the traffic loads, which will be used in the next iteration. With given $\{V_1^{[t]}, \dots, V_J^{[t]}\}$, we have the following problem:

$$\mathbf{P5}: \max_{\{\mathbf{x}^{[t]}\}} \sum_{k=1}^{K}\sum_{j=1}^{J} x_{k,j}^{[t]} \tilde{\Delta}_{k,j}(\mathbf{x}^{[t]}) \quad (45)$$
$$\text{s.t.:} \ (35) - (37)$$
$$\sum_{k=1}^{K} x_{k,j}^{[t]} = V_j^{[t]}, \ j \in \mathcal{J}, \quad (46)$$
$$x_{k,j}^{[t]} \in \{0, 1\}, \ k \in \mathcal{K}, \ j \in \mathcal{J}. \quad (47)$$

Let $\boldsymbol{\lambda}^{[t]} = [\lambda_1^{[t]}, \dots, \lambda_J^{[t]}]$ be the Lagrangian multipliers for the constraints (46). Applying a partial relaxation on these constraints, we have the following Lagrangian function:

$$\mathcal{L}\left(\mathbf{x}^{[t]}, \boldsymbol{\lambda}^{[t]}\right) = \sum_{k=1}^{K}\sum_{j=1}^{J} x_{k,j}^{[t]} \tilde{\Delta}_{k,j}(\mathbf{x}^{[t]}) + \sum_{j=1}^{J} \lambda_j^{[t]} \left(\sum_{k=1}^{K} x_{k,j}^{[t]} - V_j^{[t]}\right). \quad (48)$$

The corresponding dual problem of **P5** is given by:

$$\textbf{P5-Dual:} \ \min_{\{\boldsymbol{\lambda}^{[t]}\}} g(\boldsymbol{\lambda}^{[t]}) \quad (49)$$

where $g(\boldsymbol{\lambda}^{[t]})$ is given by:

$$g(\boldsymbol{\lambda}^{[t]}) = \max_{\{\mathbf{x}^{[t]}\}} \mathcal{L}\left(\mathbf{x}^{[t]}, \boldsymbol{\lambda}^{[t]}\right). \quad (50)$$

The problems given in (49) and (50) are solved iteratively. At each iteration, $\mathbf{x}^{[t]}$ and $\boldsymbol{\lambda}^{[t]}$ are calculated and updated by UEs and ENs, respectively.

Based on the expression of $\mathcal{L}(\mathbf{x}^{[t]}, \boldsymbol{\lambda}^{[t]})$, the problem of maximizing $g(\boldsymbol{\lambda}^{[t]})$ can be decomposed into K independent subproblems that are solved by each UE. Let $\tau = 1, 2, \dots$ be the index of iteration for the dual decomposition algorithm.[5] At the τth iteration, each UE solves its subproblem by selecting the EN $j^{*[t]}(\tau)$ that satisfies:

$$j^{*[t]}(\tau) = \arg\max_{j\in\pi_k} \left\{\tilde{\Delta}_{k,j}(\mathbf{x}^{[t]}(\tau)) - \lambda_j^{[t]}(\tau)\right\}. \quad (51)$$

where $j^{*[t]}(\tau)$, $\mathbf{x}^{[t]}(\tau)$, and $\lambda_j^{[t]}(\tau)$ are the optimal selection of $j^{[t]}$, the matrix $\mathbf{x}^{[t]}$, and value of $\lambda_j^{[t]}$ at iteration τ, respectively.

After completing the selection, each UE notifies the selected EN via a message. Receiving the messages from UEs, each EN updates $\mathbf{x}_j^{[t]} = [x_{1,j}^{[t]}, \dots, x_{K,j}^{[t]}]$ by:

$$x_{k,j}^{[t]}(\tau) = \begin{cases} 1, & j = j^{*[t]}(\tau) \\ 0, & \text{otherwise,} \end{cases} \quad (52)$$

On the other hand, Problem **P5-Dual** can be decomposed into J subproblems, each to be solved by the corresponding EN. At iteration τ, each EN updates $\lambda_j^{[t]}(\tau)$ by:

$$\lambda_j^{[t]}(\tau + 1) = \lambda_j^{[t]}(\tau) - \varepsilon_j^{[t]}(\tau)\varphi_j^{[t]}(\tau) \quad (53)$$

[5]The updates indexed by $\tau = 1, 2, \dots$ are performed within each time period t, i.e., the updates indexed by $t = 1, \dots, T$ are the outer loop that is performed with a larger time scale.

Algorithm 4: Dual Decomposition-Based User Association Algorithm

```
1  Initialize $\mathbf{V}^{[t]}$ and $\boldsymbol{\lambda}^{[t]}$ ;
2  do
3      for $k = 1 : K$ do
4          UE $k$ selects EN according to (51) and notifies the
           selected EN ;
5      end
6      for $j = 1 : J$ do
7          EN $j$ updates $\mathbf{x}_j^{[t]}$ with (52) ;
8          Updates $\varphi_j^{[t]}$ with (54) ;
9          Updates $\lambda_j^{[t]}$ with (53) ;
10         Updates $V_j^{[t]}$ with (56) ;
11     end
12     $\tau++$
13 while ($\mathbf{x}^{[t]}$ does not converge);
```

where $\varphi_j^{[t]}(\tau)$ is the gradient of $\lambda_j^{[t]}(\tau)$, given by:

$$\varphi_j^{[t]}(\tau) = V_j^{[t]}(\tau) - \textstyle\sum_{k=1}^{K} x_{k,j}^{[t]}(\tau) \tag{54}$$

where $\varepsilon_j^{[t]}(\tau)$ is the step size, given by:

$$\varepsilon_j^{[t]}(\tau) = \frac{g(\boldsymbol{\lambda}^{[t]}(\tau)) - g(\boldsymbol{\lambda}^{*[t]})}{\left\|\boldsymbol{\varphi}^{[t]}(\tau)\right\|^2}. \tag{55}$$

With the updated $\lambda_j^{[t]}(\tau)$, each EN updates $V_j^{[t]}(\tau)$ by:

$$V_j^{[t]}(\tau+1) = \min\left\{\textstyle\sum_{k=1}^{K} x_{k,j}^{[t]}(\tau), U_j\right\}. \tag{56}$$

Finally, all ENs broadcast the updated values of $\lambda_j^{[t]}(\tau)$ and $V_j^{[t]}(\tau)$. Each UE then performs EN selection for the next iteration with (51). The updates performed by UEs and ENs continue until convergence is achieved. The dual decomposition-based user association algorithm is summarized in Algorithm 4.

The solution obtained by solving **P5** using Algorithm 4 is expected to be near-optimal for **P4**. As presented above, **P4** is transformed into **P5** in two steps: relaxing the integer constraint and introducing the auxiliary variables $V_j^{[t]} = \sum_{k=1}^{K} x_{k,j}^{[t]}$. In the first step, although the integer constraint is relaxed when solving **P5**, all the solutions for **P5** obtained by Algorithm 4 are binary integers, which are directly used as the solutions for **P4** without any rounding operations. Thus, the first step does not cause performance loss. In the second step, the performance loss is determined by the optimality of $V_j^{[t]}$, i.e., the traffic loads at various ENs. Note that, the optimal solution of **P4** can be obtained by exhaustively solving **P5** under all possible combinations of $\{V_1^{[t]}, ..., V_J^{[t]}\}$, which is computationally prohibitive for MEC systems since **P5** needs to be solved $\prod_{j=1}^{J} U_j$ times. In the proposed solution, the optimality of $\{V_j^{[t]}\}$ depends on dynamic, network-specific factors that cannot be accurately characterized. These factors include the locations of ENs and the spatial distributions of users over time. Thus, the exact performance gap cannot be accurately quantified. However, an important property of Algorithm 4 is that each $V_j^{[t]}$ will always approach its optimal value as the algorithm proceeds iteratively. Specifically, when $V_j^{[t]}$ (the number of UEs associated with EN j at time t) is larger than its optimal value, $\lambda_j^{[t]}$ (the Lagrangian variable for EN j) will be increased, forcing fewer UEs to select EN j in the next iteration; and vice versa when $V_j^{[t]}$ is smaller than its optimal value. As a result, the value of each $V_j^{[t]}$ stays close to its optimal value, especially when the spatial distribution of UEs varies slowly. Based on the above analysis for the two steps that transform **P4** into **P5**, the solution of **P5** is a near-optimal solution for **P4**.

Lemma 1. *Algorithm 4 converges with a rate faster than the sequence* $\{1/\sqrt{\tau}\}$.

Proof. The optimality gap of $\boldsymbol{\lambda}^{[t]}$ satisfies:

$$\begin{aligned}
&\|\boldsymbol{\lambda}^{[t]}(\tau+1)) - \boldsymbol{\lambda}^{*[t]}\|^2 \\
&= \left\|\boldsymbol{\lambda}^{[t]}(\tau) - \frac{g(\boldsymbol{\lambda}^{[t]}) - g(\boldsymbol{\lambda}^{*})}{\|\boldsymbol{\varphi}^{[t]}\|^2}\boldsymbol{\varphi}^{[t]} - \boldsymbol{\lambda}^{*[t]}\right\|^2 \\
&= \left\|\boldsymbol{\lambda}^{[t]}(\tau) - \boldsymbol{\lambda}^{*[t]}\right\|^2 + \left(\frac{g(\boldsymbol{\lambda}^{[t]}(\tau)) - g(\boldsymbol{\lambda}^{*[t]})}{\|\boldsymbol{\varphi}^{[t]}(\tau)\|^2}\boldsymbol{\varphi}^{[t]}(\tau)\right)^2 \left\|\boldsymbol{\varphi}^{[t]}\right\|^2 \\
&\quad - 2\Big(\boldsymbol{\lambda}^{[t]}(\tau) - \boldsymbol{\lambda}^{*[t]}\Big)^{\mathrm{T}} \frac{g\left(\boldsymbol{\lambda}^{[t]}(\tau)\right) - g\left(\boldsymbol{\lambda}^{*[t]}\right)}{\|\boldsymbol{\varphi}^{[t]}(\tau)\|^2}\boldsymbol{\varphi}^{[t]}(\tau) \\
&\overset{(a)}{\leq} \|\boldsymbol{\lambda}^{[t]}(\tau)) - \boldsymbol{\lambda}^{*[t]}\|^2 - 2\frac{\left(g\left(\boldsymbol{\lambda}^{[t]}(\tau)\right) - g\left(\boldsymbol{\lambda}^{*[t]}\right)\right)^2}{\|\boldsymbol{\eta}^{[t]}(\tau)\|^2} \\
&\quad + \left(\frac{g\left(\boldsymbol{\lambda}^{[t]}(\tau)\right) - g\left(\boldsymbol{\lambda}^{*[t]}\right)}{\|\boldsymbol{\eta}^{[t]}(\tau)\|^2}\right)^2 \|\boldsymbol{\varphi}^{[t]}(\tau)\|^2 \\
&\leq \|\boldsymbol{\lambda}^{[t]}(\tau) - \boldsymbol{\lambda}^{*[t]}\|^2 - \frac{\left(g\left(\boldsymbol{\lambda}^{[t]}(\tau)\right) - g\left(\boldsymbol{\lambda}^{*[t]}\right)\right)^2}{\hat{\boldsymbol{\varphi}}^2}.
\end{aligned}$$

Inequality (a) is due to the convexity of problem **P5-dual**, given by $g\left(\boldsymbol{\lambda}^{[t]}(\tau)\right) - g\left(\boldsymbol{\lambda}^{*[t]}\right) \leq \left(\boldsymbol{\lambda}^{[t]}(\tau) - \boldsymbol{\lambda}^{*[t]}\right)^{\mathrm{T}}\boldsymbol{\varphi}^{[t]}(\tau)$. Variable $\hat{\boldsymbol{\varphi}}$ is an upper bound for $\boldsymbol{\varphi}^{[t]}(\tau)$. Since $\lim_{\tau\to\infty} \boldsymbol{\lambda}^{[t]}(\tau+1) = \lim_{\tau\to\infty} \boldsymbol{\lambda}^{[t]}(\tau)$, we have $\lim_{\tau\to\infty} g\left(\boldsymbol{\lambda}^{[t]}(\tau)\right) = g\left(\boldsymbol{\lambda}^{*[t]}\right)$. Summing this inequality over τ, we have:

$$\sum_{\tau=1}^{\infty}\left(g\left(\boldsymbol{\lambda}^{[t]}(\tau)\right) - g\left(\boldsymbol{\lambda}^{*[t]}\right)\right)^2 \leq \hat{\boldsymbol{\varphi}}^2 \left\|\boldsymbol{\lambda}^{[t]}(1) - \boldsymbol{\lambda}^{*[t]}\right\|^2. \tag{57}$$

Suppose for contradiction, $g\left(\boldsymbol{\lambda}^{[t]}(\tau)\right)$ converges slower than $\{1/\sqrt{\tau}\}$. Then, $\lim_{\tau\to\infty}\sqrt{\tau}\left(g\left(\boldsymbol{\lambda}^{[t]}(\tau)\right) - g\left(\boldsymbol{\lambda}^{*[t]}\right)\right) > 0$. Therefore, there exists a positive number π and a sufficiently large τ' such that:

$$\sqrt{\tau}\left(g\left(\boldsymbol{\lambda}^{[t]}(\tau)\right) - g\left(\boldsymbol{\lambda}^{*[t]}\right)\right) \geq \pi, \forall \tau > \tau'. \tag{58}$$

Taking the square sum of (58) from τ' to ∞, we have:

$$\sum_{\tau=\tau'}^{\infty}\left(g\left(\boldsymbol{\lambda}^{[t]}(\tau)\right)-g\left(\boldsymbol{\lambda}^{*[t]}\right)\right)^2 \geq \pi^2\sum_{\tau=\tau'}^{\infty}\frac{1}{\tau}=\infty. \quad (59)$$

This contradicts the fact given in (57). We conclude that $g\left(\boldsymbol{\lambda}^{[t]}(\tau)\right)$ converges faster than the sequence $\{1/\sqrt{\tau}\}$. □

Lemma 2. *The complexity of Algorithm 4 in terms of the number of iterations is upper bounded by $1/\omega^2$, where ω is the threshold of convergence for $\boldsymbol{\lambda}^{[t]}$.*

Proof. According to Lemma 2, for a sufficiently large τ and a sufficiently small ω, $g\left(\boldsymbol{\lambda}^{[t]}(\tau)\right)-g\left(\boldsymbol{\lambda}^{*[t]}\right)$ is guaranteed to be smaller than ω. Thus, it takes less than $1/\omega^2$ steps for the sequence $\boldsymbol{\lambda}^{[t]}$ to achieve an optimality gap that is smaller than ω. □

VII. Simulation Results

In this section, we validate our proposed schemes with simulations. We consider multiple ENs and users randomly located in a 400 m $\times$ 400 m rectangular area. The channel is modeled by a distance-dependent path loss of $140.7+36.7\log_{10}d$ in dB and Rayleigh fading [7], where d is the distance between an EN and a UE in meters. The UE transmission power is 20 dBm, and the noise density is -174 dBm/Hz. The system bandwidth for uplink transmission is 10 MHz. The input data size follows a truncated normal distribution in the range of $[400, 600]$ Kb with a mean of 500 Kb. The complexity of a task is uniformly distributed in $[500, 1500]$ CPU cycles/bit. The computational capabilities of UEs and ENs are 1 GHz and 20 GHz, respectively. The storage spaces of disk and RAM of each EN are in the ranges of $[50, 200]$ GB and $[5, 10]$ GB, respectively. The size of each program follows a truncated normal distribution within the range of 100 MB around its mean. The number of programs N ranges from 100 to 500. Unless otherwise stated, the default mean program size and the number of programs are 500 MB and 200, respectively; the default storage spaces of disk and RAM are 100 GB and 8 GB, respectively. For each EN j at each time t, $\{\theta_{i,j}^{[t]}\}$ is generated by a Zipf distribution [31]. Let $W_1, ..., W_N$ be probabilities of N variables that follow a Zipf distribution, they are calculated by:

$$W_n = n^{-\zeta}/\textstyle\sum_{n=1}^{N} n^{-\zeta}, n = 1, ..., N. \quad (60)$$

To generate the task request pattern of EN j at time t (i.e., $\boldsymbol{\theta}_j^{[t]}$), each $\theta_{i,j}^{[t]}$ takes a distinct value from the set $\{W_1, ..., W_N\}$. Obviously, the mapping between $\{\theta_{i,j}^{[t]}\}$ and $\{W_n\}$ determines the task popularity pattern of EN j at time t. As mentioned, each $\boldsymbol{\theta}_j^{[t]}$ is in one of the $N!$ patterns, which is periodically updated with a periodicity of 1000 time slots. In (60), ζ is the parameter indicating the "skewness" of the distribution (i.e., task popularity). When $\zeta = 0$, the task popularity is uniformly distributed among all tasks. As ζ grows, the task's popularity becomes concentrated on a few popular tasks. We set the default value of ζ to be 0.2. At each round of the simulation, we first generate the parameters of each task i by sampling s_i, z_i, $\tilde{s}_{k,i}$, and $\theta_{i,j}^{[t]}$ according to the corresponding distributions. Then, each UE generates one of the N tasks according to the values of $\theta_{i,j}^{[t]}$. We focus on evaluating the long-term average latency of all users, which is the time-averaged cumulated average latency from $t = 1$ to $t = T$, i.e., $\frac{\sum_{t=1}^{T}\sum_{k=1}^{K} D_k^{[t]}}{T}$.

A. Single-EN Scenario

We first evaluate the performance of different schemes in the single-EN scenario. We make comparisons with two benchmark schemes. The first scheme is *random placement*, in which the programs are randomly selected and put into the disk and RAM of each EN. The second scheme is *greedy placement*, in which we first apply Algorithm 2 to obtain the task popularity at each time t, then apply the greedy program placement strategy in Section VI-C. Note that the complexity of the greedy scheme is similar to the proposed scheme, since both of them adopt DTS to learn the task popularity (one is direct and the other is indirect).

We first plot the average latency versus the number of UEs in Fig. 2(a). It can be seen that the average latency increases as the number of UEs grows, since both the communication and computing resources are shared among an increased number of UEs, resulting in higher offloading and computing time. The proposed scheme outperforms other schemes as the optimal program placement strategy obtained from the DTS algorithm is applied. It is also observed that the gaps between different schemes decrease as the number of users increases. This is because the computing and communication resources allocated to each user are reduced; hence, the latency reduction provided by offloading a task to the EN becomes less significant. The average latency under different values of ζ is plotted in Fig. 2(b). Due to similar reasons, the proposed scheme achieves the lowest average latency. As ζ grows, the latency of the random placement scheme increases while the latency of the other two schemes decreases. This is because when ζ is large, only a few tasks will be frequently requested, and the advantage of storing popular tasks becomes significant.

Fig. 2(c) shows the convergence property of different schemes. The greedy placement scheme converges faster, since it only needs to learn the task popularity, which corresponds to solving an MAB problem with N arms. In contrast, the proposed scheme needs to learn a subset of feasible program placement strategies by solving an MAB problem with a much larger number of arms, resulting in slow convergence. Note that the reward of the greedy placement scheme is lower than that of the proposed scheme after convergence.

B. Multi-EN Scenario

In the multi-EN scenario, we compare the proposed scheme with two benchmark schemes and a lower bound of latency. The first scheme is *random placement*, which is the same scheme as described in the single-EN scenario. The second scheme is *Heuristic UA*, in which each user is associated with the EN with the highest SINR. For a fair comparison, the proposed dual decomposition-based user association is applied

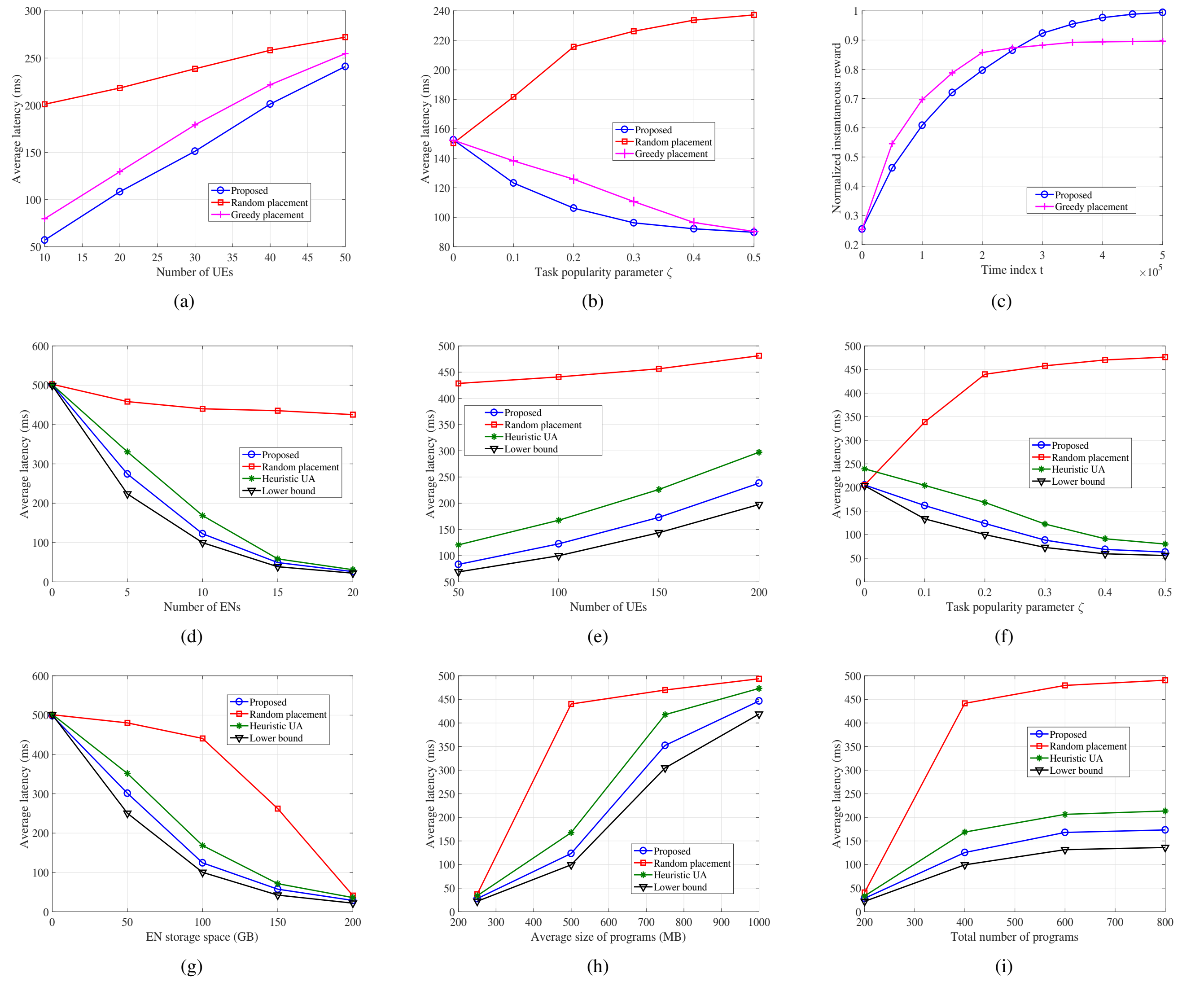

Fig. 2. Performance evaluation results. (a) Average latency vs. number of UEs for single-EN scenario. (b) Average latency vs. γ for single-EN scenario. The number of UEs is 20. (c) Convergence comparison between the proposed and greedy placement strategy. The number of programs is 50. (d) Average latency vs. number of ENs. (e) Average latency vs. number of UEs. (f) Average latency vs. γ for multi-EN scenario. (g) Average latency vs. EN storage space. (h) Average latency vs. average program size. (i) Average latency vs. total number of programs.

to the random placement scheme; the proposed program placement strategy is applied to the Heuristic UA scheme. The lower bound of latency is derived by relaxing all integer constraints in Problem **P3** and solving the resulting linear programming (LP). The value of the objective function under the optimal solution of the LP is a lower bound for the sum latency of all UEs.

The average latency versus the number of ENs is shown in Fig. 2(d). We observe that as the number of ENs grows, the average latencies of all schemes decrease, since more UEs can employ nearby ENs for task offloading. Without optimizing program placement, the random placement scheme achieves a quite limited latency reduction compared to other schemes. The proposed scheme outperforms the heuristic UA scheme because load balancing is achieved with our dual decomposition-based user association, which contributes to lower average latency. Furthermore, the performance of the proposed scheme is close to the lower bound, showing that our solution is near-optimal. The average latency versus the number of UEs in the area is shown in Fig. 2(e). We observe that the latencies of all schemes are increased as more UEs join the system. This happens because the communication and computational resources are shared among all UEs. Besides, ENs receive stronger aggregated interference caused by the uplink transmission of UEs served by neighboring ENs. The proposed scheme outperforms the other schemes since both user association and program placement are optimized.

The impact of ζ on average latency is presented in Fig. 2(f), where similar trends exhibited in the single-EN scenario are observed. In particular, when ζ is sufficiently large, the average latency of the proposed scheme becomes closer to the lower bound. This is because the task popularity is concentrated on a few tasks when ζ is large, hence the greedy program placement strategy presented in Section VI-C is highly likely to be optimal. The average latency versus the storage space of EN is shown in Fig. 2(g). As the storage space increases, the latencies of our proposed scheme and heuristic UA scheme drop much faster than the random placement scheme, since the storage space is allocated to the programs that are frequently requested and occupy relatively small storage space, resulting in improved utilization. When the storage space is sufficiently large, the tension caused by limited storage is mitigated, and all schemes can achieve relatively low latency.

The average latency versus the average program size is shown in Fig. 2(h). When the average program size is small, the ENs are able to store most of the programs, hence the latencies of all schemes are low. As the programs get larger, the latencies of the proposed scheme and heuristic UA scheme grow at a slower rate than the random placement scheme, due to the optimized program placement. The average latency versus the total number of programs is shown in Fig. 2(i), where similar trends are observed.

VIII. Conclusion

In this paper, we investigated the problem of optimizing program placement and user association in storage-constrained MEC systems. We formulated such a problem as a sequential decision-making problem. We first considered the single-EN scenario and proposed an MAB-based solution to derive the optimal solution. Then, we proposed a solution framework for the multi-EN scenario, in which we decomposed the original problem into three subproblems and solved them iteratively. Simulation results show that the proposed schemes reduce the average latency by $30\%\sim70\%$.

References

[1] M. Feng and M. Krunz, “Program placement optimization for storage-constrained mobile edge computing systems: A multi-armed bandit approach,” in *IEEE Proc. WoWMoM’21,* Online, June 2021.

[2] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing: A key technology towards 5G,” *ETSI White Paper,* vol. 11, 2015.

[3] M. Chen and Y. Hao, “Task offloading for mobile edge computing in software defined ultra-dense network,” *IEEE J. Sel. Areas Commun.,* vol. 36, no. 3, pp. 587–597, Mar. 2018.

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

[5] A. Ndikumana, N. H. Tran, T. M. Ho, Z. Han, W. Saad, D. Niyato, and C. S. Hong, “Joint communication, computation, caching, and control in big data multi-access edge computing,” *IEEE Trans. Mobile Comput.,* vol. 19, no. 6, pp. 1359–1374, June 2020.

[6] I. Blair, “Mobile app download and usage statistics (2019),” [Online]. Available: https://buildfire.com/app-statistics/

[7] T. X. Tran and D. Pompili, “Joint task offloading and resource allocation for multi-server mobile-edge computing networks,” *IEEE Trans. Veh. Technol.,* vol. 68, no. 1, pp. 856–868, Jan. 2019.

[8] 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, Mar. 2017.

[9] N. Golrezaei, A. F. Molisch, A. G. Dimakis, and G. Caire, “Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution,” *IEEE Commun. Mag.,* vol. 51, no. 4, pp. 142–149, Apr. 2013.

[10] W. Jiang, G. Feng and S. Qin, “Optimal cooperative content caching and delivery policy for heterogeneous cellular networks,” *IEEE Trans. Mobile Comput.,* vol. 16, no. 5, pp. 1382–1393, May 2017.

[11] S. Wang. T. Wang, and X. Cao, “In-network caching: An efficient content distribution strategy for mobile networks,” *IEEE Wireless Commun.,* vol. 26, no. 5, pp. 84–90, Oct. 2019.

[12] Z. Chang, L. Lei, Z. Zhou, S. Mao, and T. Ristaniemi, “Learn to cache: Machine learning for network edge caching in the big data era,” *IEEE Wireless Commun.,* vol. 25, no. 3, pp. 28–35, June 2018.

[13] B. N. Bharath, K. G. Nagananda, and H. V. Poor, “A learning-based approach to caching in heterogeneous small cell networks,” *IEEE Trans. Commun.,* vol. 64, no. 4, pp. 1674–1686, Apr. 2016,

[14] P. Blasco and D. Gündüz, “Learning-based optimization of cache content in a small cell base station,” in *Proc. ICC’14,* Sydney, Australia, June 2014, pp. 1897–1903.

[15] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, “Cooperative content caching in 5G networks with mobile edge computing,” *IEEE Wireless Commun.,* vol. 25, no. 3, pp. 80–87, June 2018.

[16] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, “In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning,” *IEEE Netw.,* vol. 33, no. 5, pp. 156–165, Sept./Oct. 2019.

[17] D. T. Hoang, D. Niyato, D. N. Nguyen, E. Dutkiewicz, P. Wang, and Z. Han, “A dynamic edge caching framework for mobile 5G networks,” *IEEE Wireless Commun.,* vol. 25, no. 5, pp. 95–103, Oct. 2018.

[18] L. T. Tan and R. Q. Hu, “Mobility-aware edge caching and computing framework in vehicle networks: A deep reinforcement learning,” *IEEE Trans. Veh. Tech.,* vol. 67, no. 11, pp. 10190–10203, Nov. 2018.

[19] Q. Ye, B. Rong, Y. Chen, M.A.-Shalash, C. Caramanis, and J. G. Andrews, “User association for load balancing in heterogeneous cellular networks,” *IEEE Trans. Wireless Commun.,* vol. 12, no. 6, pp. 2706–2716, June 2013.

[20] S. Sardellitti, M. Merluzzi, and S. Barbarossa, “Optimal association of mobile users to multi-access edge computing resources,” in *Proc. IEEE ICC’18,* Kansas City, MO, May 2018, pp. 1–6.

[21] M. Feng, M. Krunz, and W. Zhang, “Joint task partitioning and user association for latency minimization in mobile edge computing networks,” *IEEE Trans. Veh. Technol.,* vol. 70, no. 8, pp. 8108–8121, Aug. 2021.

[22] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, “Joint computation offloading and user association in multi-task mobile edge computing,” *IEEE Trans. Veh. Technol.,* vol. 67, no. 12, pp. 12313–12325, Oct. 2018.

[23] L. -T. Hsieh, H. Liu, Y. Guo, and R. Gazda, “Deep reinforcement learning-based task assignment for cooperative mobile edge computing,” *IEEE Trans. Mobile Comput.,* vol. 23, no. 4, pp. 3156–3171, Apr. 2024.

[24] H. Li et al., “Intelligent content caching and user association in mobile edge computing networks for smart cities,” *IEEE Trans. Network Sci. Eng.,* vol. 11, no. 1, pp. 994–1007, Jan.-Feb. 2024.

[25] A. Nabi and S. Moh, “Joint offloading decision, user association, and resource allocation in hierarchical aerial computing: Collaboration of UAVs and HAP,” *IEEE Trans. Mobile Comput.,* vol. 24, no. 8, pp. 7267–7282, Aug. 2025.

[26] L. Zhong, Y. Li, M. -F. Ge, M. Feng, and S. Mao, “Joint task offloading and resource allocation for LEO satellite-based mobile edge computing systems with heterogeneous task demands,” *IEEE Trans. Veh. Tech.,*, vol. 74, no. 7, pp. 11337–11352, July 2025.

[27] S. Lyu, M. Feng, L. Xiao, J. Zhou, and T. Jiang, “Intelligent task offloading and resource allocation for NOMA-based multi-beam satellite mobile edge computing systems,” *IEEE Trans. Wireless Commun.,*, to appear. DOI: 10.1109/TWC.2025.3608832.

[28] E. L. Lawler, “Fast approximation algorithms for knapsack problems,” 18th Annual Symposium on Foundations of Computer Science, Providence, RI, USA, 1977, pp. 206-213.

[29] S. Agrawal and N. Goyal, “Analysis of Thompson sampling for the multi-armed bandit problem,” in *Proc. of the 25th Annual Conference On Learning (COLT),* vol. 23, pages 39.1–39.26, June 2012.

[30] R. Vishnu and S. Kalyani, “Taming non-stationary bandits: A Bayesian approach,” *arXiv preprint,* arXiv:1707.09727, July 2017.

[31] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “Web caching and Zipf-like distributions: Evidence and implications,” *in Proc. IEEE INFOCOM’99,* vol. 1, pp. 126–134.

Appendix

In this part, we present the proof of Theorem 1. We first analyze the number of vectors $\left[a_{1,j}^{[t]}, ..., a_{N,j}^{[t]}\right]$ that satisfy steps (a) and (b). In step (b), adding any one more program would result in a violation of the storage constraint (15), meaning that even adding the smallest program $\min\{s_i\}$ would violate (15). Thus, $\sum_{i=1}^{N} a_{i,j}^{[t]} s_i$ must satisfy:

$$E_{\mathrm{H}} - \min\{s_i\} < \sum_{i=1}^{N} a_{i,j}^{[t]} s_i \leq E_{\mathrm{H}}. \tag{61}$$

Let $A = \{i \mid a_{i,j}^{[t]} = 1\}$ and $P_A = \sum_{i \in A} s_i$. Then, P_A must satisfy:

$$E_{\mathrm{H}} - \min\{s_i\} < P_A \leq E_{\mathrm{H}}. \quad (62)$$

Let $|A|$ be the number of elements in A. Then, P_A must also satisfy:

$$|A| \cdot \min\{s_i\} \leq P_A \leq |A| \cdot \max\{s_i\}. \quad (63)$$

Combining (62) and (63), $|A|$ satisfies:

$$\left\lceil \frac{E_{\mathrm{H}} - \min\{s_i\}}{\max\{s_i\}} \right\rceil \leq |A| \leq \left\lfloor \frac{E_{\mathrm{H}}}{\min\{s_i\}} \right\rfloor. \quad (64)$$

Given the range of $|A|$, the number of $\left[a_{1,j}^{[t]}, ..., a_{N,j}^{[t]}\right]$ that satisfy steps (a) and (b) is upper bounded by $\sum_{y=\left\lceil \frac{E_{\mathrm{H}} - \min\{s_i\}}{\max\{s_i\}} \right\rceil}^{\left\lfloor \frac{E_{\mathrm{H}}}{\min\{s_i\}} \right\rfloor} \binom{N}{y}$.

We then analyze the number of vectors $\left[b_{1,j}^{[t]}, ..., b_{N,j}^{[t]}\right]$ that satisfy steps (a) and (b). Let $B = \{i \mid b_{i,j}^{[t]} = 1\}$, $P_B = \sum_{i \in B} q_i$, and denote $|B|$ be the number of elements in B. Due to constraint (17), we have $B \subseteq A$. Similar to (62), P_B must satisfy:

$$E_{\mathrm{R}} - \min\{q_i\} < P_B \leq E_{\mathrm{R}}. \quad (65)$$

Similar to (64), $|B|$ satisfies:

$$\left\lceil \frac{E_{\mathrm{R}} - \min\{q_i\}}{\max\{q_i\}} \right\rceil \leq |B| \leq \left\lfloor \frac{E_{\mathrm{R}}}{\min\{q_i\}} \right\rfloor. \quad (66)$$

Since $B \subseteq A$, only $|A|$ elements in $\left[b_{1,j}^{[t]}, ..., b_{N,j}^{[t]}\right]$ can take values in $\{0, 1\}$, while the rest must be 0. Thus, the number of $\left[b_{1,j}^{[t]}, ..., b_{N,j}^{[t]}\right]$ that satisfy steps (a) and (b) is upper bounded by $\sum_{y=\left\lceil \frac{E_{\mathrm{R}} - \min\{q_i\}}{\max\{q_i\}} \right\rceil}^{\left\lfloor \frac{E_{\mathrm{R}}}{\min\{q_i\}} \right\rfloor} \binom{|A|}{y}$. Since $|A| \leq \left\lfloor \frac{E_{\mathrm{H}}}{\min\{s_i\}} \right\rfloor$, the bound is relaxed to $\sum_{y=\left\lceil \frac{E_{\mathrm{R}} - \min\{q_i\}}{\max\{q_i\}} \right\rceil}^{\left\lfloor \frac{E_{\mathrm{R}}}{\min\{q_i\}} \right\rfloor} \binom{\left\lfloor \frac{E_{\mathrm{H}}}{\min\{s_i\}} \right\rfloor}{y}$.

Finally, the number of vectors $\left[a_{1,j}^{[t]}, ..., a_{N,j}^{[t]}, b_{1,j}^{[t]}, ..., b_{N,j}^{[t]}\right]$ that satisfy steps (a) and (b) is upper bounded by:

$$M \leq \left(\sum_{y=\left\lceil \frac{E_{\mathrm{H}} - \min\{s_i\}}{\max\{s_i\}} \right\rceil}^{\left\lfloor \frac{E_{\mathrm{H}}}{\min\{s_i\}} \right\rfloor} \binom{N}{y} \right) \cdot \left(\sum_{y=\left\lceil \frac{E_{\mathrm{R}} - \min\{q_i\}}{\max\{q_i\}} \right\rceil}^{\left\lfloor \frac{E_{\mathrm{R}}}{\min\{q_i\}} \right\rfloor} \binom{\left\lfloor \frac{E_{\mathrm{H}}}{\min\{s_i\}} \right\rfloor}{y} \right). \quad (67)$$

Appendix

In this part, we present the proof of Theorems 2 and 3. We use Algorithm 2 as an example, and the proof can also be applied to Algorithm 1. Without loss of generality, we drop the subscript j in the following expressions.

For each arm i, define the discounted sum reward up to time t as:

$$S_i^{[t]} = \sum_{s=1}^{t} \gamma^{t-s} r_i^{[s]} \cdot \mathbb{1}_{\{f_s = i\}} \quad (68)$$

where $r_i^{[s]}$ is reward of arm i at time s. Specifically, $r_i^{[s]}$ is calculated by (40) if arm i is played at time s. $\mathbb{1}_{\{\cdot\}}$ is an indicator function that takes value 0 or 1, and f_s denotes the selected (played) arm at time s.

For each arm i, define the discounted pull count up to time t as:

$$N_i^{[t]} = \sum_{s=1}^{t} \gamma^{t-s} \cdot \mathbb{1}_{\{f_s = i\}}. \quad (69)$$

Apply Hoeffding's inequality to the discounted rewards, $S_i^{[t]}$ is a sub-Gaussian random variable with variance proxy $\frac{1}{4 N_i^{[t]\gamma}}$. Then, for any arm i and time t, the following inequality holds with probability $1 - \chi$:

$$|\hat{\theta_i^{[t]}} - \theta_i^{[t]}| \leq \sqrt{\frac{\log(2/\chi)}{2 N_i^{[t]\gamma}}}. \quad (70)$$

Let $\delta_i^{[t]} = \theta_{i^*}^{[t]} - \theta_i^{[t]}$, the number of times a suboptimal arm i is pulled satisfies:

$$\mathbb{E}[N_i^{[t]\gamma}] \leq \frac{2 \log T}{(\delta_i^{\min})^2} + \frac{C(T)}{1 - \gamma} \quad (71)$$

where $\delta_i^{\min} = \min_{t : \theta_i^{[t]} < \theta_{i^*}^{[t]}} \delta_i^{[t]}$. The first term in the right-hand side of (71) corresponds to the phase when the environment is stationary, which bounds the probability of pulling suboptimal arms during intervals between changes with (70); the second term is the number of additional pulls caused by the reset of the effective sample size when the environment changes.

For each suboptimal arm i, using (71), we have:

$$\sum_{t=1}^{T} \delta_i^{[t]} \cdot \mathbb{E}[\mathbb{1}_{\{f_t = i\}}] \leq \sum_{i=1}^{N} \left(\frac{2 \log T}{\delta_i^{\min}} + \delta_i^{\max} \cdot \frac{C(T)}{1 - \gamma} \right) \quad (72)$$

where $\delta_i^{\max} = \max_t \delta_i^{[t]}$.

Each environmental change introduces a regret proportional to the time needed to detect the change. Using the effective horizon $H = \frac{1}{1-\gamma}$, the total regret caused by environment change is upper bounded by $C(T) \cdot \frac{\log T}{1-\gamma}$.

For the worst-case $\delta_i^{\min} = \sqrt{\frac{\log T}{T}}$ (minimal detectable gap). Summing over all arms, the total regret is upper bounded by:

$$\mathbb{E}[\mathcal{R}'(T)] \leq \mathcal{O}\left(\sqrt{\frac{N T \log T}{1 - \gamma}} + \frac{C(T) \log T}{1 - \gamma} \right). \quad (73)$$

This completes the proof for the upper bound given in Theorems 2 and 3.

Mingjie Feng [S'15-M'24] is a Full Professor with Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, China. He is a recipient of Best Paper Award of Digital Communications & Networks, Woltosz Fellowship from Auburn University, and Best Reviewer of IEEE Transactions on Wireless Communications. He has served/is serving as an Associate Editor for several journals in wireless communications and networking, including IEEE Networking Letters and Digital Communications & Networks. He was/is a Technical Program Committee Member of various IEEE conferences, including IEEE INFOCOM, IEEE MASS, and IEEE ICC.

Marwan Krunz [S'93-M'95-SM'04-F'10] is a Regents Professor in ECE at the University of Arizona (UA) and the Edward & Maria Keonjian Endowed Chair in Electrical and Computer Engineering. He also holds a joint appointment as a Professor of Computer Science and is a member of the UA Cancer Center. He is the Deputy Center Director and Site Director of WISPER, an NSF/industry funded consortium of 3 universities and 13+ companies. WISPER's industry-focused research aims to provide solutions for secure and AI-enabled NextGen wireless systems. Previously, Dr. Krunz directed two graduated NSF/industry centers: the Broadband Wireless Access and Applications Center (2013-2024) and ConnectionOne (2008-2013). Both centers focused on wireless systems and circuits, with engagement of tens of companies and government labs. Dr. Krunz holds a courtesy appointment at University Technology Sydney. He previously held the Kenneth VonBehren Endowed Professorship in electrical and computer engineering. Dr. Krunz's research is in the fields of wireless communications, networking, and security, with recent focus on applying AI and machine learning techniques for protocol adaptation, resource management, and signal intelligence. He has published more than 350 journal articles and peer-reviewed conference papers, and is a named inventor on 13 patents. His latest h-index is 64. He is an IEEE Fellow, an Arizona Engineering Faculty Fellow, and an IEEE Communications Society Distinguished Lecturer (2013-2015). He received the NSF CAREER award. He served as the Editor-in-Chief for the IEEE Transactions on Mobile Computing. He also served as editor for numerous IEEE journals. He is/was a General Chair for several international conferences, including ICCSPA'26 WiOpt'23, WiOpt 2016 (vice-chair), and WiSec'12. He was also a TPC chair for INFOCOM'04, SECON'05, WoWMoM'06, and Hot Interconnects 9. Dr. Krunz was chief scientist/technologist for two startup companies that focus on wireless systems and networking.