doi:10.1093/comjnl/bxz027

Collaborative Task Offloading with Computation Result Reusing for Mobile Edge Computing

ZIKAI ZHANG[1], JIGANG WU[1*], LONG CHEN[1], GUIYUAN JIANG[2] AND SIEW-KEI LAM[2]

[1]*School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China*
[2]*School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798*
[*]*Corresponding author: asjgwucn@outlook.com*

The task offloading problem, which aims to balance the energy consumption and latency for Mobile Edge Computing (MEC), is still a challenging problem due to the dynamic changing system environment. To reduce energy while guaranteeing delay constraint for mobile applications, we propose an access control management architecture for 5G heterogeneous network by making full use of Base Station's storage capability and reusing repetitive computational resource for tasks. For applications that rely on real-time information, we propose two algorithms to offload tasks with consideration of both energy efficiency and computation time constraint. For the first scenario, i.e. the rarely changing system environment, an optimal static algorithm is proposed based on dynamic programming technique to get the exact solution. For the second scenario, i.e. the frequently changing system environment, a two-stage online algorithm is proposed to adaptively obtain the current optimal solution in real time. Simulation results demonstrate that the exact algorithm in the first scenario runs 4 times faster than the enumeration method. In the second scenario, the proposed online algorithm can reduce the energy consumption and computation time violation rate by 16.3% and 25% in comparison with existing methods.

Keywords: 5G; dynamic system environment; mobile edge computing; task offloading; computation reusing

Received 13 December 2017; revised 29 January 2019; editorial decision 4 March 2019
Handling editor: Gerard Parr

1. INTRODUCTION

Mobile applications are typically user-interactive and resource-hungry, and it requires quick response and leads to higher energy consumption. However, the general mobile devices have limited computation capabilities and battery power. Mobile Cloud Computing (MCC) [1, 2] is proposed to tackle this contradiction by migrating computational tasks from mobile devices to the cloud servers. In recent years, much attention has been paid to improving the efficiency of MCC by Mobile Edge Computing (MEC) [3–5] and 5G networks [6–8].

MEC is envisioned as a promising approach with low latency, high bandwidth and computing agility [9]. With the aid of MEC, mobile devices are able to offload their tasks to the MEC server on the edge of the network rather than utilizing the server in the core network.

5G network has become an important topic for the design and implementation of mobile computing, since the unprecedented increases of mobile data traffic begin to stress mobile operators' networks, weaken the communication quality and affect users' experience. As a kind of heterogeneous network, HetNet is applied to 5G network for Contents Distribution [10–12]. In this architecture, caches are used to store contents from the cloud side as a priori. But, they cannot be directly applied to cloud computing, in which data are generated in the mobile device, transmitted to the cloud side and computed in the cloud. To the best of our knowledge, most researchers have not paid enough attention to caching computation results in the cloud-devices. In most existing 5G architectures for cloud computing, such as [6, 7, 13], caches are used to caching contents or users, and servers are used for task computation. In [9] and [14], a combination of MEC and 5G network is proposed to provide cloud-computing capabilities and an IT service environment.

Task offloading policy plays a critical role in MEC, and it determines the efficiency and achievable computation performance [15]. For example, joint optimization offloading decision and resource allocation methods are proposed in [16] to reduce energy consumption for delay sensitive tasks. Due to the dynamic environment in the computing system, it is hard for task offloading policy to achieve the optimal performance. System environment contains many elements, such as the network environment, the computing environment and so on. Most of the existing works concentrated on dynamic changing network environment. In the existing works, the rarely changing network is treated as a stable environment, thus they typically utilize static algorithms such as integer linear programming approach [17] and dynamic programming based method [18] to slove problems. In [19], a time-constrained offloading policy is investigated by using a new algorithm called 'Dynamic Programming with Randomization' to get an approximate solution. For frequently changing network environment, much attention has been paid to both offline methods and online methods in recent years. Markov-decision process approaches [20–22] are used as offline methods. Lyapunov optimization techniques [23–25] and Receding Horizon methods [26] are used as the online method.

Summarizing these existing works, we obtain the fact that few works have considered the combination of MEC and 5G network, the computational reusing resource (the obtained task results) and base station's storage capability are not fully used in these network architectures. Different applications which consist of collaborative tasks from mobile devices can cooperate if there exists a duplicate computation result in the edge server. However, few works have studied the collaborative task topologies, while the dependency between tasks and the computation result reusing have significant impact to the data transmission. In addition, few works have developed an exact approach to solve the time-constrained task offloading problem, and the existing few works are relatively slow for rarely changing network environment. On the other hand, many works have considered the frequently changing network channel, but some dynamic changing elements (such as the computational reusing resource) in the system are not considered. Furthermore, offline method cannot get an energy efficiency solution in comparison to the online method, while online algorithm often leads to a large time violation rate (Time-Constraint Violation Rate).

The above weaknesses motivate us to investigate the problem of Collaborative task offloading with computation result reusing and time constraint in the dynamic changing environment of 5G network combined with MEC. As a latest development [27], multi-access edge computing is proved to scale well with the increasing number of computation tasks, as well as to reduce latency and energy consumption. In this paper, we first propose an Access Control management architecture to improve Multi-access Edge Computing presented in [14]. Then, we construct a static algorithm to generate the exact solution for rarely changing system environment. Moreover, we contribute a two-stage online algorithm to reduce energy consumption and time violation rate for frequently changing system environment.

We also investigate a new type of applications which requires more real-time information to work. The new type of applications is to be widely used in real life. For example, the latest vehicle navigation need real-time weather information and transportation information to adaptively find the best path for the current location. But, to the best of our knowledge, the task offloading in this type of applications has not been investigated so far, while the existing algorithms cannot be directly used for the task offloading. In addition, the real-time information requested by this new kind of applications can only be generated on the MEC server. These pieces of information are not related to previous tasks and bring negative influence to energy and time consumption. Therefore, we construct a new computing model to further improve the performance of the new type of applications.

Our contributions are summarized as follows:

- We propose a new graph model to characterize real-time downlink transmission from the edge server or the cloud to mobile devices, since real-time downlink transmission is critical for the designing of offloading policies.
- Then, the problem of collaborative task offloading with consideration of reusing computation results is proposed to minimize the energy consumption under the time constraint of the new type of applications which requires real-time information to work.
- For rarely changing system environment, we propose a Bellman equation with time constraint to develop an optimal algorithm based on dynamic programming technique. The proposed static algorithm is four times faster than the enumeration method in obtaining the exact solution.
- For the case of frequently changing system environment, we propose an online algorithm, which can reduce both energy consumption and time violation rate. The time violation rate of proposed two-stage online algorithm can be reduced by 25% compared with a modified offline algorithm [21], the energy consumption can also be reduced by 16.3% compared with online algorithm ARHOS [28].

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 introduces system models and problem definitions. Section 4 describes the details of our proposed algorithms. Experimental results and analysis are presented in Section 5, and Section 6 concludes this work.

2. SYSTEM MODEL AND PROBLEM DEFINITIONS

In this section, we model the computing system and formulate the time-constrained collaborative task offloading with computation result reusing problem.

2.1. System architecture

In the cell of 5G heterogeneous network equipped with MEC, there is a Macro Base Station (MBS) connected with MEC. Besides the MBS, there are some Small Base Stations (SBSs), whose service area is overlaid by that of the MBS. To reuse spectrum efficiently, both the MBS and the SBS operate in the same frequency band. The spectrum channel and the bandwidth of each channel are automatically assigned.

Inspired by Multi-access Edge Computing [14], we propose an Access Control management architecture in this paper. As shown in Fig. 1, each base station is equipped with an Access Controller (AC) [29]. ACs are divided into two levels, named level S and level M, respectively, where level S connects with SBS and level M connects with MBS. Since AC has a limited storage capability, we can use it to store and manage the fingerprints [30] (a kind of ID for tasks) of the computed tasks' results. Between a SBS and a MBS, there is a wired backhaul which relays the transmission from the SBS to the MBS. AC at level S can manage many ACs at level M with communication using wired backhaul. We ignore the time consumption of data transmission through wired backhaul since it can transmit a small amount of data very quickly.

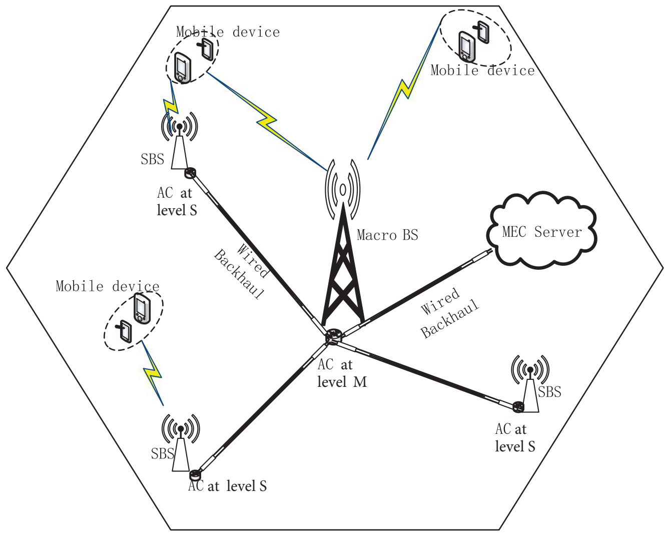

FIGURE 1. Access Control management architecture.

Fingerprints are integers generated by a one-way function [31] applied to a set of bytes. In this paper, we use fingerprints as pointers into the MEC server to find repeated tasks. We store representative fingerprints with efficient indexes that map a fingerprint to the region which describes a cached packet payload. If the fingerprint is stored on an AC, the corresponding computed results of certain tasks will be stored on MEC server and the results could potential be reused in future by other tasks. If the corresponding fingerprint is stored on an AC (i.e. tasks' results can be reused), we will download the result directly for further computation. This task's result reusing process is called as 'computation result reusing' in this paper.

In this paper, we mainly focus on task offloading. However, how to manage computation result reusing is one of the significant problems. Since it is out the scope of this work, we next provide a brief description for the management of computation result reusing in this paragraph. Since the fingerprint in AC is mapped to the reusable task's result, computation result reusing is controlled by managing and storing fingerprints. Due to the limited storage capacity of an AC, we need to efficiently manage and store fingerprints. Therefore, ACs are divided into two levels to management. AC at level M manages the whole area of the cell, while AC at level S just manages the local area in the cell. For fingerprint management, we first rank the request frequency of tasks on SBS in descending order and store the fingerprints on AC at level S by the ranking order. Then, for the rest of tasks that do not store their fingerprints on AC in level S, we rank the request frequency of them on MBS in descending order, and store the fingerprints on AC at level M by the ranking order. In this way, local space storage utility is managed by AC at level S and the whole space storage utility is managed by AC at level M. If fingerprints are not managed in these two levels, the local and the whole utility will be unbalanced. For example, when information of computing results is just stored and managed by SBS, some high ranked tasks' computing result in the whole area (managed by MBS) cannot be reused if they don't have high rank in each locality area managed by SBS.

Since the data size of the fingerprint is very small, the energy consumption and time overhead for sending requests to the AC can be ignored [29]. When a mobile device chooses to compute its task by the MEC server, it sends a request to AC to check whether the corresponding fingerprint has been stored. If the result has been stored, the results will be directly sent to the mobile device instead of being transmitted and computed again. In this way, the time consumption and energy consumption for task transmitting and recomputing can be saved. At the same time, the transmission stress of base station can be released. If the result has not been stored, the computation task will be sent to the MEC server for execution.

In [32], a cooperative multi-agent sequential decision model (MA-RDPG) is proposed to share the centralized learning results with private agents to guarantee the globally optimal performance of the entire system. This model achieves remarkable performance with online settings on a large E-commerce platform with applications such as retrieval, advertising and recommendation. We apply MA-RDPG scheme to our proposed access control management architecture to enable knowledge (results) reuse to improve system performance. For example, we can deploy the access control management architecture in the problem of guiding the tourism travel route in a scenic spot. Tourist with a mobile device is treated as an agent (actor), and agents collaborate with each other by sharing a global action-value function on the MEC as the critic. By centralized learning on the MEC, tourists make their private visit strategies on their mobile devices. When the results of global action-value functions (corresponding to specific visiting demands) can be found on MEC, these results are reusable to reduce energy consumption and time delay for each mobile device.

In this paper, we focus on a stochastic network environment since the dynamic changing transmission rate of data will significantly impact offloading decisions [33, 34]. We use the Gilbert–Elliott channel [34, 35] to model the communication link because it is used to refer to the wide class of finite-state fading channels. This channel model is a Markov chain with two states: good (denoted by g) or bad (denoted by b). If the channel is good, data are transmitted at a high rate (γ_g). Otherwise, it sends data at a low rate (γ_b) [36]. The transition between the two channel states occurs in discrete time instants so that the channel is assumed to stay in a given state over a single unit of time. Let $\gamma_n = g$, if the channel is good during the n-th time unit; and $\gamma_n = b$, otherwise. Hence, the state transition matrix of the channels Pr is given by [34]

$$Pr = \begin{bmatrix} Pr\left[\gamma_n = g|\gamma_{n-1} = g\right] & Pr\left[\gamma_n = g|\gamma_{n-1} = b\right] \\ Pr\left[\gamma_n = b|\gamma_{n-1} = g\right] & Pr\left[\gamma_n = b|\gamma_{n-1} = b\right] \end{bmatrix} = \begin{bmatrix} Pr_{gg} & Pr_{gb} \\ Pr_{bg} & Pr_{bb} \end{bmatrix} \tag{1}$$

If applications upload data and download data with different channels, previous works are not available. So, in this paper, we solve problems by separately modeling data uploading channel and data downloading channel. Let Pr_{up} and Pr_{down} be the state transition matrix of data uploading and downloading channels, respectively. Let γ^{up} and γ^{down} be the state of data uploading and downloading channels, respectively.

Unlike the existing work [34] where a VIA-based algorithm is proposed to minimize energy consumption, we propose a computing model with consideration of computation result reusing and real-time data in the downlink transmission. A static algorithm with time constraint is proposed based on dynamic programming. Moreover, we contribute an online algorithm to reduce energy consumption and time violation rate.

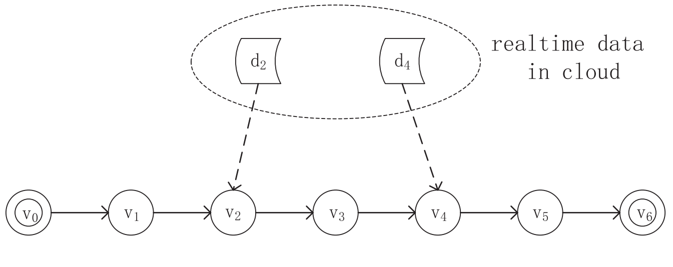

FIGURE 2. Collaborative task graph.

2.2. Computing model

A new type of applications that request the real-time information to compute is taken into consideration in this paper. First, applications could be represented by many kinds of topologies (sequential, mesh, tree, general, etc). But, they can be transferred to the sequential topology by critical path based methods, such as [37]. For general techniques on other topologies, see [38]. In order to model a relevant task graph, without loss of generality, this paper concentrate on sequential topology whose algorithms can be adapted to other topologies. Then, some tasks cannot be offloaded to remote servers [39], such as sensing tasks. But, in a collaborative sequential task topology, local execution only tasks could be treated as initial or terminal tasks. For general research, we just explore a sequential topology segment. And, only the initial and terminal tasks are fixed on the mobile device. Furthermore, the collaborative tasks are executed sequentially, and the output of a task will be taken as input by its following task. However, in order to ensure the real-time services, some tasks need some data obtained from the remote cloud. Those cloud-side data, such as traffic information and weather conditions, are dynamically acquired. Since these pieces of real-time information can only be generated on the cloud, normal topologies are not suitable to present this new kind of applications. So, we model a new task graph, as shown in Fig. 2.

Directed acyclic graph $G = (V, A)$ is used to represent an application, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of tasks and A is the set of arcs. The number of computation tasks in G is $n = |V|$. In addition, two dummy nodes, v_0 and v_{n+1}, are added into the graph to denote the initialization and termination of the application. The parametric context of each task is defined as a tuple $v_i = (y_i, \omega_i, d_i)$, where $y_i = 1$ indicates that the result of v_i is stored on the cloud and it can be reused, $y_i = 0$ indicates that the result of v_i cannot be reused, ω_i is the computation workload of the ith task, and d_i is the size of the data, which is transmitted between the cloud-side and task

v_i. We use arc $a_{ij}\,(a_{ij} \in A)$ to indicate the data dependency that task v_j cannot start before the completion of its parent task v_i. And β_{ij} is used to store the output data size from task v_i to task v_j.

Computation overhead of a task depends on its executive decision and whether its result has been computed and stored on the server. Notation $x_i = 1$ indicates that task v_i is executed on the cloud side, and $x_i = 0$ otherwise. We also define $X = \{x_0, x_1, x_2, \ldots, x_n, x_{n+1}\}$ as the offloading policy. As the initialization and the termination of the application must be executed on the mobile device, we have $x_0 = 0$ and $x_{n+1} = 0$. If $x_i = 0$, task v_i is executed on the mobile device and the time consumption should be determined by mobile computing capability.

We denote $T_{v_i}^{\text{comp}}(x_i, y_i)$ and $E_{v_i}^{\text{comp}}(x_i, y_i)$ as the computation time and the energy consumption of task v_i with respect to x_i and y_i. Formally,

$$T_{v_i}^{\text{comp}}(x_i, y_i) = \begin{cases} \frac{w_i}{f_m} & \text{if } x_i = 0, \\ 0 & \text{if } x_i = 1 \;\&\; y_i = 1, \\ \frac{w_i}{f_c} & \text{if } x_i = 1 \;\&\; y_i = 0, \end{cases} \tag{2}$$

where f_m and f_c are the computing ability for one CPU cycle of the mobile device and the edge server, respectively. And, energy consumption on the mobile device for the execution of task i is given by

$$E_{v_i}^{\text{comp}}(x_i, y_i) = \begin{cases} \frac{w_i p_m}{f_m} & \text{if } x_i = 0, \\ 0 & \text{if } x_i = 1 \;\&\; y_i = 1, \\ \frac{w_i p_{idle}}{f_c} & \text{if } x_i = 1 \;\&\; y_i = 0, \end{cases} \tag{3}$$

where p_m and p_{idle} are the energy consumption power of the mobile device in computation and the idle state, respectively.

Communication cost between tasks depends on where tasks are hosted, data transmission rate as well as whether tasks have been computed and stored on the cloud.

Let $T_{v_i,v_j}^{\text{comm}}(x_i, x_j, y_j)$ be the required communication time between task v_i and task v_j. Formally, when $a_{ij} \in A$, the communication time between task v_i and task v_j can be calculated as

$$T_{v_i,v_j}^{\text{comm}}(x_i, x_j, y_j) = \begin{cases} \dfrac{\beta_{ij}}{r} & \text{if } x_i = 0 \;\&\; x_j = 1 \;\&\; y_j = 0, \\ \dfrac{\beta_{ij} + d_j}{r} & \text{if } x_i = 1 \;\&\; x_j = 0, \\ \dfrac{d_j}{r} & \text{if } x_i = 0 \;\&\; x_j = 0, \\ 0 & \text{otherwise}, \end{cases} \tag{4}$$

where r is the data transmission rate.

Then, we denote $E_{v_i,v_j}^{\text{comm}}(x_i, x_j, y_j)$ as the communication energy on the mobile device for data transmission between task v_i and task v_j, which is given by

$$E_{v_i,v_j}^{\text{comm}}(x_i, x_j, y_j) = \begin{cases} \frac{\beta_{ij}}{r} p_{up} & \text{if } x_i = 0 \;\&\; x_j = 1 \;\&\; y_j = 0, \\ \frac{\beta_{ij} + d_j}{r} p_{\text{down}} & \text{if } x_i = 1 \;\&\; x_j = 0, \\ \frac{d_j}{r} p_{\text{down}} & \text{if } x_i = 0 \;\&\; x_j = 0, \\ 0 & \text{otherwise}, \end{cases} \tag{5}$$

where p_{up} and p_{down} are the power rate of the mobile device for sending and receiving data, respectively.

2.3. Problem formulation

Based on the above-discussed models, the overall energy consumption on mobile device, denoted as $E(X)$, is calculated as the sum of energy consumption on task execution as well as data transmission

$$E(X) = \sum_{v_i \in V} E_{v_i}^{\text{comp}}(x_i, y_i) + \sum_{(v_i,v_j) \in A} E_{v_i,v_j}^{\text{comm}}(x_i, x_j, y_j) \tag{6}$$

Let $T(X)$ be the time consumption of the application. it can be formally calculated as

$$T(X) = \sum_{v_i \in V} T_{v_i}^{\text{comp}}(x_i, y_i) + \sum_{(v_i,v_j) \in A} T_{v_i,v_j}^{\text{comm}}(x_i, x_j, y_j) \tag{7}$$

Based on the above discussion, the offloading problem can be formulated as follows:

$$\begin{array}{ll} Obj. & Min E(X) \\ S.t. & \\ (C_1): & T(X) \le t_d \\ (C_2): & X \in \{0,1\}^{n+1} \\ (C_3): & y_i \in \{0,1\}, i \in \{0,1,\ldots,n+1\} \end{array}$$

The objective is to minimize the value of energy consumption. C_1 is the constraint indicating that the running time of application should be no more than time deadline t_d, where t_d is a priori knowledge that presents the maximum time which users could endure for the given application. C_2 is the binary constraint for the offloading policy, where $X = \{x_0, x_1, x_2, \ldots, x_n, x_{n+1}\}$ and any task in the application can only be implemented on either the mobile device or MEC server. C_3 is the constraint indicating that whether tasks in the application could reuse their results which have been computed and stored on MEC server.

It is worthwhile to point out that the problem discussed in this subsection is NP-hard. This is because, the problem is identical to 0-1 knapsack problem if the time constraint and computation result reusing are omitted. Noting that 0-1 knapsack problem is NP-hard [40], we conclude that the problem in this subsection is also NP-hard.

3. PROPOSED ALGORITHM

In this section, we tend to solve offloading problems under two different kinds of system environment: rarely changing system environment and frequently changing system environment. For rarely changing system environment, a static algorithm is developed to get the exact optimal solution. For frequently changing system environment, static algorithms are not able to get good performance. So, we design a fast two-stage online algorithm to adapt to the evolving environment elements (such as the communication channel) and continuously updated computation results for reusing.

3.1. Optimal static algorithm

In this subsection, we aim to provide an exact optimal method that suitable to the rarely changing system environment. So, dynamic programming method is considered to provide good performance. In order to solve the problem optimally, we build a new dynamic programming algorithm STA with time constraint rather than executing enumeration method to get exact solution.

Firstly, let $t^i_{x_{i-1},x_i}$ be the sum of execution time of task v_i and data transmission time between v_{i-1} and v_i. If both task v_i and task v_{i-1} run on cloud or both of them run on the mobile device, $t^i_{x_{i-1},x_i}$ is the computing time of task v_i; otherwise, the data transmission time between task v_i and task v_{i-1} should be considered. The expression of $t^i_{x_{i-1},x_i}$ is

$$t^i_{x_{i-1},x_i} = \begin{cases} T^{\text{comp}}_{v_i}(x_i, y_i) & \text{if } x_i = x_{i-1}, \\ T^{\text{comp}}_{v_i}(x_i, y_i) \\ \quad + T^{\text{comm}}_{v_{i-1},v_i}(x_{i-1}, x_i, y_i) & \text{otherwise.} \end{cases} \tag{8}$$

Then, let $Q^i_{x_{i-1},x_i}$ be the value of execution cost of task v_i and communication cost between v_{i-1} and v_i

$$Q^i_{x_{i-1},x_i} = \begin{cases} E^{\text{comp}}_{v_i}(x_i, y_i) & \text{if } x_{i-1} = x_i, \\ (E^{\text{comp}}_{v_i}(x_i, y_i) \\ \quad + E^{\text{comm}}_{v_{i-1},v_i}(x_{i-1}, x_i, y_i) & \text{otherwise.} \end{cases} \tag{9}$$

Next, let $OPT(i, x_i, t)$ be the optimal value of the tradeoff function for the first $i+1$ tasks with time consumption t. x_i indicates offloading decision of task v_i. $OPT(i, x_i, t)$ recursively depends on $OPT(i-1, x_{i-1}, t - t^i_{x_{i-1},x_i})$.

In this paper, we propose an STA Bellman's equation with time constraint, which is an extension of the well-known shortest path algorithm of Ford and Bellman [41]. The STA Bellman's equation can be formulated as follows:

$$\begin{cases} OPT(0,0,t) = 0,\ OPT(0,1,t) = \infty \quad \text{for } i{=}0,\ 0{<}t < t_a, \\ OPT(i, x_i, t) = \min\limits_{x_{i-1}\in 0,1} \{OPT(i-1, x_{i-1}, t - t^i_{x_{i-1},x_i}/t_s) \\ \quad + Q^i_{x_{i-1},x_i}\} \quad \text{for } 0{<}i < n+1,\ 0{<}t < t_a, \\ OPT(n+1,0,t_a) = \min\limits_{x_n\in 0,1} \{OPT(n, x_n, t_a - t^{n+1}_{x_n,x_{n+1}}/t_s) \\ \quad + Q^{n+1}_{x_n,x_{n+1}}\} \quad \text{for } i = n+1. \end{cases} \tag{10}$$

As t_d is the QoE factor and given by user's demand, we can know the accuracy of time consumption deadline and the smallest time unit t_s (for example, if t_d is 1.20 s, t_s is set as 0.01 s.). Then, using t_s, we can get possible decision epoch number $t_a = t_d/t_s$, and use it to build time-related relations. As shown in Algorithm 1, we first initialize tasks to compute. Second, $OPT(i, x_i, t)$ is calculated by the nested for-loops using (10) and $trace(i, x_i, t)$ is calculated for the backtracking. Then, the optimal value is calculated with backtracking. Finally, we obtain the offloading policy X.

Given n tasks and possible decision epoch number t_a, the nested for-loops runs in $O(nt_a)$ time, which dominates the computing time of algorithm STA.

Algorithm 1 STA /∗ static algorithm ∗/

```
Input: $R_{up}$, $R_{down}$; $p_{up}$, $p_{down}$; $G$; $Y = \{y_1, y_2, \ldots, y_n\}$
Output: $X$ /∗ initialize the first task∗/
1:  for $t = 0$ to $t_a$ do
2:      $OPT(0,0,t) = 0, OPT(0,1,t) = \infty$.
3:  end for /∗ calculate other tasks ∗/
4:  for $i = 1$ to $n$ do
5:      for $t = 0$ to $t_a$ do
6:          Set $t^i_{x_{i-1},x_i}$, $Q^i_{x_{i-1},x_i}$ according to Eq. (8) and (9).
7:          Set $OPT(i, x_i, t)$ according to Eq. (10).
8:          Set $trace(i, x_i, t) = \arg\min_{x_{i-1}}\{OPT(i-1, x_{i-1},$
            $t - t^i_{x_{i-1},x_i}) + Q^i_{x_{i-1},x_i}\}$
9:      end for
10: end for /∗ calculate the virtual terminal task ∗/
11: Set $t^{n+1}_{x_n,0} := 0, Q^{n+1}_{x_n,0} := 0$.
12: Set $OPT(n+1,0,t_a)$ according to Eq. (10).
13: Set $trace(n+1, x_{n+1}, t_a) = \arg\min_{x_n}\{OPT(n, x_n, t_a)\}$
    /∗ backtracking along the trace ∗/
14: for i = n + 1 down to 2 do
15:     $X[i-1] := trace(i, x_i, t_a)$
16:     $t_a := t_a - t^i_{x_{i-1},x_i}$
17: end for
18: Return $X$.
```

3.2. Online algorithm

In this subsection, we aim to provide an online algorithm to adapt to the frequently changing system environment. Then, a two-stage online algorithm is designed to minimize energy consumption under the time constraint. In the offline stage, using Markov-decision process, we get the expected optimal cost from the current time to the end of the application. In the online stage, with the current network situation, the current computation reusing situation and the current time consumption, we can get a real-time task offloading policy which minimizes not only the energy consumption but also the delay constraint violation probability. In this paper, we ignore the related overheads which are generated by monitoring the system state information, as previous work [28] did.

3.2.1. Offline stage

In the offline stage, we build a Markov Decision Model for the above offloading problem. But, our offline stage algorithm is different from other Markov-Decision-based algorithms, such as [20–22]. First, with our new network architecture, we should take repetition probability of task results (i.e. dynamic changing caching results) into consideration. Then, since data uploading and downloading have huge different transmission rates and use different channels in the cellular network (5G), the channel state is subdivided as upload channel state and download channel state. Third, previous algorithms [20–22] always get a static offloading policy. Here, instead of static policy, we record the expected optimal decision tables for every state and use it for an online policy in the next stage (online stage).

We consider a finite horizon discrete time problem, where decisions are made at the beginning of a stage. Each stage is formed based on the number of tasks in the application. The decision epoch represents a point of time for the decision at the beginning of a stage. We represent the decision epoch as T=0,1,2, …,n, $n+1$, where decision epoch $t \in T$ indicates that task v_t has already been executed.

The decision maker (i.e. *offline algorithm*) chooses an action based on the system state information, denoted by S. Each state $s \in S$ is characterized by the combination of the channel states and the execution location of a task. The system state at decision epoch t is defined as $s_t = (t, x_i, \gamma_t^{\text{up}}, \gamma_t^{\text{down}})$, where γ_t^{up} and γ_t^{down} denote the upload and download channel states for the next epoch between the mobile and offloading sites (i.e. either g or b). Because the initial channel state is assumed to be good and the executions start and end at the mobile device, the initial and final system states are given as $s_0 = (0, 0, \gamma_0^{\text{up}} = g, \gamma_0^{\text{down}} = g)$ and $s_{n+1} = (n+1, 0, \gamma_{n+1}^{\text{up}}, \gamma_{n+1}^{\text{down}})$, respectively.

Given the current state, the decision maker can choose two major actions, offloading task to cloud side or continuing execution on mobile device. Based on the chosen action a_t in state s_t at decision epoch t, the state transition probability function for the next state s_{t+1} is given by $Pr[s_{t+1}|s_t, a_t]$. The system state transition probability between two decision epochs is given by

$$Pr(s_{t+1}|s_t, a_t) = \begin{cases} Pr_{\text{up}}Pr_{\text{down}} \text{ if } a_t \otimes x_t = x_{t+1}, \\ 0 \qquad\qquad \text{otherwise}, \end{cases} \tag{11}$$

where the exclusive or operation $a_t \otimes x_t$ is to determine if task v_t in x_t can transmit data to next task in x_{t+1} with action a_t

$$\begin{cases} C_E(s_t, a_t) = \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t) \\ \qquad \left[\sum_{y_t \in \{0,1\}} E_{v_t}^{\text{comp}}(x_t, y_t) p_t \right. \\ \qquad \left. + \sum_{y_{t+1} \in \{0,1\}} E_{v_t, v_{t+1}}^{\text{comm}}(x_t, x_{t+1}, y_{t+1}) p_{t+1} \right] \\ C_T(s_t, a_t) \quad = \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t) \\ \qquad \left[\sum_{y_t \in \{0,1\}} T_{v_t}^{\text{comp}}(x_t, y_t) p_t \right. \\ \qquad \left. + \sum_{y_{t+1} \in \{0,1\}} T_{v_t, v_{t+1}}^{\text{comm}}(x_t, x_{t+1}, y_{t+1}) p_{t+1} \right] \end{cases} \tag{12}$$

Dynamic caching computation result is considered in this subsection. According to [10], let $\mathbb{P} = \{p_0, p_1, \ldots, p_n, p_{n+1}\}$ be repetition probability of task's result (resource reusing possibility), where p_i is the probability that the result of task i can be reused. If we assume the transition of a state occurs from s_t to s_{t+1}, then the expected cost function C is calculated as in Eq. (12), where C_E denotes the expected energy cost function and C_T denotes the expected time cost function.

In the established MDP model, Bellman's equation expresses the optimality condition. Function $Q_t(s_t, a_t)$ is the expected optimal cost from decision epoch t with state s_t and action a_t to the terminal decision epoch. Function $V_t(s_t)$ is the expected optimal cost from decision epoch t with state s_t to the terminal decision epoch. The Bellman equation within our context takes the form as (13) and (14):

$$\begin{cases} V_t(s_t) = \min_{a_t \in a} Q(s_t, a_t) \\ V_{n+1}(s_{n+1}) = 0 \end{cases} \tag{13}$$

$$Q_t(s_t, a_t) = \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t)(C(s_t, a_t) + V_{t+1}(s_{t+1})) \tag{14}$$

Function $Q(s_t, a_t)$ and $V_t(s_t)$ are separated into two kinds. If time consumption is calculated, C, $Q(s_t, a_t)$ and $V_t(s_t)$ are denoted as C_T, $Q_t^T(s_t, a_t)$, $V_t^T(s_t)$. If energy consumption is calculated, C, $Q(s_t, a_t)$ and $V_t(s_t)$ are denoted as C_E, $Q_t^E(s_t, a_t)$, $V_t^E(s_t)$, respectively.

Using (13) and (14) iteratively, we can get the set of expected optimal cost tables (denote as $\mathbb{V}$) from any state to

Algorithm 2 OCT /∗Offline Stage to get expected optimal cost table∗/

```
Input: $<S, A, Pr, C, S_0>$
Output: Set $\mathbb{V}$
1:  Init $V_{n+1}(s_{n+1}) := 0$.
2:  for $k = n$ to 0 do
3:      for each state $s_k \in S$ do
4:          for $a_k \in \{0,1\}$ do
5:          Get $Q_k(s_k, a_k)$ according to Eq. (14).
6:          end for
7:          Get $V_k(s_k)$ according to Eq. (13).
8:      end for
9:  end for
10: Return $\mathbb{V}$.
```

the terminal decision epoch based on the dynamic programming method. There exist two kinds of expected optimal cost tables in the set $\mathbb{V}$, i.e. expected optimal energy cost table $V^E(s_t)$, expected optimal time cost table $V^T(s_t)$. The detail of the algorithm OCT is shown in Algorithm 2.

This algorithm uses traversal to get expected optimal cost from each state to the terminal decision epoch, which can be viewed as a forward process of dynamic programming. The time complexity of OCT is $O(nk^2)$ and the space complexity of OCT is $O(nk)$, where n is the number of the nodes and k is the number of the system states.

3.2.2. *Online stage*

In order to adaptively solve this problem with a dynamic changing system environment, an online algorithm GRP is developed using the set of expected optimal cost tables $\mathbb{V}$ which have been calculated in the offline stage, as shown in Algorithm 3.

Based on A∗ algorithm [42], at each iteration of online decision epoch, the optimal offloading policy is selected to minimize

$$E_t = E^{\mathrm{comm}}_{v_{t-1},v_t}(x_{t-1}, x_t, y_t) + V^E(s_t), \tag{15}$$

where v_t is the current task to get the offloading decision, $E^{\mathrm{comm}}_{v_{t-1},v_t}(x_{t-1}, x_t, y_t)$ is obtained using (5), and $V^E(s_t)$ is the expected optimal energy cost from current task to the terminal.

Since there exists delay constraint, at each iteration of online decision epoch, the optimal policy is selected to satisfy

$$T^{\mathrm{comm}}_{v_{t-1},v_t}(x_{t-1}, x_t, y_t) + V^T(s_t) + T_{\mathrm{current}} \le t_d, \tag{16}$$

where $T^{\mathrm{comm}}_{v_{t-1},v_t}(x_{t-1}, x_t, y_t)$ is obtained using (4), $V^T(s_t)$ is the expected optimal time consumption from current task to the terminal, T_{current} is the time consumption from beginning till current offloading epoch, and t_d is the delay constraint of the application.

Algorithm 3 GRP/∗online stage to get real-time policy∗/

```
Input: $\mathbb{V}$, $t_d$
Output: online policy $\pi$
1:  Measure current channel state $\gamma_i^{\mathrm{up}}, \gamma_i^{\mathrm{down}}$.
2:  Get the latest offloading decision $\pi(i-1)$ and
    application running time from initial to current $T_{\mathrm{current}}$.
3:  Update $y_i$ for the coming task $i$.
    /∗ Calculate energy cost and time consumption of data
    transmission from task $i-1$ to task $i$ with current
    channel state ∗/
4:  for $x_i \in \{0,1\}$ do
5:      calculate $T^{\mathrm{comm}}_{v_{i-1},v_i}(\pi(i-1), x_i, y_i)$ using Eq. (4).
6:      calculate $E^{\mathrm{comm}}_{v_{i-1},v_i}(\pi(i-1), x_i, y_i)$ using Eq. (5).
7:  end for
8:  if $E^{\mathrm{comm}}_{v_{i-1},v_i}(\pi(i-1), 0, y_i) + V^E(i, 0, \gamma_i^{\mathrm{up}}, \gamma_i^{\mathrm{down}}) >$
    $E^{\mathrm{comm}}_{v_{i-1},v_i}(\pi(i-1), 1, y_i) + V^E(i, 1, \gamma_i^{\mathrm{up}}, \gamma_i^{\mathrm{down}})$
9:  if $T^{\mathrm{comm}}_{v_{i-1},v_i}(\pi(i-1), 1, y_i) + V^T(i, 1, \gamma_i^{\mathrm{up}}, \gamma_i^{\mathrm{down}})$
    $+ T_{\mathrm{current}} < t_d$ then
10:         $\pi(i) := 1$
11: else
12:         $\pi(i) := 0$
13:     end if
14: end if
```

Using this online algorithm GRP, the optimal offloading policy can be obtained for the current task under current system environment.

4. NUMERIAL ANALYSIS

In this section, we evaluate the performance of our proposed algorithms in various situations.

4.1. Parameters for devices and applications

We evaluate the performance of the proposed algorithm on Matlab. Detail notations are shown in Table 1 and adapted in our simulation as in [39].

In this paper, we concentrate on sequential task topology. Computation workload and data transmission between tasks are set based on [20, 21]. To better reflect real-world scenarios, the size of the data is obtained based on the measurement of the real-time weather information API [43], indoor/outdoor map information API [44] and transportation information API [45]. We randomly generate applications' task graph based on the following parameters:

TABLE 1. Parameters of machine profile.

Data transmission power of the MD[a]	$p_{\text{up}} = 0.1\,\text{W}$
Data receiving power of the MD	$p_{\text{down}} = 0.05\,\text{W}$
Computation power of the MD	$p_m = 0.5\,\text{W}$
Idle power of the MD	$p_{\text{idle}} = 0.001\,\text{W}$
CPU frequency of the MD	$f_m = 500\,\text{MHz}$
CPU frequency of the cloud clone	$f_c = 3000\,\text{MHz}$

[a]MD is short for mobile device in this table.

- Node weight: computation workload $\omega = [20, 60]$ Mcycles.
- Edge weight: data transmission between tasks $\alpha = [10, 100]$ KB.
- The size of the data transmitted between the cloud-side database and task: $d = [4, 40]$ KB.

4.2. Performance of rapid static algorithm

The rapid static algorithm is proposed with rarely changing system environment. In this subsection, assuming the environment is static, we evaluate the running speed of the rapid static algorithm and explore whether the algorithm is suitable to the proposed Access Control management architecture.

4.2.1. Running speed with different time constraint and task scale

First, the rapid static algorithm is a kind of dynamic programming algorithm that always gets the exact optimal solution. So, the offloading performance of this algorithm is known as the optimal and it is not tested in this simulation. Then, we simulate and compare the computation speed between enumeration method and algorithm STA, since the offloading problem with a time constraint is not convex and no exact solution is delivered without enumeration method.

Assuming that the total time consumption of all tasks running on the mobile device is t, we compare the enumeration method and the algorithm STA with time constraint t, $0.9t$ and $0.8t$ in terms of the computation time. As shown in Fig. 3(a), time constraint does not have a significant impact on the running speed. In general, algorithm STA runs more than four times faster than the enumeration method. However, the running time of the two algorithms changes with the scale of task graph, as shown in Fig. 3(b). As a result, our proposed algorithm STA achieves much better improvement when the number of tasks increases.

4.2.2. Offloading performance with different communication rates and different PTR

Since the environment is assumed as static, y_i is taken as a priori in the modeling process. In the simulation, we randomly generate y_i during each repetitive experiment. In order to evaluate the impact of resource reusing to task offloading, we randomly chose different percentages of tasks whose results have been stored on MEC server (PTR). For simulation, the PTR is set to 0, 10%, 20%, 30%, respectively, and the communication rate ranges from $20\,\text{kb/s}$ to $100\,\text{kb/s}$. When giving PTR as r and the total number of tasks as n, we randomly select rn tasks whose task results have already been stored on MEC server. As shown in Fig. 4, we observe that the objective function value decreases with the increasing communication rate and the energy consumption decreases when PTR increase.

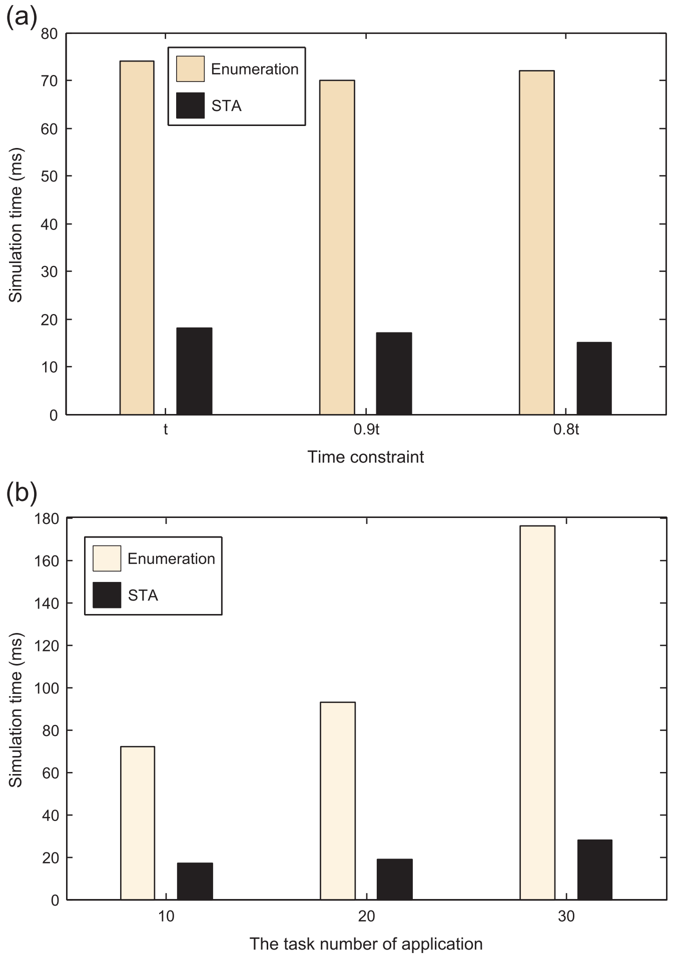

FIGURE 3. Comparisons of algorithm simulation time with different time constraints and different task scales.

4.3. Performance of two-stage online algorithm

In order to evaluate the performance of the proposed algorithms in frequently changing environment, we set four fading channels with different state transfer probability and different resource reusing possibilities. Parameters of the network environment are shown in Table 2. In this paper, the state transition rates in Table 2 are set from 0.7 to 1, while Pr_{gg} is

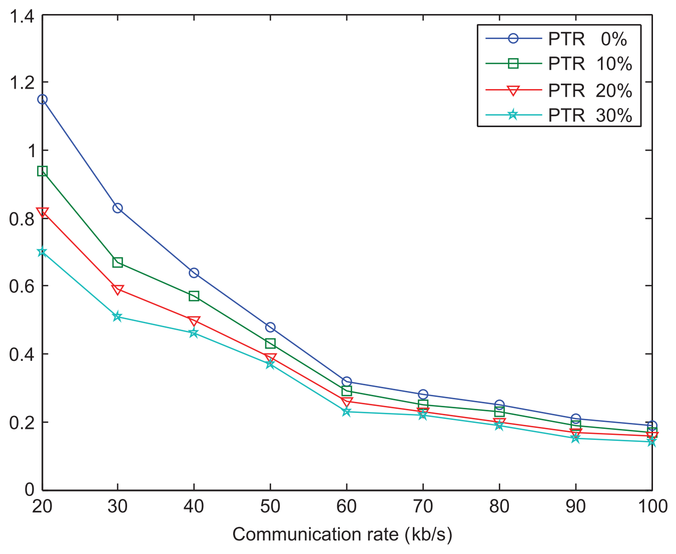

FIGURE 4. Impact of PTR and communication rate for static algorithm STA.

TABLE 2. Parameters of network environment.

Fading channel 1	$C_1 = \{Pr_{gg} = 1\}$
Fading channel 2	$C_2 = \{Pr_{gg} = 0.9, Pr_{bb} = 0.9\}$
Fading channel 3	$C_3 = \{Pr_{gg} = 0.8, Pr_{bb} = 0.8\}$
Fading channel 4	$C_4 = \{Pr_{gg} = 0.7, Pr_{bb} = 0.7\}$

set to 0.995 and Pr_{bb} is set to 0.96 in previous works [19–22, 34]. The state transition rate set as in Table 2 is to verify the performance of the proposed algorithms in dynamic changing environment. Data uploading and downloading with good channel state are set according to [46] and data uploading and downloading with bad channel state are set according to average bad data transmission rate of our real measurement using Speedtest [47]. The details are as follows:

- Data uploading rate with good channel state: $r_g^{\text{up}} = 256\,\text{KB/s}$.
- Data downloading with good channel state: $r_g^{\text{down}} = 1800\,\text{KB/s}$.
- Data uploading with bad channel state: r_b^{up}=50 KB/s.
- Data downloading with bad channel state: $r_b^{\text{down}} = 200\,\text{KB/s}$.

4.3.1. *Offloading performance with different repetition possibilities of task's results*

In order to measure the performance of our proposed architecture while eliminating the influence of the dynamic network, we compare the average energy consumption with the static network environment (fading channel C_1) and different resource reusing possibilities. Without loss of generality, all tasks' results are set with the same repetition possibilities

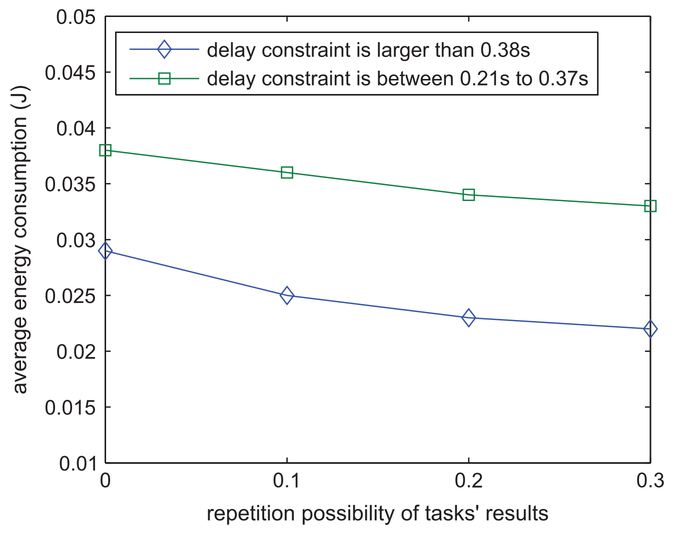

FIGURE 5. Energy consumption performance of GRP with different resource reusing possibilities. $\omega = \{31,42,59,26,21\}$, $\alpha = \{69,13,86,94,71,78\}$, $d = \{0,28,0,33,0\}$.

which are varying from 0 to 0.3, and y_i is randomly generated for each time slot under the constraint of repetition possibility.

Due to the application's inscape, two kinds of delay constraints are used, as shown in Fig. 5. When delay constraint is larger than 0.38 s, the offloading policy remains {1, 1, 1, 1, 1}; when delay constraint is between 0.21 s and 0.37 s, the offloading policy remains {0, 1, 1, 1, 1}. We can see that, with resource reusing possibility increasing, the energy consumption is smaller for any kind of delay constraint. This demonstrates the efficiency of our proposed Access Control management architecture. When delay constraint is larger than 0.38 s, the energy consumption changes more slowly compared with delay constraint between 0.21 s and 0.37 s.

4.3.2. *Offloading performance with different fading channels*

In order to measure the performance of the proposed online algorithm GRP, we compare the average energy consumption with different kinds of fading channel (C_1, C_2, C_3, C_4) and a fixed repetition possibility of tasks' result (every repetition possibility of tasks' results is set as 0.1). The baseline policies are the offline policy produced by exhaustive enumeration with static system environment and the online policy produced by the online algorithm ARHOS [28].

As shown in Fig. 6, we can see that, with channel state more frequently changed, the average energy consumption of both the baseline policies and the proposed online policy becomes larger. This indicates that the frequency change of channel states has negative influence to the energy consumption. GRP's policy is more efficient than the baseline policies. When the fading channel is static, GRP performances better

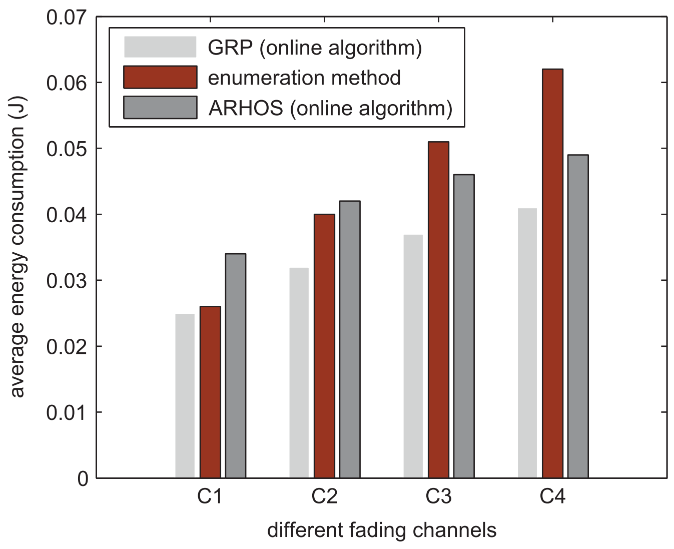

FIGURE 6. Energy consumption with different fading channels.

than offline method (exhaustive enumeration), since GRP can get the optimal solution for current state using OCT and more flexible with dynamic changing repetition possibility of tasks' results. When fading channel changes more frequently (C_3 and C_4), online algorithms performance much better than offline method (exhaustive enumeration), and GRP performances better than online algorithm ARHOS. For fading channel C_4, nearly 34% of energy consumption is saved compared with the offline method (exhaustive enumeration) and 16.3% of energy consumption is saved compared with online algorithm ARHOS.

4.3.3. *Time violation rate with different delay constraints*

Time violation means that the real-time consumption of the policy violates the given delay constraint, and it can affects users' experience significantly. In the test, resource reusing possibility (repetition possibility of tasks' result) is fixed as 0.1, network environment is fixed as C_2 and delay constraint is set between the time consumptions that all tasks are running on cloud side and on the mobile device. Then, we evaluate time violation rates with different delay constraints.

In this subsection, we use MDM to indicate baseline algorithm [21]. When time violation rate is no more than 0.25 with Gilbert–Elliott model for MDM, no policy can satisfy the delay constraint. But, since our Access Control management architecture can save more time and energy consumption, it is not fair to directly compare with [21]. So we adapt MDM with our new architecture that can reuse computational resource, and named it as MDM-NEW. As shown in Fig. 7, when delay constraint is 0.4, 52% of time violation rate is reduced by MDM-NEW compared with MDM.

Our online algorithm GRP has smaller time violation rates than MDM-NEW. When delay constraint is 0.4, 25% of time violation rate is reduced compared with MDM-NEW. That is because our online algorithm GRP can dynamically choose a suitable policy to satisfy the delay constraint.

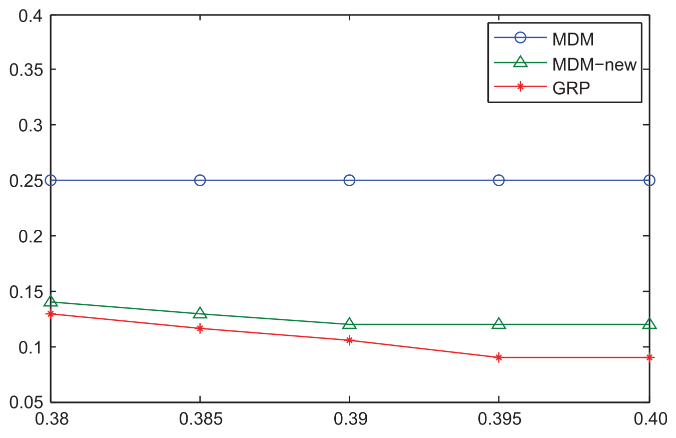

FIGURE 7. Time violation rate with different delay constraints. $\omega = \{31,42,59,26,21\}$, $\alpha = \{69,13,86,94,71,78\}$, $d = \{0,28,0,33,0\}$.

5. CONCLUSION

In this paper, we have proposed an architecture of access controller management for 5 G heterogeneous network, by making full use of Base Station's storage capability and reusing repetitive task computation. The architecture can reduce both energy and time consumption for mobile applications. Then, a collaborative task graph with real-time computational information was proposed which supports energy efficient offloading policy while satisfies the time constraint. Two algorithms have been proposed for determining offloading under two different scenarios, i.e. rarely changing system environment and frequently changing system environment, respectively. For rarely changing system environment, an optimal static algorithm runs more than four times faster than enumeration method. For frequently changing system environment, proposed algorithm can save 16.3% energy compared with algorithm ARHOS and reduce the time violation rate by 25% compared with a modified offline algorithm. In the future, we will investigate the effect of the management of computation results reusing.

FUNDING

This work was supported by the National Key R&D Program of China under Grant no. 2018YFB1003201, and Guangdong Key R&D Program of China under Grant no. 2018B010107003. It was also supported in part by the National Natural Science Foundation of China under Grant nos. 61702115 and 61672171, by the Major Research Project of Guangdong Province under Grant nos. 2016KZDXM052 and 2018B030311007.

REFERENCES

[1] Dinh, H.T., Lee, C., Niyato, D. and Wang, P. (2013) A survey of mobile cloud computing: architecture, applications, and approaches. *Wirel. Commun. Mobile Comput.*, **13**, 1587–1611.

[2] Cui, Y., Ma, X., Wang, H., Stojmenovic, I. and Liu, J. (2013) A survey of energy efficient wireless transmission and modeling in mobile cloud computing. *Mobile Netw. Appl.*, **18**, 148–155.

[3] Computing, E.M.E., Initiative, I. *et al* (2014). Mobile-edge computing: introductory technical white paper.

[4] Kumar, K., Liu, J., Lu, Y.-H. and Bhargava, B. (2013) A survey of computation offloading for mobile systems. *Mobile Netw. Appl.*, **18**, 129–140.

[5] Hu, Y.C., Patel, M., Sabella, D., Sprecher, N. and Young, V. (2015) Mobile edge computing a key technology towards 5g. *ETSI white paper*, **11**, 1–16.

[6] Zhang, K., Mao, Y., Leng, S., Zhao, Q., Li, L., Peng, X., Pan, L., Maharjan, S. and Zhang, Y. (2016) Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE Access*, **4**, 5896–5907.

[7] Guo, S., Xiao, B., Yang, Y. and Yang, Y. (2016) Energy-Efficient Dynamic Offloading and Resource Scheduling in Mobile Cloud Computing. *INFOCOM 2016—35th Annu. IEEE Int. Conf. Computer Communications, IEEE*, pp. 1–9. IEEE.

[8] Chen, L., Wu, J., Dai, H.N. and Huang, X. (2018) Brains: joint bandwidth-relay allocation in multi-homing cooperative d2d networks. *IEEE Trans. Vehicular Technol.*, **PP**, 1–12. doi:10.1109/TVT.2018.2799970.

[9] Rimal, B.P., Van, D.P. and Maier, M. (2017) Mobile edge computing empowered fiber-wireless access networks in the 5g era. *IEEE Commun. Mag.*, **55**, 192–200.

[10] Tamoor-ul Hassan, S., Bennis, M., Nardelli, P.H. and Latva-Aho, M. (2016) Caching in wireless small cell networks: a storage-bandwidth tradeoff. *IEEE Commun. Lett.*, **20**, 1175–1178.

[11] Jiang, W., Feng, G. and Qin, S. (2017) Optimal cooperative content caching and delivery policy for heterogeneous cellular networks. *IEEE Trans. Mobile Comput.*, **16**, 1382–1393.

[12] Chen, L., Wu, J., Zhang, X.X. and Zhou, G. (2017) Tarco: two-stage auction for d2d relay aided computation resource allocation in hetnet. *IEEE Trans. Serv. Comput.*, **PP**, 1–14. doi:10.1109/TSC.2018.2792024.

[13] Wang, T., Wei, X., Tang, C. and Fan, J. (2018) Efficient multi-tasks scheduling algorithm in mobile cloud computing with time constraints. *Peer-to-Peer Network. Appl.*, **11**, 793–807.

[14] Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S. and Sabella, D. (2017) On multi-access edge computing: a survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Commun. Surv. Tutorials*, **19**, 1657–1681.

[15] Barbarossa, S., Sardellitti, S. and Di Lorenzo, P. (2014) Communicating while computing: distributed mobile cloud computing over 5g heterogeneous networks. *IEEE Signal Processing Magazine*, **31**, 45–55.

[16] Lyu, X., Tian, H., Ni, W., Zhang, Y., Zhang, P. and Liu, R.P. (2018) Energy-efficient admission of delay-sensitive tasks for mobile edge computing. *IEEE Trans. Commun.*, **PP**, 1–14. doi:10.1109/TCOMM.2018.2799937.

[17] Cuervo, E., Balasubramanian, A., Cho, D.-K., Wolman, A., Saroiu, S., Chandra, R. and Bahl, P. (2010) Maui: Making Smartphones Last Longer with Code Offload. *Proc. 8th Int. Conf. Mobile Systems, Applications, and Services*, pp. 49–62. ACM.

[18] Wu, H. and Wolter, K. (2015) Software Aging in Mobile Devices: Partial Computation Offloading as a Solution. *2015 IEEE Int. Sympos. Software Reliability Engineering Workshops (ISSREW)*, pp. 125–131. IEEE.

[19] Shahzad, H. and Szymanski, T.H. (2016) A Dynamic Programming Offloading Algorithm Using Biased Randomization. *2016 IEEE 9th Int. Conf. Cloud Computing (CLOUD)*, pp. 960–965. IEEE.

[20] Zhang, W., Wen, Y. and Wu, D.O. (2013) Energy-Efficient Scheduling Policy for Collaborative Execution in Mobile Cloud Computing. *INFOCOM, 2013 Proc. IEEE*, pp. 190–194. IEEE.

[21] Zhang, W., Wen, Y. and Wu, D.O. (2015) Collaborative task execution in mobile cloud computing under a stochastic wireless channel. *IEEE Trans. Wirel. Commun.*, **14**, 81–93.

[22] Liu, J., Mao, Y., Zhang, J. and Letaief, K.B. (2016) Delay-Optimal Computation Task Scheduling for Mobile-edge Computing Systems. *IEEE Int. Sympos. Information Theory*, pp. 1451–1455.

[23] Kwak, J., Kim, Y., Lee, J. and Chong, S. (2015) Dream: dynamic resource and task allocation for energy minimization in mobile cloud systems. *IEEE J. Selected Areas Commun.*, **33**, 2510–2523.

[24] Wang, J., Peng, J., Wei, Y., Liu, D. and Fu, J. (2017) Adaptive application offloading decision and transmission scheduling for mobile cloud computing. *China Commun.*, **14**, 169–181.

[25] Mao, Y., Zhang, J. and Letaief, K.B. (2016) Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE J. Selected Areas Commun.*, **34**, 3590–3605.

[26] Mattingley, J., Wang, Y. and Boyd, S. (2011) Receding horizon control. *IEEE Control Syst.*, **31**, 52–65.

[27] Guo, H. and Liu, J. (2018) Collaborative computation offloading for multi-access edge computing over fiber-wireless networks. *IEEE Trans. Vehicular Technol.*, **67**, 4514–4526.

[28] Lyu, X. and Tian, H. (2016) Adaptive receding horizon offloading strategy under dynamic environment. *IEEE Commun. Lett.*, **20**, 878–881.

[29] Song, J., Cui, Y., Li, M., Qiu, J. and Buyya, R. (2014) Energy-Traffic Tradeoff Cooperative Offloading For Mobile Cloud Computing. *2014 IEEE 22nd Int. Sympos. Quality of Service (IWQoS)*, pp. 284–289. IEEE.

[30] Anand, A., Gupta, A., Akella, A., Seshan, S. and Shenker, S. (2008) Packet caches on routers: the implications of universal redundant traffic elimination. *ACM SIGCOMM Comput. Commun. Rev.*, **38**, 219–230.

[31] Spring, N.T. and Wetherall, D. (2000) A protocol-independent technique for eliminating redundant network traffic. *ACM SIGCOMM Comput. Commun. Rev.*, **30**, 87–95.

[32] Feng, J., Li, H., Huang, M., Liu, S., Ou, W., Wang, Z. and Zhu, X. (2018) Learning to Collaborate: Multi-scenario Ranking via

Multi-agent Reinforcement Learning. *Proc. 2018 World Wide Web Conf. World Wide Web*, pp. 1939–1948. International World Wide Web Conferences Steering Committee.

[33] Gkatzikis, L. and Koutsopoulos, I. (2013) Migrate or not? exploiting dynamic task migration in mobile cloud computing systems. *IEEE Wirel. Commun.*, **20**, 24–32.

[34] Terefe, M.B., Lee, H., Heo, N., Fox, G.C. and Oh, S. (2016) Energy-efficient multisite offloading policy using Markov decision process for mobile cloud computing. *Pervasive Mobile Comput.*, **27**, 75–89.

[35] Johnston, L.A. and Krishnamurthy, V. (2006) Opportunistic file transfer over a fading channel: a pomdp search theory formulation with optimal threshold policies. *IEEE Trans. Wirel. Commun.*, **5**, 394–405.

[36] Wu, Y. and Krishnamachari, B. (2012) Online Learning to Optimize Transmission over an Unknown Gilbert–Elliott Channel. *2012 10th Int. Sympos. Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt)*, pp. 27–32. IEEE.

[37] Cai, Z., Li, X. and Gupta, J.N. (2016) Heuristics for provisioning services to workflows in xaas clouds. *IEEE Trans. Serv. Comput.*, **9**, 250–263.

[38] Zhang, Z., Wu, J., Jiang, G., Chen, L. and Lam, S.K. (2017) Qoe-Qware Task Offloading for Time Constraint Mobile Applications. *IEEE Conf. Local Computer Networks*, pp. 510–513.

[39] Zhang, W. and Wen, Y. (2015) Energy-efficient task execution for application as a general topology in mobile cloud computing. *IEEE Trans. Cloud Comput.*, **PP**, 1–12. doi:10.1109/TCC.2015.2511727.

[40] Kellerer, H., Pferschy, U. and Pisinger, D. (2004) *Knapsack Problems*. Springer, Berlin Heidelberg.

[41] Smith, D.K. (1994) Network flows: theory, algorithms, and applications. *J. Oper. Res. Soc.*, **45**, 1340.

[42] Zeng, W. and Church, R.L. (2009) Finding shortest paths on real road networks: the case for a. *Int. J. Geogr. Inf. Sci.*, **23**, 531–543.

[43] weatherdt (2017). weather_api. http://www.weatherdt.com/market/datastore/api_details/96.

[44] baidu (2017). Baidumap_api. http://lbsyun.baidu.com/apiconsole/key.

[45] data, J. (2017). traffic_infomation_api. http://v.juhe.cn/trafficInfo/getTrafficInfoByName.

[46] wikipedia (2017). 4g_data_transmission_wiki. https://en.wikipedia.org/wiki/4G.

[47] by Ookla, S. (2017). Speedtest to test speed. http://www.speedtest.net/.