Edge-Aided Computing and Transmission Scheduling for LTE-U-Enabled IoT

Hongli He, *Member, IEEE*, Hangguan Shan, *Member, IEEE*, Aiping Huang, *Senior Member, IEEE*, Qiang Ye, *Member, IEEE*, and Weihua Zhuang, *Fellow, IEEE*

***Abstract*— To facilitate the deployment of private industrial Internet-of-Things (IoT), applying long-term-evolution (LTE) over unlicensed spectrum (LTE-U) is a promising technology, which can deal with the licensed spectrum scarcity problem and the stringent quality-of-service (QoS) requirement via centralized control. In this paper, we investigate the computing offloading problem for LTE-U-enabled IoT, where computing tasks on an IoT device are either executed locally or offloaded to the edge server on an LTE-U base station. Considering a constrained edge computing cost (e.g., operation power consumption) for offloaded tasks, the task scheduling problem is formulated as a constrained Markov decision process (CMDP) to maximize the long-term average reward, which integrates both task completion profit and task completion delay. In order to address the uncertainty of task arrivals and channel availability, a constrained deep Q-learning-based task scheduling algorithm with provable convergence is proposed, where an adaptive reward function can appropriately bound the average edge computing cost. Extensive simulation results show that the proposed scheme considerably enhances the system performance.**

***Index Terms*— Mobile edge computing, task offloading, Internet-of-Things, LTE over unlicensed, constrained deep reinforcement learning.**

Manuscript received July 31, 2019; revised April 7, 2020 and August 2, 2020; accepted August 3, 2020. Date of publication August 25, 2020; date of current version December 10, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFE0126300, in part by the National Natural Science Foundation Program of China under Grant 61771427 and Grant U1709214, in part by the Huawei Technologies Company Ltd. under Grant YBN2018115223, in part by the Ng Teng Fong Charitable Foundation under Grant ZJU-SUTD IDEA, in part by the 5G Open Laboratory of Hangzhou Future Sci-Tech City, and in part by the Faculty Research Grant (FRG) from Minnesota State University, Mankato. This article was presented in part at the IEEE GLOBECOM 2018 [1]. The associate editor coordinating the review of this article and approving it for publication was R. Tandon. *(Corresponding author: Hangguan Shan.)*

Hongli He and Aiping Huang are with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China, and also with the Zhejiang Provincial Key Laboratory of Information Processing and Communication Networks, Zhejiang University, Hangzhou 310027, China (e-mail: hongli_he@zju.edu.cn; aiping.huang@zju.edu.cn).

Hangguan Shan is with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China, also with the Zhejiang Provincial Key Laboratory of Information Processing and Communication Networks, Zhejiang University, Hangzhou 310027, China, and also with the Zhejiang Laboratory, Hangzhou 310000, China (e-mail: hshan@zju.edu.cn).

Qiang Ye is with the Department of Electrical and Computer Engineering and Technology, Minnesota State University, Mankato, MN 56001 USA (e-mail: qiang.ye@mnsu.edu).

Weihua Zhuang is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: wzhuang@uwaterloo.ca).

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

Digital Object Identifier 10.1109/TWC.2020.3017207

I. Introduction

RECENTLY, there has been a soaring proliferation of smart devices with ubiquitous interconnections for providing Internet-of-Things (IoT) services [2]. By enabling the osmotic convergence of transmission, computing, and storage on IoT devices, a diversity of promising applications have emerged, encompassing smart homing, intelligent connected vehicles, and wearable IoT systems [3]. While many IoT applications can be supported via cellular networks, there are certain entities that prefer their own private networks for reasons such as data security and/or network controllability [4]. Without having licensed radio spectrum resources, these entities generally resort to operate their private networks using unlicensed spectrum. One solution is to use non-3GPP air interface technologies, such as low-power wide-area (LPWA)-based systems over the unlicensed spectrum [5]. Most existing LPWA-based systems such as long range (LoRa) over the unlicensed spectrum are usually simplified for cost-saving purposes, at the price of service performance [6]. On the other hand, long-term evolution (LTE) over unlicensed spectrum (LTE-U) provides an alternative solution that can share the cellular IoT ecosystem with unlicensed spectrum resources and satisfy the enterprise customers' requirements for a dedicated IoT network. Technically, many of the advanced wireless techniques of LTE-U inherited from cellular systems are proven to offer better performance for IoT connections than traditional IoT technologies over the unlicensed spectrum, and overcome performance limitations of existing LPWA systems [7]. For better promoting the performance of LTE-U network in IoT services, a lot of organizations, especially Multefire, have proposed some dedicated modifications, for example, supporting eMTC (with bandwidth of 1.4 MHz) in unlicensed bands, supporting narrow band (NB)-IoT (with bandwidth of 200kHz) in unlicensed bands, and being compatible with lower spectrum bands (sub-1GHz). Currently, LTE-U-based technology has been deployed in some practical IoT systems such as Shanghai Yangshan Phase VI Port, where the LTE-U network can support the automated guided vehicle controlling, wireless close-circuit Television (CCTV), and remote crane monitoring system in an area with length of 2350 meters and seven berths [8].

Meanwhile, the constrained computational capability and power capacity of IoT devices pose a significant challenge for the quality-of-service (QoS) guarantee of the emerging applications, especially for those with intensive computation, e.g., image processing in multimedia IoT, although local computing at the device can significantly reduce the latency for transmission and the privacy leakage [9]. One simple way to address this issue is to establish a sustainable tunnel between the resource-limited devices and the cloud computing system. However, cloud servers are usually located far away from the end-devices, leading to high transmission delay over backhaul link and excessive stress of traffic load on the core networks [10]. To tackle these challenges, a mobile edge computing (MEC) paradigm emerges via shifting some tasks from the devices to vicinal computation capable entities, i.e., the edge of wireless networks. The advent of MEC facilitates the extension of cloud computing services to the edge within the radio access network of the end-devices, and provides a swift computing and mobility-aware support [11].

However, employing MEC for LTE-U-enabled IoT faces significant challenges to satisfy the high reliability and low latency requirements. Specifically, the task forwarding from an IoT device to the edge server can be interrupted by other transmissions from the legacy networks (e.g. a Wi-Fi network) operating over the unlicensed bands, leading to a performance degradation. Further, the carrier-sensing-based channel access in LTE-U networks causes extra waiting time of detecting channel availability in the reverse transmission from the edge server to the IoT device, while the delay for the reverse transmission over licensed spectrum is not considered in existing studies due to small data size of task execution results. Therefore, dynamics of the unlicensed channel availability lead to long delay and unreliability in data transmission of the MEC system, and make the delay analysis intractable.

In this paper, we study an edge-aided task scheduling problem in an LTE-U-enabled network, where an LTE-U base station (BS) helps IoT devices to decide whether tasks should be locally computed or be offloaded to the edge server on the LTE-U BS. Specifically, by taking account of the dynamic queueing, transmission, and computing of the associated tasks, we propose a state-wise model for the evolution of the MEC system and the availability of wireless channels. Correspondingly, the profit of task completion and the joint computing and transmission delay are integrated for evaluating the effect of different task scheduling policies, under a constraint of the edge computing cost (e.g., energy consumption for offloaded tasks). The main contributions of this paper are threefolded:

- The applicability of the constrained Markov decision process (CMDP) framework is explored for the task scheduling in an LTE-U-enabled IoT network for attaining a high reward related to the profit and delay of task completion, with a constraint of the average edge computing cost. The random generation, transmission, and computation of the tasks, the status of the edge server, and the dynamics of the unlicensed channel availability are captured in the proposed stochastic optimization-based task scheduling framework;
- With no priori knowledge about channel status and task arrivals, we propose a constrained deep Q-learning (CDQL) solution for the task scheduling problem. Motivated by applying the Lagrange duality in solving CMDP problem with available environment information, we propose, for the first time, a subgradient-based reinforcement learning algorithm to solve the cost-constrained task scheduling, which is transfered to a standard reinforcement learning problem without cost constraint when the optimal Lagrange dual multiplier is given. Specifically, the value of the Lagrange dual multiplier is proposed to be updated based on the counted average edge computing cost when the environment information is unavailable, and the convergence of the subgradient-based learning algorithm is proved;
- For implementing the proposed CDQL algorithm, we modify the traditional deep Q-learning (DQL) architecture including the deep neural network (DNN) structure for estimating the Q-values, the preprocessing of the training samples, and the pretraining with adaptive learning rates to accelerate the learning for Q values.

The remainder of the paper is organized as follows: We discuss the related work in Section II. System model is described in Section III. In Section IV, we formulate the edge-aided task scheduling problem as a CMDP framework and present the CDQL algorithm in Section V. The customized modification for the realization of our proposed algorithm is presented in Section VI and simulation results are given in Section VII, followed by conclusions in Section VIII.

II. Related Work

There have been a collection of research works studying how to guarantee the performance of IoT over the unlicensed spectrum. For achieving extended coverage, low power consumption, and massive connectivity over unlicensed spectrum, NB-IoT-U, a cellular industrial IoT system over unlicensed spectrum, is proposed by expanding NB-IoT technology in sub-1-GHz [7]. In [12], a distributed channel sensing and resource allocation scheme is proposed to achieve a high resource reuse ratio for massive IoT connections via appropriately mitigating intra/inter-network interference over the unlicensed spectrum. In order to extend the lifetime of IoT devices, energy harvesting is enabled and analyzed for an IoT network over unlicensed spectrum in [13]. In [14], a cellular-user-aided relay scheme is proposed to bridge the communication between an IoT device and its destined BS via machine-to-machine communication over the unlicensed spectrum, which reduces the energy consumption and supports more connected devices. These works have addressed many communication-related issues for the LTE-U-enabled IoT, which builds a robust radio access network for supporting services in terms of coverage, connectivity, interference coordination, and energy consumption. However, accommodating computing-intensive services in LTE-U-based networks still faces significant challenges due to the unreliable communication links over the unlicensed spectrum and the complex orchestration for both transmission and computing resources [15].

While the MEC in LTE-U-enabled IoT is under investigated, applying MEC for licensed IoT has been exploited extensively. The existing researches mainly use two kinds of methods to design the task scheduling strategy for edge computing. The first one is Lyapunov-based method. In [16], by integrating the queue length of edge servers, Chen *et al.* propose a dynamic computing offloading scheme to achieve the tradeoff between offloading cost and performance based on Lyapunov optimization. In [17], considering the uncertainty and dynamics of wireless channel state and task arrival process, and the large scale of solution space, an energy-efficient task offloading scheme is proposed by leveraging the Lyapunov drift. To relieve the stringent requirement of tasks' feedback, a scheduling algorithm tolerant to out-of-date network knowledge is proposed in [18] and can achieve asymptotical optimality with only partial information via applying both the perturbed Lyapunov function and the knapsack problem formulation. The queueing-theory-based and Lyapunov-drift-based model in these papers can maintain the theoretical stability of the MEC system when the task generation process is dynamic. However, such models cannot integrate some non-queue-based information, e.g., the channel access state of the LTE-U network when employing listen-before-talk (LBT)-based channel access schemes, and neglect the exploration for the network statistics in a large time scale (e.g., task generation). Therefore, such algorithms usually sacrifice their performance for the system roubutness. The second one is the reinforcement learning method. In [19], a reinforcement-learning-based scheduling scheme is proposed for the virtual machine assignment and task offloading in a space-air-ground integrated network. In [20], considering the stochastic vehicle traffic, dynamic computation requests, and time-varying communication conditions, a semi-Markov process is formulated and a deep reinforcement learning method is proposed to obtain the optimal policies of computation offloading and resource allocation. In [21], a multi-user multi-edge-node computation offloading problem is formulated as an unknown payoff game and solved by the distributed reinforcement learning method. In these works, the reinforcement learning methods are mostly based on the Q-value framework, and cannot be directly applied in the scenario with an edge computing cost constraint. In the following, we present a comprehensive CDQL-based task scheduling framework for an LTE-U-enabled IoT with MEC to facilitate the state-wise task offloading decision while guaranteeing the edge computing cost constrained within the given cost budget.

III. System Model

A. Network Scenario

We consider an MEC-aided LTE-U-enabled IoT as shown in Fig. 1, consisting of one LTE-U BS, and K LTE-U-enabled IoT devices. Let the indexes of the K devices be denoted by the elements in $\mathcal{K} = \{1, 2, ..., K\}$ and the index of the BS be 0. Transmission and computing functions are both integrated in the LTE-U BS operating over unlicensed spectrum. A channel with bandwidth B is assigned for the MEC service supported by an edge server on the LTE-U BS. The system evolves over time in a slotted structure indexed by t (> 0) and the length of each slot is τ. Because the IoT devices are normally

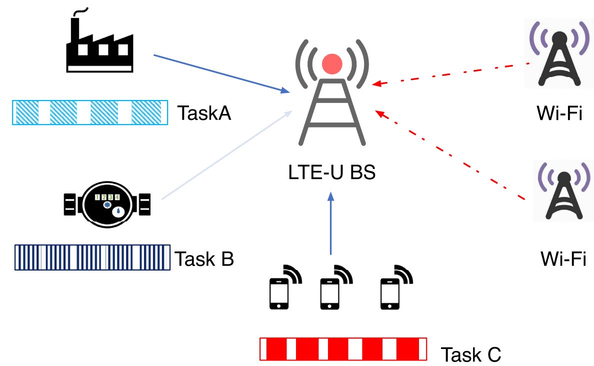


Fig. 1. The deployment scenario.

deployed for some dedicated services [22], we consider that each device has only one application and needs to keep performing one kind of corresponding task [23]. Due to the stringent computing capability of the devices and the possible low latency requirements, some tasks should be offloaded to the edge server on the LTE-U BS which is empowered with much richer computing resources.

The structure of tasks generated from the same device has a consistent format. The task structure at device k is denoted by $M_k \triangleq (L_k, X_k, f_k)$, where L_k denotes the size of the task input (in bit), X_k is the required computation intensity (in CPU cycles per bit), and f_k is the profit for task completion [24]. These statistics can be obtained using the method proposed in [25]. A linear dependancy is considered for two consecutive tasks of the same device, i.e., a task of device k $(\in \mathcal{K})$ can only be performed after the completion of its previous task. In a practical scenario, an urgent task with high priority can preempt the unserved low-priority tasks, which changes the original serving order. However, such special cases do not affect the system modeling under consideration in this work because the computing of the reordered tasks leads to the same reward (as specified in Section IV) when the scheduling policy is fixed. By setting the duration of each time slot as a small value, it is assumed that at most one task is generated at a device within each time slot. The task arrival probability at device k in slot t is denoted by $p_k(t)$. The computing capacity (in the unit of cycles per second) of device k is represented by C_k $(k \in \mathcal{K})$ while the computing capacity of the BS is C_0 $(C_0 \gg C_k, \forall k \in \mathcal{K})$.

Since an LTE-U network shares the unlicensed spectrum with other legacy networks, e.g., Wi-Fi, for fair coexisting, an LBT mechanism is applied for the LTE-U network [26]. By taking into account both flexibility and simplicity of the channel access, a category 4 scheme (LBT with random back-off and a contention window of variable size) defined by 3GPP is adopted in the LTE-U BS for the unlicensed channel access [27]. The listening (i.e., channel sensing) of the LTE-U network is executed by the BS and the channel-related state is represented by tuple $s_{\mathrm{C}}(t) = (a(t), b(t), m(t))$, where $a(t)$ is the phase indicator, $b(t)$ is the phase counter, and $m(t)$ is the backoff stage. When $a(t) = 0$ (or 1), the BS is under the sensing (or transmission) phase. Meanwhile, $b(t)$ is a backoff counter with different indications under different phases. When the BS intends to access the channel, the backoff stage $m(t)$ is set as 0, and the backoff counter $b(t)$ is randomly initialized between 0 and $W_0 - 1$. Then the BS senses the channel at the beginning of each slot. The backoff counter $b(t)$ is decreased by 1 at each time slot under idle channel, while it keeps frozen if the channel is busy. If there is no transmission collision with other networks over the unlicensed channel after the backoff counter $b(t)$ is reduced to 0, the BS begins to access the unlicensed channel for scheduling the devices to do their task-related transmission. Under the collision case, the backoff stage $m(t)$ is increased by 1, and the backoff window is doubled, i.e., $W_{m(t)} = 2W_{m(t)-1}$, where maximum backoff stage is M. Then the backoff counter $b(t)$ is randomly reset between 0 and $W_{m(t)} - 1$ for the subsequent channel sensing.

After successfully receiving the uplink grant signal, the associated IoT devices immediately upload their task data and equally share the available frequency bands. The uplink transmission rate of device k at slot t is given by

$$r_k(t) = \frac{B}{J(t)} \log\left(1 + \frac{P_k|h_k(t)|^2}{\sigma^2}\right) \tag{1}$$

where $J(t)$ is number of devices with transmission demand in slot t, P_k is the fixed transmission power of device k, $h_k(t)$ is the channel gain from device k to the BS, including the path loss and Rayleigh fading, and σ^2 is the received noise power. At the end of the transmission phase, $a(t)$ and $b(t)$ are reset to 0 and W_0, respectively, for the next round of channel sensing. The Wi-Fi system works as one external environmental element. Specifically, as the number of Wi-Fi users increases, the average channel quality of the LTE-U decreases, which leads to a longer transmission delay. Hence, the tasks are more likely to be processed locally. That is, the activity of Wi-Fi devices affects the evolution of channel availability, hence the transmission delay and finally the offloading decision.

B. Offloading Policy and Device Status

When a task becomes the head of the task queue in device k at the beginning of time slot t, it is scheduled by the BS for locally processing or offloading. The locally processing at device k is denoted by action $A_k(t) = 0$ while offloading decision is denoted by $A_k(t) = 1$. The decision epochs for task processing are discussed in Section III-C. The device status is highly dependent on its task that is being processed, referred to as in-service task. The status of all devices is captured by vector $\boldsymbol{z}(t) = (z_1(t), z_2(t), \ldots, z_k(t), \ldots, z_K(t))$, $z_k(t) \in \{0, 1, 2, 3, 4\}, k \in \mathcal{K}$. Specifically, for in-service task of device k in slot t, $z_k(t) = 0$ means that the task is being locally processed (including the case that the task queue is empty); $z_k(t) = 1$ means that the device is uploading the input data of the task to the BS; $z_k(t) = 2$ denotes that the in-service task of device k is queueing at the edge server; $z_k(t) = 3$ denotes that the task is being processed at the edge server; $z_k(t) = 4$ indicates that the task is accomplished by the edge server and is waiting for returning task execution result from the edge server. As shown in Fig. 2, the evolution

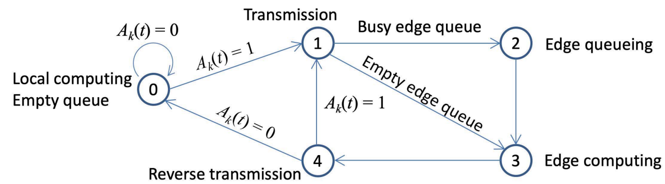


Fig. 2. The task processing flow.

for the device status depends on the availability of the unlicensed channel and computing resources, as discussed in the following subsections.

C. Task Backlog

Each device is equipped with a task buffer of capacity $\bar{B}_k$, containing all its task backlog. The number of unaccomplished tasks in the buffer of device k at the beginning of time slot t is denoted by $B_k(t)$. The generation of new tasks and the processing of in-service task jointly decide backlog update, expressed as

$$B_k(t+1) = \min\left\{B'_k(t) + Y_k(t), \bar{B}_k\right\} \tag{2}$$

where $Y_k(t)$ is a Bernoulli-distributed random variable with parameter $p_k(t)$, indicating whether in slot t a new task is generated at device k, and $B'_k(t)$ denotes the updated backlog size without new task generation, derived by

$$B'_k(t) = \begin{cases} \max\left\{B_k(t) - \frac{C_k\tau}{L_k X_k}, \lfloor B_k(t) \rfloor^-\right\}, & \text{if } z_k(t) = 0 \\ B_k(t) - 1, & \text{if } z_k(t) = 4 \text{ and } a(t) = 1 \\ B_k(t), & \text{otherwise} \end{cases} \tag{3}$$

where $\lfloor \cdot \rfloor^-$ is the revised floor function mapping to the greatest integer less than itself. The first subequation in (3) represents the case that a task is being locally computed at device k, and its backlog size is reduced by the normalized fraction. The second subequation indicates that an in-service task of device k has been completed at the edge server, and the backlog is updated when the task execution result is returned from the edge server once the channel is available.

The decision epochs for device k can be classified into two categories. One is referred to as time slots with a local task completion or a return of the task output from the edge, while the local task backlog is not empty, which is expressed by $\mathcal{H}_k = \{t | t \in \mathcal{T}_k^0,\ B_k(t+1) > 0\}$, $\mathcal{T}_k^0 = \{t | B'_k(t) \in \mathbf{Z},\ B'_k(t) < B_k(t)\}$; The other is the case that a new task is generated during slot t at device k when its task backlog is empty at the slot beginning, which is denoted by $\mathscr{E}_k = \{t | B_k(t) = 0, Y_k(t) > 0\}$. Therefore, the overall decision epochs for device k are denoted as $\mathcal{E}_k = \mathcal{H}_k \bigcup \mathscr{E}_k$.

D. Local Transmission Queue of Device

After a task of device k is scheduled to be offloaded, device k pushes the task input data into its transmission queue for uploading to the edge server. We normalize the transmission queue length of device k at the beginning of slot t with regard to L_k, which is denoted by $F_k(t)$. Because the tasks are scheduled sequentially, there is at most one task in the transmission queue, i.e., $F_k(t) \in [0,1]$. The evolution of the transmission queue length is given by

$$F_k(t+1) = \begin{cases} \max\left\{F_k(t) - \frac{r_k(t)\tau}{L_k}, 0\right\}, & \text{if } z_k(t)=1 \text{ and } a(t)=1 \\ F_k(t)+1, & \text{if } A_k(t)=1 \\ F_k(t), & \text{otherwise.} \end{cases} \tag{4}$$

The first subequation in (4) means that the transmission queue length of device k is decreased due to the uplink transmission of its task input data when the LTE-U BS is in the transmission phase. The second subequation indicates that the transmission queue length is added by 1 if a new task is to be offloaded to the edge server ($A_k(t)=1$). The epochs of transmission completion of device k are denoted as $\mathcal{T}_k^1 = \{t|F_k(t+1)=0,\ F_k(t)>0\}$ and we have the overall transmission completion epochs for any device as $\mathcal{T}^1 = \bigcup_{k\in\mathcal{K}} \mathcal{T}_k^1$.

E. Edge Computing Queue at Edge Server

After the uplink transmission for the task input of device k is finished, the task is pushed into the edge computing queue. A first-come-first-serve edge computing order is considered. To guarantee that the tasks are served in an appropriate sequence, an edge computing priority $\mathcal{W}_k(t)$ ($k\in\mathcal{K}$) is used for the in-service task from device k at time t. Once any new task finishes its uploading for its input data and enters the edge computing queue, the priority of all tasks at the edge (including the new task) are increased by one, guaranteeing that the earlier tasks are allocated with higher priority. The update of the edge computing priority is denoted by

$$\overline{\mathcal{W}}_k(t) = \begin{cases} \mathcal{W}_k(t)+1, & \text{if } t\in\mathcal{T}^1 \text{ and } k\in\mathscr{C}(t) \\ \mathcal{W}_k(t), & \text{otherwise} \end{cases} \tag{5}$$

where $\mathscr{C}(t)$ is the set containing devices with tasks in the edge computing queue, i.e., $\mathscr{C}(t) = \{k|\mathcal{W}_k(t)>0 \text{ or } t\in\mathcal{T}_k^1\}$. Moreover, for integrating the computing process of tasks at the edge, the decimal part of $\mathcal{W}_k(t)$ is interpreted as the unfinished task fraction of device k's in-service task at the edge server, which is normalized by L_kX_k. The computing-related update for the priority is described as

$$\hat{\mathcal{W}}_k(t) = \begin{cases} \max\left\{\overline{\mathcal{W}}_k(t) - \frac{C_0\tau}{L_kX_k}, \lfloor\overline{\mathcal{W}}_k(t)\rfloor^-\right\}, & \text{if } z_k(t)=3 \\ \overline{\mathcal{W}}_k(t), & \text{otherwise.} \end{cases} \tag{6}$$

In (6), the first subequation means that, if a task is being processed at the edge server ($z_k(t)=3$), the decimal part of its priority value is reduced by the normalized processed fraction. Once a task releases the edge computing resources due to the task accomplishment, its priority value is reset to zero, which is denoted by

$$\mathcal{W}_k(t+1) = \begin{cases} 0, & \text{if } \hat{\mathcal{W}}_k(t)\in\mathbf{Z} \text{ and } z_k(t)=3 \\ \hat{\mathcal{W}}_k, & \text{otherwise.} \end{cases} \tag{7}$$

The task completion epochs of edge computing for device k are denoted by set $\mathcal{T}_k^2 = \{t|\hat{\mathcal{W}}_k(t)\in\mathbf{Z},\ z_k(t)=3\}$. Since the data size of task execution result is normally much smaller than the input data size, the transmission of the output data for all tasks is assumed to take only one time slot [28].

F. Delay Counter

Suppose that task i of device k is generated at T_i^{A}, scheduled at T_i^{S}, and completed at T_i^{E}. The delay of this task is calculated as $d_i = T_i^{\mathrm{E}} - T_i^{\mathrm{A}}$. For recording the delay of each task, a collection of counters should be maintained in each device, which significantly increases the modeling complexity. Fortunately, we can exploit the correlation between two successive tasks in the same device to represent the delay in a more effective way. Suppose that we have only one delay counter D_k representing the number of slots between the generation epoch of the in-service task of device k and the current slot. For task $i+1$ generated at T_{i+1}^{A}, scheduled at T_{i+1}^{S}, and completed at T_{i+1}^{E}, its scheduling delay of task $i+1$ is given by

$$d_{i+1} = T_{i+1}^{\mathrm{E}} - T_{i+1}^{\mathrm{A}} = T_{i+1}^{\mathrm{E}} - T_{i+1}^{\mathrm{S}} + T_{i+1}^{\mathrm{S}} - T_{i+1}^{\mathrm{A}}. \tag{8}$$

If task i has been completed when task $i+1$ is generated, the latter enters an empty task queue and the delay counter of device k is directly used to record the delay of task $i+1$. If the task buffer is not empty when task $i+1$ is generated, the delay counter is being used by the former task i. The history scheduling epochs and ending epochs before task $i+1$ are respectively denoted by $\boldsymbol{T}_i^{\mathrm{S}} = \left(T_{i'}^{\mathrm{S}}\right)_{i'\le i}$ and $\boldsymbol{T}_i^{\mathrm{E}} = \left(T_{i'}^{\mathrm{E}}\right)_{i'\le i}$. Actually, considering that the ending epoch of task i is actually the scheduling epoch of task $i+1$, i.e., $T_{i+1}^{\mathrm{S}} = T_i^{\mathrm{E}}$, we have

$$d_{i+1} = T_{i+1}^{\mathrm{E}} - T_{i+1}^{\mathrm{S}} + T_i^{\mathrm{E}} - (T_i^{\mathrm{A}} + \Delta_i) = T_{i+1}^{\mathrm{E}} - T_{i+1}^{\mathrm{S}} + d_i - \Delta_i \tag{9}$$

where $\Delta_i = T_{i+1}^{\mathrm{A}} - T_i^{\mathrm{A}}$ is the interval between the generations of tasks i and $i+1$. Considering that the task arrivals of each device in a period of interest can be modeled as a stationary stochastic process, the expectation of Δ_i is denoted by $\overline{\Delta}$. We calculate the expected delay of task $i+1$ under the condition of the real measured scheduling and ending epochs history as

$$\mathbb{E}\left(d_{i+1}\middle|\boldsymbol{T}_{i+1}^{\mathrm{S}}, \boldsymbol{T}_{i+1}^{\mathrm{E}}\right) = \underbrace{T_{i+1}^{\mathrm{E}} - T_{i+1}^{\mathrm{S}}}_{\text{after scheduling}} + \underbrace{\mathbb{E}\left(d_i\middle|\boldsymbol{T}_i^{\mathrm{S}}, \boldsymbol{T}_i^{\mathrm{E}}\right) - \overline{\Delta}}_{\text{before scheduling}}. \tag{10}$$

From (10), the expected delay of a task is divided by the time before scheduling and that after scheduling. The expected waiting time before scheduling is calculated based on the expectation of previous task's delay ($\mathbb{E}\left(d_i\middle|\boldsymbol{T}_i^{\mathrm{S}}, \boldsymbol{T}_i^{\mathrm{E}}\right)$) and the average interval of two successive task arrivals ($\overline{\Delta}$). Therefore, we only set one delay counter for each IoT device for counting the expected delay of the in-service task conditioned on the epochs of task scheduling and completion. Upon completing the current task and scheduling the next task, the delay counter is reset to the expected waiting time ($\mathbb{E}\left(d_i \middle| \boldsymbol{T}_i^{\mathrm{S}}, \boldsymbol{T}_i^{\mathrm{E}}\right) - \overline{\Delta}$) before scheduling for the next task. Thus, we obtain the average delay of all tasks, sequentially, by only using the delay counter information of the head-of-line task. The evolution of the delay counter is given by

$$D_k(t+1) = \begin{cases} D_k(t) - \overline{\Delta}, & \text{if } t \in \mathcal{H}_k \\ 0, & \text{if } t \in \mathscr{E}_k \\ D_k(t) + 1, & \text{if } B_k(t) > 0 \text{ and } t \notin \mathcal{E}_k \\ D_k(t), & \text{otherwise.} \end{cases} \tag{11}$$

In (11) the first subequation indicates that a new task of device k is scheduled at the end of slot t and its delay counter is initialized as $D_k(t) - \overline{\Delta}$; the second subequation indicates that when the task buffer is empty after a task completion at the end of slot t, the delay counter is reset to 0; the third subequation indicates that, when the task is being scheduled, the delay is increased by one after each time slot. It is noted that the delay counter can be negative due to the estimation error between the true sampling interval of the two successive tasks and its expectation. However, this potential negative value can be balanced in the long-term evolution. Different from directly estimating the delay information based on LTE-U/Wi-Fi coexistence analysis such as in [29], [30] which needs the specific information of Wi-Fi network, setting the delay counter at each device can provide decision maker with extra real-time information and make the task scheduling strategy more intelligently without the coordination between LTE-U and W-Fi.

IV. Problem Formulation

To capture the dynamic evolution of task-related states, we leverage a Markov decision process (MDP) framework to model the complex interactions among the processes of transmission, computing, and queueing in the MEC system. Basically, an MDP is defined by a tuple of state space $\mathcal{S}$, decision space $\mathcal{A}$, state transition probability function $P := \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$, and reward function $R := \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The system state is defined as $\boldsymbol{s}(t) = (\boldsymbol{z}(t), \boldsymbol{B}(t), \boldsymbol{F}(t), \boldsymbol{\mathcal{W}}(t), \boldsymbol{D}(t), a(t), b(t), m(t))$, where $\boldsymbol{B}(t), \boldsymbol{F}(t)$, $\boldsymbol{\mathcal{W}}(t)$, and $\boldsymbol{D}(t)$ are the vectorized values of the task backlog sizes, transmission queue sizes, edge computing priorities, and delay counters, respectively, of all devices at time t, and their values can be obtained from the derivations in Section III. In each generalized decision epoch, denoted by $t \in \mathcal{E} = \bigcup_k \mathcal{E}_k$, an action $\boldsymbol{A}(t) = [A_k(t)]_{k \in \mathcal{K}}$ is taken. The available action space is dependent on the current state and is expressed as

$$\mathcal{A}(t) = \prod^{\otimes} \mathcal{A}_k(t) \tag{12}$$

where $\prod^{\otimes}$ is the cumulative Cartesian product operation, and $\mathcal{A}_k(t)$ is the sub-space containing the available actions for device k. Specifically, we have

$$\mathcal{A}_k(t) = \begin{cases} \{0,1\} & \text{if } t \in \mathcal{E}_k \\ \{0\} & \text{otherwise} \end{cases} \tag{13}$$

which indicates that the decision space of device k includes both local computing ($A_k(t) = 0$) and offloading ($A_k(t) = 1$) when the current slot is a decision epoch of the device, otherwise, it is just set as 0 to keep the notation consistent.

The state transitions consist of three steps: The first one is to update $\boldsymbol{B}(t)$, $\boldsymbol{F}(t)$, $\boldsymbol{\mathcal{W}}(t)$, and $\boldsymbol{D}(t)$; The second one is to update the channel status based on the transition probability of different states, which depends on the external environment and the LBT setting of the LTE-U BS [31]; The third one is to update the task status, given by (14), shown at the bottom of the next page. In (14), the first case represents that, in a decision epoch, the task status changes into the local computing status if $A_k(t) = 0$ or the transmission status if $A_k(t) = 1$; The second case indicates that the task transits into the edge queueing status after it finishes uploading the task input data and the edge server is occupied; In the third case, if the edge computing queue is empty, the task directly begins to be served by the edge server after its transmission of the task input, or the task status transits from edge queueing status to the edge computing status when other tasks with higher priority have all been completed; The forth case means that a completed task of device k begins to wait at the BS for returning its output data back to device k when the channel is accessible.

After each task is completed under state $\boldsymbol{s}(t)$, a reward $R_k(\boldsymbol{s}(t))$ is generated for device k, including both the positive profit $(1-\eta)f_k$ and the negative delay-related impact $-\eta D_k$, given by

$$R_k(\boldsymbol{s}(t)) = \begin{cases} (1-\eta)f_k - \eta D_k(t), & \text{if } \ t \in \mathcal{T}_k^0 \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

where $\eta \in [0,1]$ is a weight parameter to balance the tradeoff between the profit and delay of the task completion. The reward function can be customized in different forms for the worst-case delay guarantee. For example, in [32], a hard delay deadline is set for offloading or local execution of each task. If a task computing delay exceeds the deadline, the task would be dropped, and a large dropping cost is added into the computing cost function in the form of a weighted negative indicator function on satisfying the required delay bound, to guarantee the worst-case delay to the maximum extent.

Meanwhile, a device-related cost G_k, is generated when each task of device k is executed at the edge server for capturing the cost of task transmission, service maintenance, e.g., energy consumption, and wear and tear of the equipment [33]. The cost function is represented by

$$C_k(\boldsymbol{s}(t)) = \begin{cases} G_k, & \text{if } \ t \in \mathcal{T}_k^0 \text{ and } z_k(t) \neq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

Thus, the overall reward and cost for this network are defined as

$$R(\boldsymbol{s}(t)) = \sum_{k\in\mathcal{K}} R_k(\boldsymbol{s}(t)) \tag{17}$$

$$C(\boldsymbol{s}(t)) = \sum_{k\in\mathcal{K}} C_k(\boldsymbol{s}(t)). \tag{18}$$

Because the system evoloves in a Markov way, to maximize the expectation of the long-term average reward subject to edge computing cost constraint, a CMDP problem for choosing the optimal task scheduling action in each time slot can be formulated as P0,

$$\begin{aligned}\mathbf{P0}:\ \max_{\pi}\ & \mathbb{E}\left[\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T} R(\boldsymbol{s}(t))\bigg|\pi\right]\\ \text{s.t.}\ \ & \mathbb{E}\left[\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T} C(\boldsymbol{s}(t))\bigg|\pi\right]\le \bar{C}\end{aligned} \tag{19}$$

where π is a stationary policy mapping from state $\boldsymbol{s}(t)$ to action $\boldsymbol{A}(t)$. For the scenario with some urgent tasks of high priority, although the delay calculation in (11) for each task may be biased due to the reordered task computing, for one fixed policy, it can still complete the same number of tasks within the same time regardless of the serving order, i.e., the sum of the task completion profit and the sum of the task completion delay are the same. Therefore, the average performance of P0 in (19) under the above formulation is still the same as that of the scenario without reordering the tasks.

V. Algorithm for Task Scheduling

If we know the environment information, i.e., the state transition probabilties $P(\boldsymbol{s}|\boldsymbol{s}',\boldsymbol{A}')$, the CMDP defined as P0 can be equivalently transformed to a linear programming (LP) problem [34], given by

$$\max_{\boldsymbol{x}}\ \sum_{\boldsymbol{s}\in\mathcal{S}}\sum_{\boldsymbol{A}\in\mathcal{A}(\boldsymbol{s})} x(\boldsymbol{s},\boldsymbol{A})R(\boldsymbol{s}) \tag{20a}$$

$$\text{s.t.}\ \sum_{\boldsymbol{s}'\in\mathcal{S}}\sum_{\boldsymbol{A}'\in\mathcal{A}(\boldsymbol{s}')} x(\boldsymbol{s}',\boldsymbol{A}')P(\boldsymbol{s}|\boldsymbol{s}',\boldsymbol{A}') = \sum_{\boldsymbol{A}\in\mathcal{A}(\boldsymbol{s})} x(\boldsymbol{s},\boldsymbol{A}),\ \forall s\in\mathcal{S} \tag{20b}$$

$$\sum_{\boldsymbol{s}\in\mathcal{S}}\sum_{\boldsymbol{A}\in\mathcal{A}(\boldsymbol{s})} x(\boldsymbol{s},\boldsymbol{A}) = 1 \tag{20c}$$

$$\sum_{\boldsymbol{s}\in\mathcal{S}}\sum_{\boldsymbol{A}\in\mathcal{A}(\boldsymbol{s})} x(\boldsymbol{s},\boldsymbol{A})C(\boldsymbol{s}) \le \bar{C}. \tag{20d}$$

In (20), variable $x(\boldsymbol{s},\boldsymbol{A})$ can be interpreted as the long-run visiting probability to the state-action pair $(\boldsymbol{s},\boldsymbol{A})$, $R(\boldsymbol{s})$ is the reward under state $\boldsymbol{s}$, and $C(\boldsymbol{s})$ is the generated cost under state $\boldsymbol{s}$. Constraint (20b) is the global balance equations for the steady-state probabilities, (20c) implies that the sum of all state-action pairs' probabilities is equal to 1, (20d) shows that the edge computing cost should be limited by cost budget $\bar{C}$. For brevity, time index t is written as the subscript in the following. Different from the deterministic policy in traditional MDP, the optimality can be achieved only when the policy is a mapping from a state to a distribution of available actions in the CMDP, defined by

$$P(\boldsymbol{A}_t=\boldsymbol{A}|\boldsymbol{s}_t=\boldsymbol{s}) = \frac{x(\boldsymbol{s},\boldsymbol{A})}{\sum_{\boldsymbol{A}'\in\mathcal{A}(\boldsymbol{s})} x(\boldsymbol{s},\boldsymbol{A}')}. \tag{21}$$

The LP problem in (20) can be solved in general via methods such as simplex algorithm and interior-point algorithm [34]. However, it is impractical to apply an LP-based algorithm in this scenario since the transition probabilities (i.e., $P(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{A})$) cannot be known as a priori, and the high dimensional state representation leads to an exponentially increasing space complexity for $x(\boldsymbol{s},\boldsymbol{A})$.

A. Task Scheduling Without Edge Computing Cost Constraint

When the state transition probabilities are not accessible, reinforcement learning (RL) can be leveraged to gradually extract the optimal policy by interacting with the environment. However, the constrained RL is an elusive challenge due to the difficulty of transferring from value-based formulation in RL to the LP-based formulation in CMDP. In order to find a solution to the cost-constrained task scheduling problem, we first design an algorithm to deal with the case in which the average edge computing cost constraint is removed. The simplified problem for maximizing the long-term average reward is then approximated by maximizing the accumulated discounted reward [35], i.e.,

$$\mathbf{P1}:\ \max_{\pi}\ \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t R(\boldsymbol{s}_t)\bigg|\pi\right] \tag{22}$$

where γ (<1) is the discount factor to represent that the learning agent pays more attention to the current reward than the future reward. The larger γ, the closer the optimal solution of P1 approaches the undiscounted one.

When the state transition probabilities are unknown, Q-learning is a common model-free RL method which can

$$z_k(t+1) = \begin{cases} A_k(t), & \text{if } t\in\mathcal{E}_k\\ 2, & \text{if } z_k(t)=1 \text{ and } t\in\mathcal{T}_k^1 \text{ and } k\neq \arg\max_{k\in\mathcal{K}}\mathcal{W}_k(t+1)\\ 3, & \text{if } z_k(t)=1 \text{ and } t\in\mathcal{T}_k^1 \text{ and } k= \arg\max_{k\in\mathcal{K}}\mathcal{W}_k(t+1),\ \text{or}\\ & \text{if } z_k(t)=2 \text{ and } t\in\mathcal{T}^2 \text{ and } k= \arg\max_{k\in\mathcal{K}}\mathcal{W}_k(t+1)\\ 4, & \text{if } z_k(t)=3 \text{ and } t\in\mathcal{T}_k^2\\ z_k(t), & \text{otherwise.}\end{cases} \tag{14}$$

learn the optimal policy via successively updating its estimation about the Q-values based on the interaction with the external environment [36]. The Q-values can be interpreted as an evaluation for how good a state-action pair is. Therefore, when the estimation about the Q-values is accurate, the action with the largest Q-value, i.e., $\arg\max_{\boldsymbol{A}\in\mathcal{A}(\boldsymbol{s})} Q(\boldsymbol{s},\boldsymbol{A})$, should be taken when the learning agent observes state $\boldsymbol{s}$. The traditional tabular Q-learning maintains the estimation for Q-values by constructing a Q-value table for all state-action pairs with size $\sum_{\boldsymbol{s}\in\mathcal{S}} |\mathcal{A}(\boldsymbol{s})|$, where $|\cdot|$ is the set cardinality. In each decision epoch, an action is chosen probabilistically based on the current state $\boldsymbol{s}_t$ and the Q-value table, and the probability for choosing action $\boldsymbol{A}_t$ is expressed as

$$P(\boldsymbol{A}_t|\boldsymbol{s}_t)=\begin{cases}1-\epsilon+\dfrac{\epsilon}{|\mathcal{A}(\boldsymbol{s}_t)|}, & \text{if } \boldsymbol{A}_t=\arg\max_{\boldsymbol{A}} Q(\boldsymbol{s}_t,\boldsymbol{A})\\ \frac{\epsilon}{|\mathcal{A}(\boldsymbol{s}_t)|}, & \text{otherwise}\end{cases} \tag{23}$$

where ϵ is a tradeoff coefficient balancing between exploitation (choosing the most promising action) and exploration (trying a random action). After the action, a reward, $R(\boldsymbol{s}_t)$, is fed back and the next state $\boldsymbol{s}_{t+1}$ is observed. Based on the full backup equation, only the Q-value for state-action pair $(\boldsymbol{s}_t,\boldsymbol{A}_t)$ in the Q-table is updated [37], given by

$$\begin{aligned}&Q(\boldsymbol{s}_t,\boldsymbol{A}_t)\\&(1-\beta)Q(\boldsymbol{s}_t,\boldsymbol{A}_t)+\beta\left(R(\boldsymbol{s}_t)+\gamma\max_{\boldsymbol{A}\in\mathcal{A}(\boldsymbol{s}_{t+1})} Q(\boldsymbol{s}_{t+1},\boldsymbol{A})\right)\end{aligned} \tag{24}$$

where β is the learning rate which balances the learning step length and the learning accuracy. Such an action selection and Q-value updating mechanism can guarantee that the optimal policy can be obtained with probability 1 [37]. However, the traditional tabular Q-learning is considerably space-costly for recording the Q-values of all possible state-action pairs, and has the curse of dimension problem when it is applied in our task scheduling problem with multi-dimensional state. To overcome the high space complexity, we adopt a DQL method [38] instead, to approximate Q-values via a DNN, i.e., $Q(\cdot;\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the network parameter consisting of the weights and biases for connecting the neural units of different layers in the DNN. The detailed DQL for task scheduling without the edge computing cost constraint is given in Algorithm 1.

The proposed algorithm is executed at the beginning of each slot by the LTE-U BS which is able to observe the system states via the inherent reporting scheme of LTE. There are two DQNs, i.e., the evaluation network and the target network, and their parameters are denoted by $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$, respectively. The evaluation network can directly obtain an evaluation Q-value for a state-action pair via a forwarding calculation. Alternatively, the target Q-value for a state-action pair sample $(\boldsymbol{s}_j,\boldsymbol{A}_j)$ can be estimated via the full backup equation in (24), i.e.,

$$y_j=R_j+\gamma\max_{\boldsymbol{A}\in\mathcal{A}(\boldsymbol{s}_{j+1})}(Q'(\boldsymbol{s}_{j+1},\boldsymbol{A};\boldsymbol{\theta}')) \tag{25}$$

Algorithm 1 DQL Algorithm for Task Scheduling

Initialize:
Set an empty replay memory Ψ with capacity M;
Randomly initialize the parameter of the evaluation DQN (Q) as $\boldsymbol{\theta}$;
Initialize the parameter of target DQN (Q') as $\boldsymbol{\theta}'=\boldsymbol{\theta}$;
Initialize $\epsilon=1$ and state $\boldsymbol{s}_0$;
while $t<T$ **do**
  if $t\in\mathcal{E}$ **then**
    Choose action based on (23);
    Execute the action and observe the reward R_t and the next state $\boldsymbol{s}_{t+1}$;
  else
    Set $\boldsymbol{A}_t=\boldsymbol{0}$;
  end if
  Decay the exploration probability $\epsilon\leftarrow\max\{\epsilon\phi,\epsilon_{\min}\}$;
  Store transition vector $(\boldsymbol{s}_t,\boldsymbol{A}_t,R_t,\boldsymbol{s}_{t+1})$ in Ψ;
  Sample random minibatch of transitions $(\boldsymbol{s}_j,\boldsymbol{A}_j,R_j,\boldsymbol{s}_{j+1})$ from Ψ;
  Calculate the target Q value of $(\boldsymbol{s}_j,\boldsymbol{A}_j)$, i.e., y_j, based on (25);
  Perform a gradient decent on $\boldsymbol{\theta}$ for minimizing $|y_j-Q(\boldsymbol{s}_j,\boldsymbol{A}_j;\boldsymbol{\theta})|^2$;
  $t=t+1$;
  Every N steps set $\boldsymbol{\theta}'=\boldsymbol{\theta}$;
end while
Output: Evaluation DQN with parameter $\boldsymbol{\theta}$.

where R_j is the reward feedback under state-action pair $(\boldsymbol{s}_j,\boldsymbol{A}_j)$, $\boldsymbol{s}_{j+1}$ is the following state, and the Q-value for $\boldsymbol{s}_{j+1}$ is derived from the forwarding calculation based on the target network. The action selection in DQL is the same as in (23), where the Q-value calculation is based on the evaluation network. The exploration probability ϵ gradually decays with a rate ϕ to transfer its concern from exploration to exploitation. After the action selection, a transition sample including the current state, current action, reward feedback, and the next state, is stored in the replay memory. The transition samples in the replay memory are randomly selected for the gradient-decent-based training of the evaluation DQN for shrinking the gaps between the evaluation Q-values and the target Q-values. After every N steps, the target DQN gets updated using $\boldsymbol{\theta}$, the parameter of the trained evaluation DQN. The design of two separated DQNs reduces the oscillations and avoids the divergence of the policy. By iteratively training the evaluation DQN, the agent can gradually obtain the optimal policy [39].

B. Task Scheduling With Edge Computing Cost Constraint

Developing a solution for the constrained task scheduling is mainly inspired by [40], where the constraint can be integrated into the reward function by an appropriate weight, λ, leading to a new MDP with a revised reward and no constraint. From the perspective of the DQN formulation with parameter $\boldsymbol{\theta}$, the original problem P0 with constraint can be expressed as

$$\textbf{P2:}\ \max_{\boldsymbol{\theta}} V_{\pi(\boldsymbol{\theta})} \quad \text{s.t. } \mathbb{E}(C(t)|\pi(\boldsymbol{\theta}))\le\bar{C} \tag{26}$$

where $\pi(\boldsymbol{\theta})$ is the policy that the action with the maximal Q value for the current state is selected based on a DQN parameterized by $\boldsymbol{\theta}$, and $V_{\pi(\boldsymbol{\theta})} = \lim_{T\to\infty} \frac{1}{T}\, \mathbb{E}\left(\sum_{t=0}^{\infty} R(\boldsymbol{s}_t)\big|\pi(\boldsymbol{\theta})\right)$ is the expectation of the long-term average reward under policy $\pi(\boldsymbol{\theta})$. The objective of task scheduling is to maximize the average reward while guaranteeing the edge computing cost constraint. The Lagrange function and the dual function of prime problem P2 are expressed, respectively, as

$$L(\boldsymbol{\theta}, \lambda) = V_{\pi(\boldsymbol{\theta})} - \lambda\left(\mathbb{E}(C|\pi(\boldsymbol{\theta})) - \bar{C}\right), \quad \lambda > 0 \tag{27}$$

$$g(\lambda) = \sup_{\boldsymbol{\theta}}\left\{V_{\pi(\boldsymbol{\theta})} - \lambda\left(\mathbb{E}(C|\pi(\boldsymbol{\theta})) - \bar{C}\right)\right\}. \tag{28}$$

Because the underlying problem is a CMDP with a standard LP form in (20), it is obvious that the dual problem (i.e., $\min_{\lambda>0} g(\lambda)$) has the same solution as the prime problem, and the dual function is reformulated as

$$g(\lambda) = \sup_{\boldsymbol{\theta}} \lim_{T\to\infty} \frac{\mathbb{E}\left[\sum_{t=0}^{T} r(\boldsymbol{s}_t, \boldsymbol{A}_t) - \lambda \sum_{t=0}^{T}\left(C(\boldsymbol{s}_t, \boldsymbol{A}_t) - \bar{C}\right)\bigg|\pi(\boldsymbol{\theta})\right]}{T}. \tag{29}$$

Theorem 1: Function $g(\lambda)$ is continuous even when the scheduling policy is restricted to be deterministic policy, which means that, for each state, there is only one optimal action and random policy is not allowed.

Proof: See Appendix A. □

Therefore, the optimal expected reward in P2 is the same as the optimal value of $g(\lambda^*)$ when the optimal Lagrange multiplier λ^* is found. Accordingly, the constrained scheduling problem can be transformed as another scheduling problem without constraint, but with a revised reward function, given by

$$R'(\boldsymbol{s}_t) = R(\boldsymbol{s}_t) - \lambda^*(C(\boldsymbol{s}_t) - \bar{C}). \tag{30}$$

It is noted that the revised MDP and the original MDP have the same state space, action space, and transition probabilities. In order to obtain the optimal Lagrange multiplier and the optimal scheduling policy under the average edge computing cost constraint, an iteration-based algorithm is developed based on the subgradient theory, which is presented in Algorithm 2. The key idea is to use the counted average edge computing cost to update the Lagrangian multiplier, where α_i is the step size for updating of λ, and $[\cdot]^+$ is the non-negative function.

Assumption 1: In Algorithm 2, the counted average edge computing cost, C_i, has mean of $\mu_i = \mathbb{E}\left[C|\pi(\boldsymbol{\theta}_i)\right]$ and relatively small variance σ_i with $\sigma_i \le M\mu_i$, where M is a large positive constant.

Assumption 2: The DQL algorithm without constraint in Algorithm 1 can obtain the optimal policy when any revised reward function with determined λ is given.

For Assumption 1, if we count the average edge computing cost in a long term, the variance can be small enough according to the law of large numbers; For Assumption 2, many existing works prove that the DQL algorithm is effective in addressing such scheduling problems [41].

Algorithm 2 CDQL Algorithm for Task Scheduling With Edge Edge Computing Cost Constraint

Initialize:
Set $i = 0$, $\lambda_i = 0$, and the step size $\alpha_i = 0.5$;
while 1 **do**
  Set the reward function as: $R_i(\boldsymbol{s}) = R(\boldsymbol{s}) - \lambda_i C(\boldsymbol{s})$;
  Run Algorithm 1 with reward function R_i and obtain the optimal policy $\pi(\boldsymbol{\theta}_i)$;
  Count the average edge computing cost C_i;
  Update λ as: $\lambda_{i+1} = \left[\lambda_i + (C_i - \bar{C})\alpha_i\right]^+$;
  Update the step size as $\alpha_{i+1} = \alpha_0/(i+1)$;
  $i = i + 1$;
end while
Output: λ_i and DQN with parameter $\boldsymbol{\theta}$

Theorem 2: In Algorithm 2, when $i \to \infty$, λ converges to optimal λ^* with probability 1 under Assumptions 1 and 2.

Proof: See Appendix B. □

VI. Realization of the CDQL

To enhance the performance and efficiency of the proposed CDQL-based task scheduling algorithm, we make the following customized modifications for its realization:

- **Structure of the DQN:** The output dimension in the conventional DQN is usually the same as the size of action space, in order to identify the difference for adopting different available actions. However, the available action space in our problem varies with different states and the generalized action space is 2^K, which indicates that constructing a DQN where the last output layer has the same size as the action space is unrealistic in the practical implementation. Therefore, we construct the output layer as shown in Fig. 3, where the Q-value for a state-action pair is calculated based on the action value layer, action mask, and the base value node. The base value node can be considered as a basic estimation of the average Q-value of the current state, while the action value layer with size $2K$ depicts the effect of each device's action on the overall Q-value. Specifically, the $(2k-1)$-th or $2k$-th node in the act-value layer represents the Q-value bias when action 0 or 1 is adopted for device k. The input action makes up the action mask layer by a binary coding, and extracts only the act-value of the selected action to the Q-value output. The Q-value under this modified structure is calculated as

$$Q(\boldsymbol{s}, \boldsymbol{A}; \boldsymbol{\theta}) = \text{Base\_value}(\boldsymbol{s}; \boldsymbol{\theta}) + \text{Action\_value}(\boldsymbol{s}; \boldsymbol{\theta})\text{Action\_mask}(\boldsymbol{A})^{\mathrm{T}}. \tag{31}$$

- **Preprocessing of data samples:** The traditional memory replay of DQL records all past samples of the state-action pairs, rewards, and future states. However, in our problem, there are many time slots in which no action is required. In other words, regardless of the adopted policy, these transitions are the same, which means that feeding these transition samples in the learning process cannot provide any extra information for the policy learning but leads to some redundant processing time. To utilize the data samples in a more efficient way, the samples for training are compressed with form $(\boldsymbol{s}_t, \boldsymbol{A}_t, T, R_{t\sim T}, \boldsymbol{s}_{t+T})$, where $\boldsymbol{s}_t$ is a state under a decision epoch, $\boldsymbol{A}_t$ is its corresponding action, T is the number of the slots between the former decision epoch and the later decision epoch, $\boldsymbol{s}_{t+T}$ is the state under the later decision epoch, and $R_{t\sim T} = \sum_{i=t}^{t+T} \gamma^{i-t} R_i$ is the accumulated discounted rewards between slot t and $t+T$. With the modified memory replay design, the target Q value for state-action pair $(\boldsymbol{s}_t, \boldsymbol{A}_t)$ is calculated as

$$y = R_{t\sim T} + \gamma^T \max_{\boldsymbol{A}\in\mathcal{A}(\boldsymbol{s}_{t+T})} Q(\boldsymbol{s}_{t+T}, \boldsymbol{A}; \boldsymbol{\theta}). \quad (32)$$

- **Pretraining for the Q-value:** The objective of training the network is to 1) approximate the Q values for different state-action pairs with high accuracy and 2) learn the optimal policy based on the trained DQN. However, the adopted policy is not stationary in the initial stage of the DQN training and affects both input (i.e., input action) and output (i.e., output Q values) of the DQN. Such a recurrent effect of the policy makes it necessary to choose a very small learning rate (e.g., 10^{-6}) to guarantee the convergence and stability of the concurrent learning of both Q-value and policy, which leads to very time-consuming training process. Therefore, we treat the policy as a random policy in the initial training stage to reduce the impact of the policy dynamics, and set learning rate at a relatively large value (e.g., 10^{-3}). At the same time, the accumulated discounted reward, $\hat{Q}(t) = \sum_{i=t-1000}^{t} R_i$ is counted as an approximation of Q-value under the random policy. After the output Q-value roughly approximates the real discounted accumulated reward, e.g., $0.9 < \frac{Q(\boldsymbol{s}_t, \boldsymbol{A}_t; \boldsymbol{\theta})}{\hat{Q}(t)} < 1.1$, we reduce the learning rate to a small value to further optimize the policy.

VII. Performance Evaluation

In this section, simulation results are presented to investigate the performance of the proposed computing offloading scheme. A python-based simulator including the LBT, data transmission, and computing is adopted for the simulation. In the simulation, we consider an LTE-U-enabled IoT network consisting of a BS with the coverage radius of 100m. There are in total 16 randomly distributed IoT devices associated with the BS. The LTE-U network is operated on the band of 5GHz and the distance-dependant path loss model is given by [42]

$$\mathrm{PL}(R) = 38.46 \log_{10}(R) + 20 \log_{10}(R) + 0.7R \quad (33)$$

where R is the distance from the device to the LTE-U BS. Small-scale Rayleigh fading is also considered, where channel power gain follows an exponential distribution with unit mean. The noise power spectrum density is -174dBm/Hz.

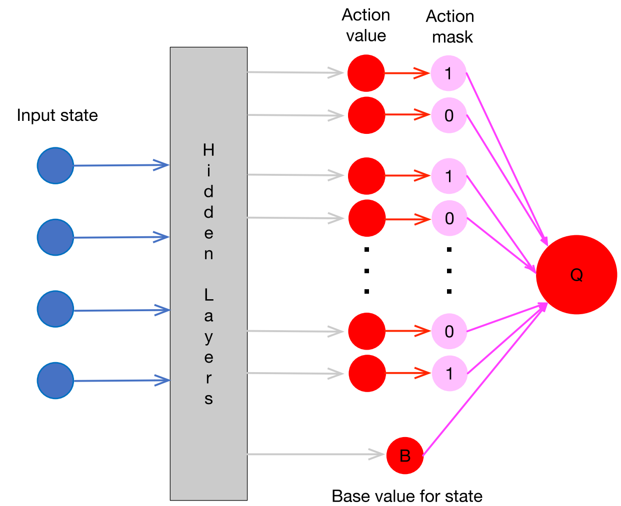


Fig. 3. The modified structure of DQN.

The dedicated bandwidth for the computing offloading service is 1.4 MHz and is equally allocated to all devices with transmission demand. There are 6 Wi-Fi devices with full-buffer traffic model randomly located in the network coverage area, employing the IEEE 802.11ac distributed coordination function (DCF) protocol for channel access with the same configuration in [26]. The task input data size of different devices (L_k) follows a uniform distribution from 15 Kbits to 20 Kbits. The workloads for all devices are set as 200 cycles/bit. The computing capacity of the edge server and the IoT devices are 5 G cycles/s and 500 M cycles/s. If not specified, the defaulted task arrival rate at each device is set as 100/s. There are two backoff stages for LTE-U's LBT-based channel access and the values of the backoff windows are set as 8 and 16 as the channell access priority 2 in [43]. The time duration of each successful transmission phase is 10ms. The cost for computing a task at the edge server is set as 5 and the edge computing cost budget is 3/ms. The duration of each slot is 1ms. The reward of completing a task is set as a random value between 6 and 9 for different devices. The weight for delay, η, is set as 0.1, and the discount factor is set as $\gamma = 0.99$.

The DQN is implemented based on Tensorflow [44], which is an open-source library. Unless otherwise specified, the structure and setting for the DQNs are set as follows: There are three hidden layers with (256, 128, 64) neural units in the DQN structure of Fig. 3. ReLu is the activation function for all the hidden layers and there is no nonlinear activation function for other layers. The replay memory can store $M = 5000$ transition samples and the training batch size is set as 64. The parameter of the target DQN is replaced after every $N = 50$ slots. The learning rates for the pretraining and training are set as 10^{-3} and 10^{-6}, respectively. Every learning episode consists of 5×10^5 time slots. In one eposide learning, the naive Q-learning without the customized modifications is selected as the benchmark. In addition, a Lyapunov optimizatin (LO)-based scheduling algorithm is set as the task scheduling benchmark. Because the delay term in the reward function (15) depends on the stochastic channel states in the following time slots, and cannot be determined in the current slot in

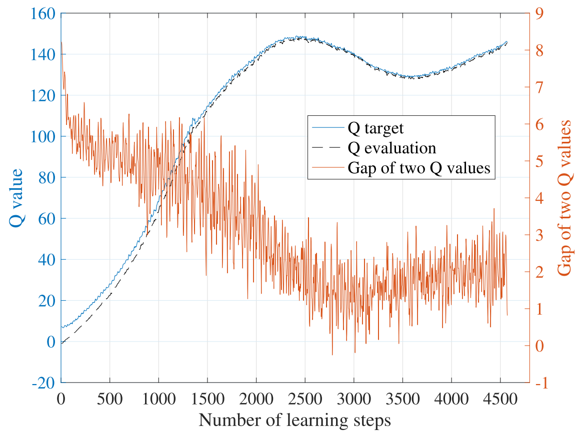


Fig. 4. Q values in the initial training stage without pretraining.

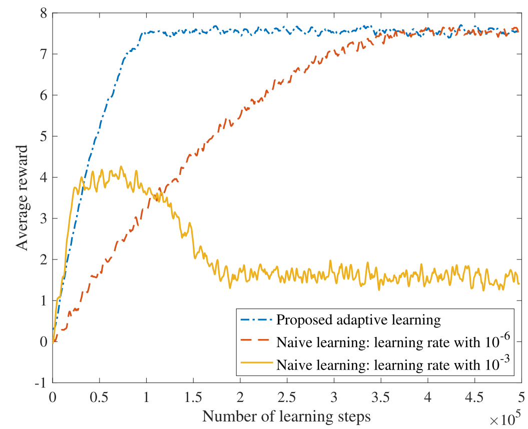


Fig. 5. Training process with different learning rate settings.

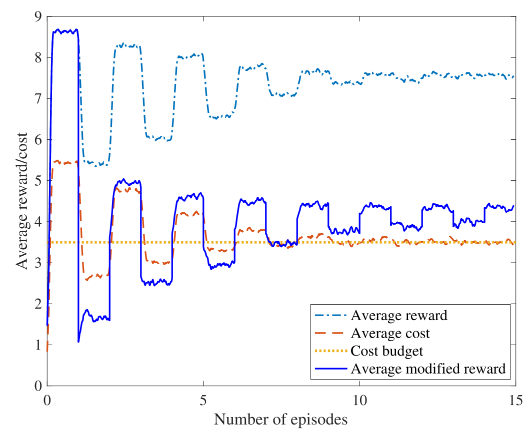


Fig. 6. Convergence of Algorithm 2.

which the decision is made. Therefore, for the benchmark comparison, the delay-related term in (15) is estimated based on the environment statistics. For guaranteeing the average computing cost within the computing budget, a virtual queue method is applied [45].

In Fig. 4, we compare the evaluation Q-values with the target Q-values in the early training phase when there is no pretraining for Algorithm 1, where both Q-values are averaged by a moving window of size 10. It is shown that the Q-value of the target network is always higher than that of the evaluation network in the initial training stage. Actually, such gap is mainly due to their underestimations of the Q-values with the small weight initialization for the two DQNs. Recall that the target Q-value is calculated by $y_j = R_j + \gamma \max_{\boldsymbol{A}' \in \mathcal{A}(\boldsymbol{s}_{j+1})} (Q'(\boldsymbol{s}_{j+1}, \boldsymbol{A}'; \boldsymbol{\theta}'))$. Although the future part of the target Q-value is also underestimated as that of the evaluation Q-value, the target Q-value has the instant reward (usually being positive) feedback as a correction, making the Q value of the target network larger than that of the evaluation network. When a small learning rate is applied for the stable convergence of the DQN, many learning steps would be wasted to reduce such a gap, which makes the training inefficient. Therefore, it is essential to design an adaptive learning rate scheme for the DQN to properly estimate the Q-values in the early training stage.

In Fig. 5, the reward convergence process in the first eposide (with $\lambda = 0$) is given under our proposed Q-learning algorithm with customized modifications and the naive Q-learning algorithm. It can be found that when the learning rate is high (e.g., 10^{-3}), the naive learning process is highly unstable and even becomes worse after training. When the learning rate is small (e.g., 10^{-6}), although the average reward can coverge to a steady and high level, the learning process is considerably slow. However, in our proposed learning with customized modifications, the learning rate is adaptive, i.e, a high learning rate in the pretraining and a low learning rate after the pretraining, the achieved average performance is much better and stable, and takes only less than half time of the learning with fix rate of 10^{-6}, validating the effectiveness of our proposed modifications.

The convergence process of Algorithm 2 is shown in Fig. 6, where the average modified reward, average reward, and average cost are given. When the learning agent keeps receiving the cost feedback via the interaction with the environment and adapting the weight for edge computing cost, the average cost gradually matches the cost budget, and all average modified reward, average reward and average cost become stable, validating the effectiveness of Algorithm 2 for guaranteeing the cost budget. Note that pretraining is applied only in the first episode, while the learning in the subsequent episodes is directly trained based on the DNN parameters in the former episode. However, we can see the learning process can converge quickly when the target (accumulative revised reward function) is changed. It is because that some learned features in the hidden layers can be shared as common knowledge among the learning models with different objectives. Such a phenomena can be interpreted as the advantage of transfer learning [46].

In Fig. 7, we compare the performance of our proposed algorithm with that of the LO algorithm. One can notice that, for both algorithms with the same average edge computing cost constraint, as the task arrival rate increases, the average task profit increases with a decreasing marginal gain due to the gradually saturated communication and computing capability of the system, while the average delay increases considerably. However, our proposed CDQL algorithm always

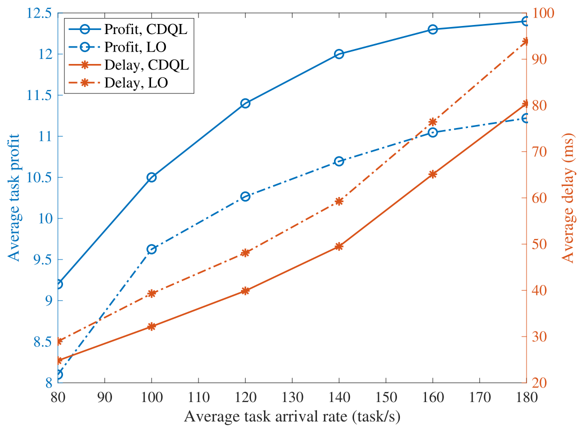


Fig. 7. Performance comparison between CDQL and LO.

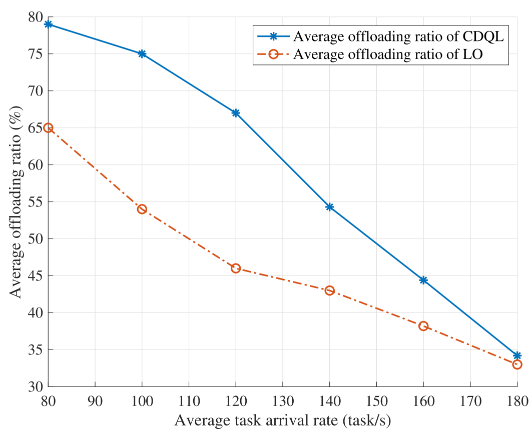


Fig. 8. Average offloading ratio versus task arrival rates.

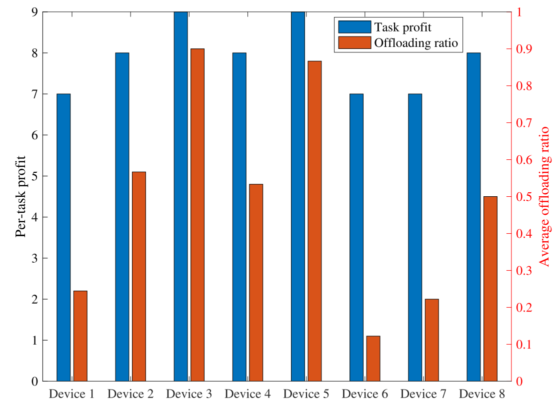


Fig. 9. Offloading ratio and task profit for different IoT devices.

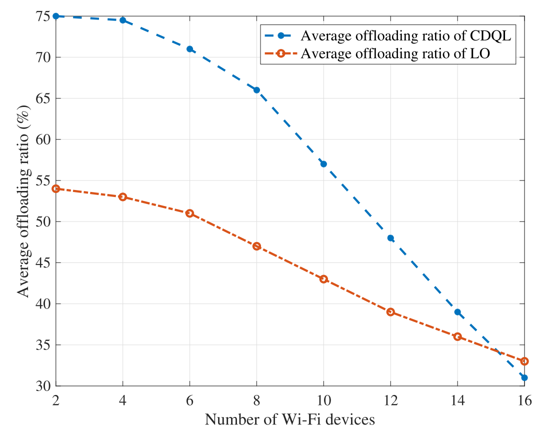


Fig. 10. Average offloading ratio versus number of Wi-Fi devices.

outperforms the LO algorithm in terms of both average profit and delay. The reasons are two-folded: Firstly, the LO algorithm suffers from the inaccuacy of the reward based on the estimated task completion delay; Secondly, the main idea of the LO algorithm is to utilize the relationship between the queue drift and the adopted actions, thus cannot exploit the environment dynamics, while our proposed algorithm can learn the channel state evolution and the task arrival pattern, leading to higher task completion profit and lower delay.

The average offloading ratios for both CDQL algorithm and LO algorithm with different task arrival rates are shown in Fig. 8, where both algorithms maintain the same edge computing cost budget. Due to more intelligent scheduling for task offloading, the proposed CDQL algorithm achieves a higher offloading ratio, leading to more efficient utilization of the edge computing resources, especially when the task arrival rate is small. However, when the task arrival rate increases, the utilization of edge computing resources becomes gradually saturated.

Figure 9 compares the average offloading ratio with per-task completion profit for randomly chosen 8 IoT devices in the LTE-U network when the task arrival rate is set as 160/s. It is found out that a device with a higher task profit can have a larger offloading ratio. Because the computing capability of the edge server is much powerful than the IoT device, processing task at the edge server is usually a more efficent way than local computing. By using our proposed task scheduling algorithm, the system can intelligently allocate more computing resources (i.e., higher offloading ratio) to those devices with higher profit because the proposed algorithm can exploit the diversity among different devices, the system state and the channel availability information.

Figure 10 illustrates the average offloading ratios when the LTE-U-enabled IoT coexisting with different number of Wi-Fi devices. It can be found out when the number of Wi-Fi devices is small, the offloading ratio maintains a stable level even if the number of Wi-Fi devices increases. It is because under such a condition the edge computing cost budget is the main bottleneck for offloading tasks. When the number of Wi-Fi devices further increases, the unlicensed channel becomes more crowded and offloading tasks to the edge server may lead to a high transmission delay. Therefore, offloading may not be a more efficient selection and the offloading ratio decreases. However, it can be seen that the LO algorithm is not as intelligent as our proposed algorithm in terms of adapting the offloading ratio to the network environment.

VIII. Conclusion

In this paper, we propose an RL-based method to solve the task scheduling problem for the computing offloading in an LTE-U-enabled IoT network. We aim to maximize the average reward that integrates both task completion profit and task delay, while guaranteeing average edge computing cost constraint. We formulate the scheduling problem as a CMDP to capture the dynamics of the unlicensed channels' availability and the task traffic. To deal with the exponentially increased space complexity due to the high state dimensions in conventional Q-learning, we utilize a DQL-based method to approximate Q values of different state-action pairs, instead of using the traditional tabular method. Based on the duality theory, a CDQL framework is proposed to integrate the average edge computing cost constraint into the DQL framework. Specifically, we define a revised reward function that combines the original reward and the edge computing cost weighted by an appropriately determined Lagrange multiplier, such that the constraint can be removed in the CDQL framework. In addition, a sub-gradient-based method is proposed to search for the optimal Lagrange multiplier, and the convergence of the searching algorithm is proved. Simulation results demonstrate that our proposed algorithm considerably improves the system performance in terms of the task completion profit and the task completion delay. For the future work, we will extend our work to the scenario where each device can learn the optimal task scheduling policy in a distributed way via multi-agent reinforcement learning.

Appendix A

To prove that $g(\lambda)$ is a continuous function, we should prove that for any λ and any ϵ, there exists δ such that, as long as $\|\lambda' - \lambda\| \leq \delta$, we have $|g(\lambda) - g(\lambda')| \leq \epsilon$, where $\|\cdot\|$ and $|\cdot|$ are Euclidean norm function and absolute-value function, respectively. The available policy set is denoted as $\mathcal{L} = \{1, 2, .., L\}$, where $L \leq |\mathcal{A}|^{|\mathcal{S}|}$ is the number of all available deterministic policies, and is a very large but finite positive integer. The optimal policy for λ is denoted as $_\lambda$, i.e.,

$$\begin{aligned} & l_\lambda \\ & = \arg\max_{l\in\mathcal{L}} \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\sum_{t=0}^{T} R(\boldsymbol{s}_t) - \lambda \sum_{t=0}^{T}(C(\boldsymbol{s}_t) - \bar{C}) \middle| \pi(\boldsymbol{\theta})\right]. \end{aligned} \tag{34}$$

For each deterministic policy $l \in \mathcal{L}$, function $L(\boldsymbol{\theta}(l), \lambda)$ in (27) with a given $\boldsymbol{\theta}$ is continuous with λ, where $\boldsymbol{\theta}(l)$ is the parameter for policy l. Therefore, we conclude that there exists δ_l such that, for any ϵ, as long as $\|\lambda' - \lambda\| \leq \delta_l$, $|L(\boldsymbol{\theta}(l), \lambda) - L(\boldsymbol{\theta}(l), \lambda')| \leq \epsilon$. It is natural to postulate that we choose $\delta = \min_{l\in\mathcal{L}} \delta_l$. For any λ' that satisfies $\|\lambda' - \lambda\| \leq \delta$, we can have

$$\begin{aligned} g(\lambda) - g(\lambda') &= L(\boldsymbol{\theta}(l_\lambda), \lambda) - L(\boldsymbol{\theta}(l_{\lambda'}), \lambda') \\ &\overset{(a)}{\leq} L(\boldsymbol{\theta}(l_\lambda), \lambda') + \epsilon - L(\boldsymbol{\theta}(l_{\lambda'}), \lambda') \\ &\overset{(b)}{\leq} L(\boldsymbol{\theta}(l_\lambda), \lambda') + \epsilon - L(\boldsymbol{\theta}(l_\lambda), \lambda') \leq \epsilon. \end{aligned} \tag{35}$$

In (35), inequality (a) holds because $L(\boldsymbol{\theta}(l_\lambda), \lambda) \leq L(\boldsymbol{\theta}(l_\lambda), \lambda') + \epsilon$ as $\|\lambda' - \lambda\| \leq \delta \leq \delta_{l_\lambda}$ and inequality (b) holds because $\boldsymbol{\theta}(l_{\lambda'})$ is the optimal policy parameter over any other parameter under Lagrangian multiplier λ'.

Further,

$$\begin{aligned} g(\lambda) - g(\lambda') &= L(\boldsymbol{\theta}(l_\lambda), \lambda) - L(\boldsymbol{\theta}(l_{\lambda'}), \lambda') \\ &\overset{(c)}{\geq} L(\boldsymbol{\theta}(l_\lambda), \lambda) - (L(\boldsymbol{\theta}(l_{\lambda'}), \lambda) + \epsilon) \\ &\overset{(d)}{\geq} L(\boldsymbol{\theta}(l'_\lambda), \lambda) - (L(\boldsymbol{\theta}(l_{\lambda'}), \lambda) + \epsilon) \geq -\epsilon \end{aligned} \tag{36}$$

where inequality (c) holds because $L(\boldsymbol{\theta}(l_{\lambda'}), \lambda') \leq L(\boldsymbol{\theta}(l_{\lambda'}), \lambda) + \epsilon$ as $\|\lambda' - \lambda\| \leq \delta \leq \delta_{l_{\lambda'}}$ and inequality (d) holds because $\boldsymbol{\theta}(l_\lambda)$ is the optimal policy parameter over any other parameter under Lagrangian multiplier λ.

As a result, we can conclude that, as long as $\|\lambda' - \lambda\| \leq \delta = \min_{l\in\mathcal{L}} \delta_l$, $|g(\lambda) - g(\lambda')| \leq \epsilon$, i.e., $g(\lambda)$ is a continuous function.

Appendix B

Lemma 1: Suppose $\{X_n, n \geq 1\}$ *are independent non-negative random variables with* $E(X_n) = \mu_n$, $Var(X_n) = \sigma_n^2$. *Define for* $n \geq 1$, $S_n = \sum_{i=1}^{n} X_i$, *and suppose that* $\sum \mu_i = \infty$ *and* $\sigma_n^2 \leq M\mu_n$ *for some* $M > 0$ *and all* n. *Then sequence* $S_n/\mathbb{E}(S_n)$ *converges to 1 in probability.*

Proof: According to Chebychev's inequality, for any $\epsilon > 0$, we have

$$P\{|S_n/\mathbb{E}(S_n) - 1| \geq \epsilon\} \leq \frac{\mathrm{Var}(S_n/\mathbb{E}(S_n))}{\epsilon^2} = \frac{\mathrm{Var}(S_n)}{\mathbb{E}^2(S_n)\epsilon^2}. \tag{37}$$

For $S_n = \sum_{i=1}^{n} X_i$, we have

$$\begin{aligned} P\{|S_n/\mathbb{E}(S_n) - 1| \geq \epsilon\} &\leq \frac{\mathrm{Var}\left(\sum_{i=1}^{n} X_i\right)}{\mathbb{E}^2\left(\sum_{i=1}^{n} X_i\right)\epsilon^2} = \frac{\sum_{i}^{n} \sigma_i^2}{\left(\sum_{i}^{n} \mu_i\right)^2 \epsilon^2} \\ &\leq \frac{M\sum_{i}^{n} \mu_i}{\left(\sum_{i}^{n} \mu_i\right)^2 \epsilon^2} = \frac{M}{\sum_{i}^{n} \mu_i \epsilon^2}. \end{aligned} \tag{38}$$

Therefore, when $n \to +\infty$, we can obtain that

$$\lim_{n\to+\infty} P\{|S_n/\mathbb{E}(S_n) - 1| \geq \epsilon\} = 0. \tag{39}$$

□

Definition 1: A subgradient of convex function f *at* $\boldsymbol{x}$ *is any vector* $\boldsymbol{z}$ *that satisfies the inequality* $f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{z}^T(\boldsymbol{y} - \boldsymbol{x})$ *for all* $\boldsymbol{y} \in \mathbf{dom} f$, *where* $\mathbf{dom} f$ *is the domain of function* f *[47].*

Lemma 2: The subgradient of function $g(\lambda)$ *can be denoted as* $h(\boldsymbol{\theta}_\lambda) = \mathbb{E}[C|\pi(\boldsymbol{\theta}_\lambda)] - \bar{C}$, *where* $\boldsymbol{\theta}_\lambda$ *is the value of* $\boldsymbol{\theta}$ *when* $L(\boldsymbol{\theta}, \lambda)$ *gets the supremum given* λ, *i.e.,* $\boldsymbol{\theta}_\lambda = \arg\max_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \lambda)$.

Proof: For any $\lambda' > \mathbf{0}$, we have

$$\begin{aligned} & g(\lambda') \\ & = \max_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \lambda') = \max_{\boldsymbol{\theta}} \left[V_{\pi(\boldsymbol{\theta})} + \lambda' h(\boldsymbol{\theta})\right] \\ & \geq V_{\pi(\boldsymbol{\theta}_\lambda)} + \lambda' h(\boldsymbol{\theta}_\lambda) = V_{\pi(\boldsymbol{\theta}_\lambda)} + \lambda h(\boldsymbol{\theta}_\lambda) + (\lambda' - \lambda) h(\boldsymbol{\theta}_\lambda) \\ & = g(\lambda) + h(\boldsymbol{\theta}_\lambda)(\lambda' - \lambda). \end{aligned} \tag{40}$$

□

Assume that the subgradient of function $g(\lambda)$ is bounded, i.e., $\|h(\lambda)\|_2 \leq G$. We prove the convergence by showing that the distance between optimal λ^* for the minimal $g(\lambda)$ and λ_i converges to zero. In the i-th iteration of Algorithm 2, the counted subgradient function is defined as $\bar{h}_i = C_i - \bar{C}$, and the measurement error for the expectation of average cost is defined as $n_i = C_i - \mathbb{E}[C|\pi(\boldsymbol{\theta}_i)]$. Thus the true subgradient in the i-th iteration of Algorithm 2 is denoted as $h_i = \mathbb{E}\left[C|\pi(\theta_i)\right] - \bar{C} = \bar{h}_i - n_i$. Accordingly, we have

$$\begin{aligned}
&\|\lambda_{i+1} - \lambda^*\|_2^2 \\
&= \left\| \left[\lambda_i - \alpha_i \bar{h}_i\right]^+ - \lambda^* \right\|_2^2 \\
&\leq \left\|\lambda_i - \alpha_i \bar{h}_i - \lambda^*\right\|_2^2 = \|\lambda_i - \alpha_i(h_i + n_i) - \lambda^*\|_2^2 \\
&\leq \|\lambda_i - \lambda^*\|_2^2 - 2\alpha_i h_i(\lambda_i - \lambda^*) - 2\alpha_i n_i(\lambda_i - \lambda^*) \\
&\quad + \alpha_i^2 \|h_i + n_i\|_2^2 \\
&\leq \|\lambda_i - \lambda^*\|_2^2 - 2\alpha_i(g(\lambda_i) - g^*) - 2\alpha_i n_i(\lambda_i - \lambda^*) \\
&\quad + \alpha_i^2 \|h_i + n_i\|_2^2 .
\end{aligned} \tag{41}$$

Applying the inequality above recursively, we have

$$\begin{aligned}
&\|\lambda_{i+1} - \lambda^*\|_2^2 \leq \\
&\|\lambda_1 - \lambda^*\|_2^2 - 2\sum_{k=1}^{i} \alpha_k(g(\lambda_k) - g^*) + \sum_{k=1}^{i} \alpha_k^2 \|h_i + n_i\|_2^2 \\
&- \sum_{k=1}^{i} 2\alpha_k n_k(\lambda_k - \lambda^*).
\end{aligned} \tag{42}$$

Based on the fact that $\|\lambda_{i+1} - \lambda^*\|_2^2 \geq 0$ and $\|\lambda_1 - \lambda^*\|_2$ must be bounded by some large number which is denoted by M, we have

$$2\sum_{k=1}^{i} \alpha_k(g(\lambda_k) - g^*) \leq M^2 + \sum_{k=1}^{i} \alpha_k^2 \|h_i + n_i\|_2^2 + z_i \tag{43}$$

where $z_i = \sum_{k=1}^{i} 2\alpha_k n_k(\lambda_k - \lambda^*)$. Then denoting the best result of $g(\lambda)$ before the $i-1$ iterations as g_i^{best}, i.e., $g_i^{\text{best}} = \max_{k<i} g(\lambda_k)$, we have

$$\begin{aligned}
\sum_{k=1}^{i} \alpha_k(g(\lambda_k) - g^*) &\geq \left(\sum_{k=1}^{i} \alpha_k\right) \min_{k<i}(g(\lambda_k) - g^*) \\
&= \left(\sum_{k=1}^{i} \alpha_k\right) (g_i^{\text{best}} - g^*).
\end{aligned} \tag{44}$$

We can further obtain the inequality

$$\begin{aligned}
&g_i^{\text{best}} - g^* \\
&\leq \frac{M^2 + \sum_{k=1}^{i} \alpha_k^2 \|h_i + n_i\|_2^2 + z_i}{\sum_{k=1}^{i} \alpha_k} \\
&= \frac{M^2 + \sum_{k=1}^{i} \alpha_k^2 \|h_i + n_i\|_2^2}{\sum_{k=1}^{i} \alpha_k} + \frac{\sum_{k=1}^{i} 2\alpha_k(h_k - \bar{h}_k)(\lambda_k - \lambda^*)}{\sum_{k=1}^{i} \alpha_k}.
\end{aligned} \tag{45}$$

When we choose α_i as square summable but not summable, i.e., $\sum_i^{\infty} \alpha_i^2 < \infty$, and $\sum_i^{\infty} \alpha_i = \infty$, it is obvious that the first term $\frac{M^2 + \sum_{k=1}^{i} \alpha_k^2 \|h_i + n_i\|_2^2}{\sum_{k=1}^{i} \alpha_k}$ converges to 0 in probability.

The second term is denoted as $\frac{\sum_{k=1}^{i} 2\alpha_k(h_k - \bar{h}_k)(\lambda_k - \lambda^*)}{\sum_{k=1}^{i} \alpha_k} = \frac{\sum_{k=1}^{i} 2\alpha_k h_k(\lambda_k - \lambda^*)}{\sum_{k=1}^{i} \alpha_k} - \frac{\sum_{k=1}^{i} 2\alpha_k \bar{h}_k(\lambda_k - \lambda^*)}{\sum_{k=1}^{i} \alpha_k} = \sum_k^i a_k - \sum_k^i b_i$, where $\sum_k^i a_k = \mathbb{E}\left(\sum_k^i b_i\right)$. Therefore, based on Lemma 2, the term in (45) converges to 0 in probability.

References

[1] H. He, H. Shan, A. Huang, Q. Ye, and W. Zhuang, "Reinforcement learning-based computing and transmission scheduling for LTE-U-enabled IoT," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.

[2] Q. Ye and W. Zhuang, "Distributed and adaptive medium access control for Internet-of-Things-Enabled mobile networks," *IEEE Internet Things J.*, vol. 4, no. 2, pp. 446–460, Apr. 2017.

[3] K. Zhang, J. Ni, K. Yang, X. Liang, J. Ren, and X. S. Shen, "Security and privacy in smart city applications: Challenges and solutions," *IEEE Commun. Mag.*, vol. 55, no. 1, pp. 122–129, Jan. 2017.

[4] D. Puthal, S. P. Mohanty, V. P. Yanambaka, and E. Kougianos, "PoAh: A novel consensus algorithm for fast scalable private blockchain for large-scale IoT frameworks," 2020, *arXiv:2001.07297*. [Online]. Available: http://arxiv.org/abs/2001.07297

[5] Z. Qin, F. Y. Li, G. Y. Li, J. A. McCann, and Q. Ni, "Low-power wide-area networks for sustainable IoT," *IEEE Wireless Commun.*, vol. 26, no. 3, pp. 140–145, Jun. 2019.

[6] R. S. Sinha, Y. Wei, and S.-H. Hwang, "A survey on LPWA technology: LoRa and NB-IoT," *ICT Express*, vol. 3, no. 1, pp. 14–21, Mar. 2017.

[7] MulteFire Alliance, "Multefire release 1.1 technical overview white paper," MulteFire Alliance, Fremont, CA, USA, Tech. Rep., Dec. 2018.

[8] S. Tatesh, *eLTE Brings Yangshan Port Into Automation Era*. Shenzhen, China: Huawei Technologies, 2018.

[9] C. Long, Y. Cao, T. Jiang, and Q. Zhang, "Edge computing framework for cooperative video processing in multimedia IoT systems," *IEEE Trans. Multimedia*, vol. 20, no. 5, pp. 1126–1139, May 2018.

[10] J. Ren, H. Guo, C. Xu, and Y. Zhang, "Serving at the edge: A scalable IoT architecture based on transparent computing," *IEEE Netw.*, vol. 31, no. 5, pp. 96–105, Aug. 2017.

[11] X. Chen, Q. Shi, L. Yang, and J. Xu, "ThriftyEdge: Resource-efficient edge computing for intelligent IoT applications," *IEEE Netw.*, vol. 32, no. 1, pp. 61–65, Jan. 2018.

[12] G. Hattab and D. Cabric, "Distributed wideband sensing-based architecture for unlicensed massive IoT communications," 2018, *arXiv:1812.06238*. [Online]. Available: http://arxiv.org/abs/1812.06238

[13] J. Ho and M. Jo, "Offloading wireless energy harvesting for IoT devices on unlicensed bands," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3663–3675, Apr. 2019.

[14] H. Zhang, B. Di, K. Bian, and L. Song, "IoT-U: Cellular Internet-of-Things networks over unlicensed spectrum," *IEEE Trans. Wireless Commun.*, vol. 18, no. 5, pp. 2477–2492, May 2019.

[15] S. Baidya and M. Levorato, "Edge-assisted content and computation-driven dynamic network selection for real-time services in the urban IoT," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, May 2017, pp. 796–801.

[16] Y. Chen, N. Zhang, Y. Zhang, and X. Chen, "Dynamic computation offloading in edge computing for Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4242–4251, Jun. 2019.

[17] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen, "Energy efficient dynamic offloading in mobile edge computing for Internet of Things," *IEEE Trans. Cloud Comput.*, early access, Feb. 11, 2019, doi: 10.1109/TCC.2019.2898657.

[18] X. Lyu *et al.*, "Optimal schedule of mobile edge computing for Internet of Things using partial information," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2606–2615, Nov. 2017.

[19] X. Cheng *et al.*, "Space/Aerial-assisted computing offloading for IoT applications: A learning-based approach," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1117–1129, May 2019.

[20] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11158–11168, Nov. 2019.

[21] T. Quang Dinh, Q. Duy La, T. Q. S. Quek, and H. Shin, "Learning for computation offloading in mobile edge computing," *IEEE Trans. Commun.*, vol. 66, no. 12, pp. 6353–6367, Dec. 2018.

[22] H. Habibzadeh, Z. Qin, T. Soyata, and B. Kantarci, "Large-scale distributed Dedicated- and non-dedicated smart city sensing systems," *IEEE Sensors J.*, vol. 17, no. 23, pp. 7649–7658, Dec. 2017.

[23] A. M. Haubenwaller and K. Vandikas, "Computations on the edge in the Internet of Things," *Procedia Comput. Sci.*, vol. 52, pp. 29–34, Jan. 2015.

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

[25] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 4, pp. 23–32, Apr. 2013.

[26] H. He, H. Shan, A. Huang, L. X. Cai, and T. Q. S. Quek, "Proportional fairness-based resource allocation for LTE-U coexisting with Wi-Fi," *IEEE Access*, vol. 5, pp. 4720–4731, 2017.

[27] *Study on Licensed-Assisted Access to Unlicensed Spectrum*, Standard 3GPP TR 36.889 May 2015.

[28] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 974–983, Apr. 2015.

[29] M. Hirzallah, M. Krunz, and Y. Xiao, "Harmonious cross-technology coexistence with heterogeneous traffic in unlicensed bands: Analysis and approximations," *IEEE Trans. Cognit. Commun. Netw.*, vol. 5, no. 3, pp. 690–701, Sep. 2019.

[30] G. J. Sutton, R. P. Liu, and Y. J. Guo, "Delay and reliability of load-based listen-before-talk in LAA," *IEEE Access*, vol. 6, pp. 6171–6182, 2018.

[31] M. Hirzallah, W. Afifi, and M. Krunz, "Full-duplex-based rate/mode adaptation strategies for Wi-Fi/LTE-U coexistence: A POMDP approach," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 1, pp. 20–29, Jan. 2017.

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

[33] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.

[34] H. C. Tijms, *A 1st Course Stochastic Models*. Hoboken, NJ, USA: Wiley, 2003.

[35] J. N. Tsitsiklis and B. Van Roy, "On average versus discounted reward temporal-difference learning," *Mach. Learn.*, vol. 49, nos. 2–3, pp. 179–191, Nov. 2002.

[36] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[37] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2011.

[38] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[39] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. Conf. Artif. Intell. (AAAI)*, Mar. 2016, pp. 2094–2100.

[40] A. M. Makowski and A. Schwartz, "Optimal index policies for MDPs with a constraint," in *Proc. 30th IEEE Conf. Decis. Control*, Dec. 1991, pp. 471–476.

[41] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602

[42] F. Liu, E. Bala, E. Erkip, M. C. Beluri, and R. Yang, "Small-cell traffic balancing over licensed and unlicensed bands," *IEEE Trans. Veh. Technol.*, vol. 64, no. 12, pp. 5850–5865, Dec. 2015.

[43] *Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Layer Procedures (Release 14)*, Standard 3GPP TS 36.213 2019.

[44] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, Nov. 2016, pp. 265–283.

[45] N. Lu, B. Ji, and B. Li, "Age-based scheduling: Improving data freshness for wireless real-time traffic," in *Proc. 18th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, Jun. 2018, pp. 191–200.

[46] Y. Bengio, "Deep learning of representations for unsupervised and transfer learning," in *Proc. ICML Workshop Unsupervised Transf. Learn.*, Jul. 2011, pp. 17–37.

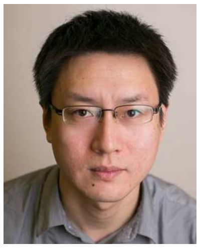
[47] S. Boyd and J. Park, "Subgradient methods," in *Notes for EE364b*. Stanford, CA, USA: Stanford Univ. 2013.

Hongli He (Member, IEEE) received the B.Sc. and Ph.D. degrees in information and communication engineering from Zhejiang University, Hangzhou, China, in 2014 and 2020, respectively. He is currently a Senior Engineer with Huawei Technologies Company Ltd. His research interests include vehicular ad-hoc networks, cellular networks over unlicensed spectrum, edge computing, and deep reinforcement learning in wireless communication.

Hangguan Shan (Member, IEEE) received the B.Sc. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 2004, and the Ph.D. degree in electrical engineering from Fudan University, Shanghai, China, in 2009. From 2009 to 2010, he was a Post-Doctoral Research Fellow with the University of Waterloo, Waterloo, ON, Canada. Since 2011, he has been with the College of Information Science and Electronic Engineering, Zhejiang University, where he is currently an Associate Professor. He is also with the Zhejiang Provincial Key Laboratory of Information Processing and Communication Networks, Hangzhou, SUTD-ZJU IDEA, Hangzhou, and the Zhejiang Laboratory, Hangzhou. His current research interests include cross-layer protocol design, resource allocation, and the quality-of-service provisioning in wireless networks. He has served as a Technical Program Committee Member of various international conferences, including the IEEE Global Communications Conference, the IEEE International Conference on Communications, the IEEE Wireless Communications and Networking Conference, and the IEEE Vehicular Technology Conference (VTC). He co-received the Best Industry Paper Award from the 2011 IEEE WCNC held in Quintana Roo, Mexico. He is currently an Editor of the IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING.

Aiping Huang (Senior Member, IEEE) graduated from the Nanjing Institute of Posts and Telecommunications, China, in 1977, and received the M.S. degree from the Nanjing Institute of Technology (Southeast University), China, in 1982, and the Licentiate of Technology degree from the Helsinki University of Technology (HUT), Finland, in 1997.

She was an Engineer with the Design and Research Institute of Posts and Telecommunications Ministry, China, from 1977 to 1980. From 1982 to 1994, she was with Zhejiang University (ZJU), China, as an Assistant Professor and then an Associate Professor. She was a Visiting Scholar and a Research Scientist with HUT (Aalto University) from 1994 to 1998. Since 1998, she has been a Full Professor with ZJU. She has published a book and more than 160 articles in refereed journals and conferences on communications and networks and signal processing. Her current research interests include heterogeneous networks, performance analysis, and cross-layer design, planning, and optimization of cellular mobile communication networks. She serves as the Vice Chair for the IEEE ComSoc Nanjing Chapter.



Qiang Ye (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Canada, in 2016. He was a Post-Doctoral Fellow and a Research Associate with the University of Waterloo from December 2016 to September 2019. He has been an Assistant Professor with the Department of Electrical and Computer Engineering and Technology, Minnesota State University, Mankato, USA, since September 2019. He has published more than 40 technical articles in different international journals/conferences. His current research interests include artificial intelligence and machine learning for future networking, 5G and beyond networks, software-defined networking and network function virtualization, network slicing, and the Internet of Things. He served as a Technical Program Committee (TPC) Member for several international conferences, including the IEEE GLOBECOM'20, VTC'17, VTC'20, and ICPADS'20. He serves as an Editor for the *International Journal of Distributed Sensor Networks* (SAGE Publishing) and *Wireless Networks* (SpringerNature) and as an Area Editor for the *Encyclopedia of Wireless Networks* (SpringerNature).

Weihua Zhuang (Fellow, IEEE) has been with the Department of Electrical and Computer Engineering, University of Waterloo, Canada, since 1993, where she is currently a Professor and a Tier I Canada Research Chair of wireless communication networks. She is a fellow of the Royal Society of Canada, the Canadian Academy of Engineering, and the Engineering Institute of Canada. She is an Elected Member of the Board of Governors and the VP Publications of the IEEE Vehicular Technology Society. She was a recipient of the 2017 Technical Recognition Award from the IEEE Communications Society Ad Hoc and Sensor Networks Technical Committee and several best paper awards from IEEE conferences. She was the Technical Program Chair/Co-Chair for the IEEE VTC Fall 2016 and Fall 2017, the Editor-in-Chief of the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY from 2007 to 2013, and an IEEE Communications Society Distinguished Lecturer from 2008 to 2011.