Joint Resource Allocation and Computation Offloading in Mobile Edge Computing for SDN based Wireless Networks

Nahida Kiran, Chunyu Pan, Sihua Wang, and Changchuan Yin

Abstract: **The rapid growth of the internet usage and the distributed computing resources of edge devices create a necessity to have a reasonable controller to ensure efficient utilization of distributed computing resources in mobile edge computing (MEC). We envision the future MEC services, where quality of experience (QoE) of the services is further enhanced by software defined networks (SDNs) capabilities to reduce the application-level response time without service disruptions. SDN, which is not proposed specifically for edge computing, can in fact serve as an enabler to lower the complexity barriers involved and let the real potential of edge computing be achieved. In this paper, we investigate the task offloading and resource allocation problem in wireless MEC aiming to minimize the delay while saving the battery power of user device simultaneously. However, it is challenging to obtain an optimal policy in such a dynamic task offloading system. Learning from experience plays a vital role in time variant dynamic systems where reinforcement learning (RL) takes a long term goal into consideration besides immediate reward, which is very important for a dynamic environment. A novel software defined edge cloudlet (SDEC) based RL optimization framework is proposed to tackle the task offloading and resource allocation in wireless MEC. Specifically, Q-learning and cooperative Q-learning based reinforcement learning schemes are proposed for the intractable problem. Simulation results show that the proposed scheme achieves 31.39% and 62.10% reduction on the sum delay compared to other benchmark methods such as traditional Q-learning with a random algorithm and Q-learning with epsilon greedy.**

Index Terms: **Mobile edge computing, resource allocation, software defined cellular networks, task offloading, wireless networks.**

I. INTRODUCTION

OVER the last decade, mobile data traffic of cellular networks has experienced exponential growth due to the explosive development of smart mobile devices (MDs). According to Cisco's report [1], mobile data traffic will grow at a compound annual growth rate (CAGR) of 46 percent between 2017 to 2022, reaching 77.5 exabytes per month by 2022, and will increase sevenfold between 2017 and 2022. This increasing demand offers new business opportunities but also poses an enormous challenge from a technical and economic perspective. Definitely, those various devices will produce exabyte level of data every day and the matter of fact is that the existing communication infrastructure will not be able to tackle the upcoming challenges. On a single mobile phone, there are numerous applications running from calling, web browsing to speech recognition and navigation [2]. To satisfy the growing demand of intelligent applications, the wireless communication technologies have experienced a massive development, such as virtual reality, augmented reality, advanced social networking and diverse intelligent terminal access. All these applications have emerged with the demand of higher computing efficiency, instantaneous communication and ubiquitous network connectivity. Hence, energy-efficient data processing is obviously vital for battery-empowered MDs.

Next generation devices are of small size with lower power, while the computation tasks are generally intensive and latency critical. Due to the limitation of physical size, computation capacity and low battery power, some computation applications can not be performed smoothly by the MDs [3]. So, enhancing the computing capability and prolonging the battery life of a mobile device is a key design challenge. mobile cloud computing (MCC) has been considered to be a potential solution and an effective technique to provide MDs with sufficient computation resources at (remote) centralized cloud servers [4]. Since the cloud servers have much higher computation and storage resources than MDs, migrating computation-intensive tasks to cloud servers can significantly reduce the burden of computing, storage, and computation energy on the MDs. In order to execute computation on cloud servers, the MDs are required to offload their tasks to remote clouds. However, cloud servers are usually logically and spatially far from MDs and the number of MDs increase dramatically, which makes the burden of the cloud heavier which leads to huge communications and processing latency and more energy consumption. This becomes a serious problem for the latency-sensitive applications.

To address these issues, mobile-edge computing (MEC) has emerged as a promising technique, which offers computation capability within the radio access network in contrast to conventional cloud computing systems that use remote public clouds [5]. The idea of bringing both communication and computational capacities close to users was firstly introduced as

Manuscript received July 19, 2019; approved for publication by Abbas Jamalipour, Division II Editor, July 29, 2019.

This work was supported in part by Beijing Natural Science Foundation and Municipal Education Committee Joint Funding Project under Grant KZ201911232046, in part by the National Natural Science Foundation of China under Grants 61671086, 61629101, and 61871041, and in part by the 111 Project under Grant B17007.

N. Kiran, C. Pan, S. Wang, and C. Yin are with the Beijing Laboratory of Advanced Information Network and the Beijing Key Laboratory of Network System Architecture and Convergence, Beijing University of Posts and Telecommunications, Beijing 100876, China, email: kiranbupt@yahoo.com, {kiran, cypan, sihuawang, ccyin}@bupt.edu.cn

N. Kiran is the corresponding author.

Digital Object Identifier: 10.1109/JCN.2019.000046

cloudlet in 2009 [6], where users can connect to nearby servers through a wireless local area network (WLAN). By offloading the computation intensive tasks from the MDs to the nearby MEC servers, the quality of computation experience, including the latency and device energy consumption, could be greatly improved. Several years later, the term mobile edge computing was used to describe the service executions at a mobile base station by Nokia Siemens Network and IBM [7]. Since then, MEC has attracted a lot of attention from academic areas [8]–[10].

The increasing demand of the mobile applications for high computing and storage capacities with free user mobility made MEC an insufficient solution for end user workload offloading. This is due to the distributed and limited resource computing capabilities of edge servers. Thus, how to manage limited computational resources of edge clouds and how to control these distributed computing resources is a challenging problem. MDs have very little information about the wireless networks to be accessed during computation tasks offloading, including computation load of edge clouds and traffic load of accessed network. So, how to conduct a task offloading and efficient resource allocation for particular task according to the residual battery capacity of mobile device and network status is challenging. Thus, a reasonable controller is needed to ensure the efficient utilization of computing resources and task placement appropriately [11], [12].

Software defined network (SDN) is a recent proposed technology for using the limited network resources optimally and enabling flexible network management by separating the control layer from the data layer [13]. The latest form of SDN has the integral capability to mitigate the barriers that prevent edge computing to reach its full potential. All task offloading management and data flow capabilities are accomplished by the central SDN controller that is transparent to the end-user. Since SDN is still a developing technology, applying SDN to edge or cloud computing has not been investigated in depth yet. Moreover, many surveys on software defined wireless networks (SDWN) [14], SDN [15], [16] and network function virtualization (NFV) [17] have been proposed as a complementary technologies for improving the operations of edge servers.

However, tasks processed only on a single edge server could not meet the requirement of low latency because of large number of tasks from different users. Thus, it is essential to carry out the distributed computing under a centralized network to balance energy, load, and to decrease latency. By integrating SDN and MEC, this paper proposes a novel software defined edge cloudlet (SDEC) framework for task offloading and resource allocation in MEC. The centralized logical controller senses the network status from a global perspective, collects distributed edge servers information (i.e., how much memory and CPU is available on the server side and how much the server is loaded), user equipment's (UE's) task information (i.e., task computation amount), and advises UE to compute task locally or offload part of it to edge or remote cloud for further processing. The motivation behind this proposed work is the urgent need to find a fast, centralized, and scalable cloud based architecture solution with a minimum cost.

The paper is organized as follows. Section II addresses the literature review. Section III describes the proposed system model. Section IV addresses the problem formulation. In Section V, we present details of low latency achieving schemes using machine learning. In Sections VI and VII, we show our simulation results and conclude the paper, respectively.

II. RELATED WORK

The centralized form of remote computing resources is not well suited with massive traffic originated from geographically distributed edge devices. Therefore, pushing the servers to the edge of the network is becoming an essential trend. Propagation of the edge devices creates a necessity for the applications to process at least some of the data at the edge rather than carrying them to the remote data centers for minimizing not only the energy consumption but also service delay. Recently, there have been many proposals for the operation and architectural design of the edge computing systems. The terms like edge computing [18], mobile cloud computing [19], fog computing [20], cloudlet [6], mobile edge computing [21], all these proposals define various practical implementations for the edge computing named as multi-access edge computing [22]. They all have mutual grounds but their target use cases are different.

Several research efforts have been dedicated for the research of computation offloading and resource allocation. Most of them focused on the process of computational tasks offloading from UEs to the MEC or cloud server [23], [24]. An adaptive offloading scheme is proposed in [25] by Xian *et al*. to improve the energy saving of MDs. In order to minimize the energy consumption of MDs in [26] Li *et al*. proposed a partitioning scheme that statistically divides a program into two different tasks (i.e., server task and client task). Ra *et al*. proposed a dynamic offloading strategy in [27] to minimize the completion time of applications by using a greedy algorithm. In [28] MEC systems have been proposed to address the computational capability issue and to enable MDs to utilize the powerful computing capability at the edge of the networks. A centralized computation offloading scheme is proposed in [29] which emphasizes on collecting global offloading information to meet highly delay requirements of MDs.

To the best of our knowledge, the first study of task offloading for mobile edge computing in a software defined ultra dense network (SD-UDN) was defined recently in [12] to offload task on edge cloud or process locally in order to minimize the task duration. A recent research survey provided use cases and future directions of the constituent technologies of the edge computing paradigm, namely cloudlets, fog computing and MEC. With this effort, a clear big picture and available technical approaches have been highlighted for edge computing domain to show how MEC benefits from SDN. This also emphasizes on the control and management challenges that exist in traditional platforms. In today's computing systems the management and administration operation costs are very high compared to other system operations. Thus, adopting the SDN approach to cut the management and administration cost has become very appealing.

There is a lot of effort both in academia and industry focusing on the MEC in particular due to its wide potential. Despite this fact, these studies remain incapable to discuss the common

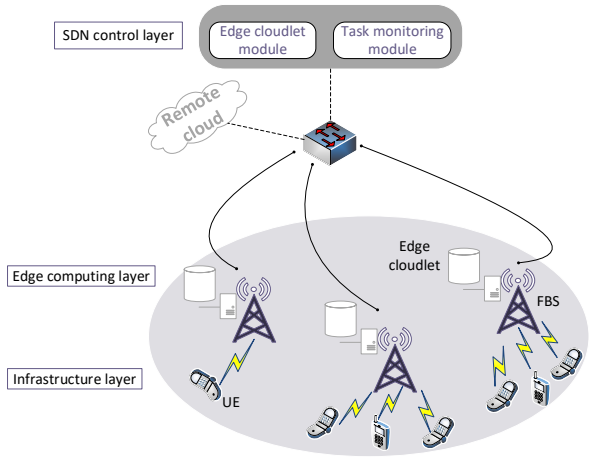

Fig. 1. System model in MEC for SDN based wireless communications.

edge computing proposals together by focusing on their actual requirements and differences among them, and depict the general view over edge computing concept. The prior works only focus either on centralized or distributed approaches, and have not taken advantage of both cloud and edge computing using SDN. The computational complexity and energy consumption increases significantly with the number of UEs, and the peak solution needs to be delivered from the distributed edge servers to UEs within the channel coherence time. Better performance can be grasped with the assistance of centralized SDN controller and distributed virtual cloudlets deployed in each femto base stations. Most of previous works focus on accessing the traditional cloud services over MDs and have not covered this area from a broad perspective and hence, managing the novel services and orchestrating the dynamic environment are not well addressed so far. There is not much literature focusing primarily on the MEC and SDN integration. Therefore, many ideas are studied, abstracted, integrated and extended inside our proposed model and the missing points that are not referenced so far for the integration of SDN and edge computing are extracted in this research for designating the future direction. Such integration guarantees provisioning an entirely software based framework for any system.

III. SYSTEM MODEL

As shown in Fig. 1, the proposed network framework consists of three layers: SDN control layer, edge computing layer and infrastructure layer. SDN control layer is composed of centralized SDN controller. Edge computing layer contains femto base stations (FBSs) and virtual edge cloudlets (ECs). Infrastructure layer includes all physical network entities having the tasks to compute like user equipments (UEs).

Important Entities in the System:

A. SDN Control Layer

SDN control layer is composed of centralized SDN controller. The SDN controller provides open unified interfaces to support task management, resource management and users access control by software programming. The centalized controller has a full view of the whole network topology and is aware of every flow in the network. This global view helps to store network traffic information and energy characteristics of the edge servers and dynamically control the data from a global perspective. Since SDN controllers have the global view of the SDEC, including all the load information, the energy saving and the load balancing can be optimized in a global way.

SDN Controller Modules:

Task Monitoring Module: responsible to collect all UEs task information (i.e., task computation amount) and is advised to compute task locally or offload part of them to edge or remote cloud for further processing.

Edge Cloudlet Module: responsible to collect distributed edge server's information (i.e., how much memory and CPU is available on the server side and how much the server is loaded).

B. Edge computing layer

Edge computing layer contains FBSs and virtual ECs with the capability of computing and storage. FBSs and ECs are collectively referred to as edge nodes (ENs). An EN get required data and service from the SDN controller. Then, the ENs give the traffic condition information of its associated users and surrounding ENs to the SDN controller, such as number of associated users, energy consumption of each node, and energy demand of each user. Moreover, the dynamic information of UE and working conditions which are stored by ENs in virtual ECs can be uploaded to the remote cloud for global information sharing.

ENs:

ENs are the intermediate nodes between UE and edge cloudlets. To reduce the burden of the controllers, we propose to deploy the wireless side of SDN in the concept of ENs where the ENs have part of control functions and all data forwarding functionality. The EN has a network-wide view of both wireless network and core network. Embedding ENs into the SDN is a promising way to reduce the burden of controllers, and integrate the wired and wireless sides of SDN seamlessly.

ECs:

It can be a virtualized server near to the FBSs available for users to carry out computing functionalities. It is proposed to overcome the problems caused by accessing the cloud data centers such as energy consumption, latency and cost. UE's task can be offloaded to the EC instead of a remote cloud server. It is a computational resource accessible by mobile users in their vicinity for making use of the services provided. ECs are controlled by SDN controller via OpenFlow (OF)-enabled switches by sending OF messages. With the help of cloud resources, ECs involve all the computing and storage functionalities, which enables new smart traffic, such as offload traffic and cache traffic.

C. Infrastructure Layer

Infrastructure layer includes all physical network entities having the tasks to compute like UEs.

Now let's consider a scenario with *N* deployed FBSs, $\mathcal{N} = \{1, 2, \cdots, N\}$ and each FBS is equipped with an EC, to serve *M* number of users, with $\mathcal{M} = \{1, 2, \cdots, M\}$.

An EC is assumed to have a finite computation capability and is located at the edge of the radio access network to provide computing services and execute UEs requests. The size of required computing task for each UE *i* is y_i. A remote cloud (RC) is also considered in the system to offload the overloaded requests of EC for further execution.

The whole network is under control of an SDN controller and the controller interacts with ENs and RC by sending OF messages to them to control the edge network centrally and globally. Since SDN controller could collect the global information of the whole system including load, energy consumption, processing speed and latency, it can formulate optimal energy saving and reduce latency strategies for the network. We assume that in a single time slot SDN controller will choose the UEs who want to offload their tasks for computation. We consider each UE generates a series of same task requests that can be offloaded to an EC through a wireless channel or locally processed by the UE. If the total request rate is not greater than a maximum acceptance rate of EC, then all the requests will be processed by the EC. Otherwise, overloaded task requests will be offloaded to RC for further execution.

We consider widely used task model [30] to describe task (q_i, y_i) in this paper with q_i be the required CPU cycles per bit to complete the task and y_i be the size of computation task. The UE selects the nearest FBS for task offloading. We assume that the size of the data that each UE *i* needs to compute is y_i and each UE *i* can transmit part of the data to its associated FBS for data processing. FBS transmits the corresponding task request information to SDN controller. SDN controller compares the task computation amount with the computing capability of EC and then gives the task offloading and resource allocation policy based on the delay and energy consumption of UE's task. The task is decided to compute locally by UE, offloaded to EC or further offloaded to RC for computation.

1. Local Computing : Consider the local computing with computing capability of l_i^{UE} at UE *i*, q_i is the required CPU cycles per bit and α_i is the ratio of task computed locally. Thus, the local computation time of task can be expressed as

$$t_i^{\text{UE}} = \alpha_i y_i \frac{q_i}{l_i^{\text{UE}}}, \tag{1}$$

where y_i is the size of computation task. The total requested tasks of UEs is represented as

$$T = \sum_{i=1}^{M} y_i. \tag{2}$$

Here *T* is the total task requests of all the users.

The consumed energy at UE *i* for local computation of tasks can be expressed as [31]

$$e_i^{\text{UE}} = (f_i q_i)\alpha_i y_i, \tag{3}$$

where f_i is the power coefficient of energy consumed per CPU cycle for local computation.

2. The Transmission Time and Energy Consumption for Task Offloaded to Edge Cloudlet: UE *i* will offload its task to EC. We can compute the uplink data rate $r_{i,j}$ as in [12] for offloading by considering the background interference and mutual interference caused by other UEs as

$$r_{i,j} = B \log_2 \left(1 + \frac{p_i \cdot h_{i,j}}{\sigma^2 + I_{i,j}}\right), \tag{4}$$

where p_i is the transmit power of UE *i* for uploading data and $h_{i,j}$ denotes the uplink channel power gain between UE *i* and the FBS *j*. For uplink transmission, we assume that all users occupy the same frequency channel of *B* bandwidth, σ^2 denotes the noise power level and $I_{i,j} = \sum_{k \in \mathcal{M}, \| \neq i} p_k h_{k,j}$ is the interference power from other users, where p_k is the transmit power of UE k and $h_{k,j}$ denotes the uplink channel power gain between UE k and the FBS *j*. Based on (4), transmission time of UE *i* for data offloading between UE *i* and FBS *j* is defined as

$$t_{i,j}^{\text{offl}} = (1 - \alpha_i) \frac{y_i}{r_{i,j}}. \tag{5}$$

The offloading energy consumption for transmitting UE's task from UE *i* to FBS *j* is

$$e_{i,j}^{\text{offl}} = \frac{p_i \cdot (1 - \alpha_i) y_i}{r_{i,j}}. \tag{6}$$

3. Execution Time in the Edge Cloudlet: After transmitting UE's requests to the edge cloudlet, the cloudlet will start executing UE's requests. As edge cloudlet is assumed to have finite computing capability (CPU cycles per second), which is denoted as l_{ij}^C allocated to user *i* by the *j*th FBS. The transmission time for UE *i* in the *j*th edge cloudlet can be defined as

$$t_{i,j}^{\text{C.trans}} = t_{i,j}^{\text{offl}} + t_i^{\text{C.exec}} = (1 - \alpha_i) \frac{y_i}{r_{i,j}} + \beta_i y_i \frac{q_i}{l_{ij}^C}, \tag{7}$$

where β_i is the ratio of task computed in the EC *j*.

4. Sending Overloaded Requests to the Remote Cloud: Further task requests must be overloaded to the RC because it has infinite capacity to execute all the UE's task. The transmission time to RC can be expressed as

$$t_j^{\text{CC.trans}} = (1 - \alpha_i - \beta_i) \frac{y_i}{r_j}, \tag{8}$$

where r_j is the data rate from FBS *j* to the RC. The transmission between FBSs to RCs normally occurs through a wired fronthaul links and wired connections definitely are generally faster than wireless connections and more reliable. The total execution time for UE *i* is expressed as

$$t_{exec}^{\text{total}} = t_i^{\text{UE}} + t_{i,j}^{\text{C.trans}} + t_j^{\text{CC.trans}}. \tag{9}$$

We can rewrite (9) as

$$t_{exec}^{\text{total}} = \alpha_i y_i \frac{q_i}{l_i^{\text{UE}}} + (1 - \alpha_i) \frac{y_i}{r_{i,j}} + \beta_i y_i \frac{q_i}{l_{ij}^C} + (1 - \alpha_i - \beta_i) \frac{y_i}{r_j}. \tag{10}$$

The total energy consumption for UE *i* is expressed as

$$e_{exec}^{\text{total}} = e_i^{\text{UE}} + e_{i,j}^{\text{offl}} = (f_i q_i)\, \alpha_i y_i + \frac{p_i \cdot (1 - \alpha_i) y_i}{r_{i,j}}. \tag{11}$$

IV. PROBLEM FORMULATION

Based on the above analytical results, we aim to minimize the total execution time subjects to the energy constraint. The problem can be mathematically written as

$$\min_{\alpha_i,\beta_i,l_{ij}^C} \sum_{i=1}^{M} \left[\alpha_i y_i \frac{q_i}{l_i^{\mathrm{UE}}} + (1-\alpha_i)\frac{y_i}{r_{i,j}} + \beta_i y_i \frac{q_i}{l_{ij}^C} + (1-\alpha_i-\beta_i)\frac{y_i}{r_j} \right], \quad (12)$$

subject to

$$C1: \sum_{i=1}^{M} l_{ij}^C \le l_j^{C.\max} \qquad \forall j = 1,2,\cdots,N. \quad (12a)$$

$$C2: e_i^{\mathrm{UE}} + e_i^{\mathrm{offl}} \le E_i^{\mathrm{UE.\max}} \qquad \forall i = 1,2,\cdots,M. \quad (12b)$$

$$C3: 0 \le \alpha_i \le 1 \qquad \forall i = 1,2,\cdots,M. \quad (12c)$$

$$C4: 0 \le \beta_i \le 1-\alpha_i \qquad \forall i = 1,2,\cdots,M. \quad (12d)$$

C1 represents that the total edge computing resources assigned to all tasks should be less than the edge cloudlet's maximum computing capability $l_j^{C.max}$. C2 indicates the maximum energy of UE $E_i^{UE.max}$. C3 shows that the ratio of task computed locally should be between 0 and 1. C4 makes sure that the ratio of task computed in the edge cloudlet should be between 0 and one minus ratio of task computed locally. Furthermore, in order to address the problem (12), we set α_i (ratio of task computed locally), β_i (ratio of task computed in the EC), and l_{ij}^C (computing capability of EC) as the optimization variables.

V. LOW LATENCY ACHIEVING SCHEMES USING MACHINE LEARNING

Our optimization problem can be solved by finding optimal value of computation resource allocation (i.e., optimal frequency cycles in the EC) and computing task ratio. As the problem is non convex according to the reference [32]. If the number of users increases the size of problem could increase very rapidly, so it's a NP-hard problem. In this section we focus on how machine learning helps in minimizing delays for the proposed problem in (12)–(12d) in MEC based wireless networks. MEC delays are caused by multiple factors, including node and link failures, unfavorable environment, delays caused by servers, and suboptimal scheduling. With multiple computation tasks sharing the same links, one delayed task can delay all the computation tasks scheduled after it. A failure in computing task either in the edge server due to heavy load or the remote cloud can lead to a complete halt of operations on a given route. This causes long delays with the estimated time for restoration contingent on various factors. The use of machine learning in wireless networks has already proven its effectiveness in performing predictive maintenance, which can be applied to MEC networks to eliminate delays. Machine learning can potentially eliminate network faults by constantly monitoring data points that can indicate any impending breakdowns. The massive amount of real-time data collected and analyzed through machine learning can not only improve the current network operations but also can assist in making long-term improvements like selecting the best edge servers and planning new optimal routes for offloading computation tasks. This can be achieved using reinforcement learning, a type of machine learning that can determine the optimal solution to problems by evaluating the results of previous actions.

The optimization problem becomes more complex due to large number of ECs participating in distributive computing. We firmly focus on transmission delay between the UEs and ECs because most of the applications are accessed from the edge servers. Hence, to extract the optimal solution for (12), we propose reinforcement learning (RL) algorithms such as Q-learning based approach and Cooperative Q-learning based approach. Then, we compare these algorithms with a random algorithm i.e., Q-learning based traditional approach.

A. *Reinforcement Learning*

In the following, we propose a RL method to solve this problem instead of using conventional optimization techniques because traditional learning methodologies such as training a model-based on historic training data and evaluating the resulting model against incoming data is not feasible as the environment is in a constant change. Conventional optimization based techniques are programmed to make particular decisions, for example there may be a scenario based on predefined rules. These rules are based on human experience of the frequently-occurring scenarios. However, as the number of scenarios increases significantly, it would demand massive investment to define rules to address all scenarios accurately, and either accuracy or efficiency is sacrificed. Furthermore, traditional approaches have a more rigorous mathematical approach while machine learning based RL algorithms are more data-intensive. The RL [33] problem consists of a single or multiple agents and an environment which is based on a chosen policy of taking actions to interact with the environment. The agent receives a feedback (reward) from the environment after each interaction, and updates its state. Any intelligent member of the problem can be the agent, for example in a cellular network it could be a small BS. The goal of RL is to maximize the cumulative received rewards during an infinite number of interactions. Such machine learning (ML) based RL system is truly a learning system if it is not programmed to perform a task, but is programmed to learn to perform the task.

B. *Proposed Q-learning Method*

In problem (12), our aim is to minimize the sum delay. Our system comprises a radio environment where UEs having computing tasks and a RL model using Q-learning [34] to perform computing capability control (i.e., cycles/s) to achieve sum delay. In RL method, three key elements are necessary to define specifically to our system model. These elements are state, action and reward. The radio environment can be in a normal state or undergo some actions that generate errors and cause the network delay. Such error generated actions can be tracked in a special register r_{error} to avoid delays.

State: The set of states $\mathcal{S}$ in our system consist of two components which can be defined as the sum delay $D_{\text{sum delay}}$ of the

entire system and the available computational capability $l_{ij}^{C.avail}$ of the MEC server which can be computed as

$$l_{ij}^{\text{C.avail}} = l_{ij}^{\text{C.max}} - \sum_{i=1}^{M} l_{ij}^{C} \tag{13}$$

where $\sum_{i=1}^{M} l_{ij}^{C}$ is the allocated computational resource (i.e., CPU cycles per sec).

Action: Set of actions $\mathcal{A}$ carried out by the agent and the actions in our system can be the resource allocation $l_{ij}^{C} = [l_{1j}^{C}, l_{2j}^{C}, \cdots, l_{nj}^{C}]$ and the computation ratio $\alpha = [\alpha_1, \alpha_2, \cdots, \alpha_n]$. The list of actions performed by the agent have a finite impact on the delay. After a series of actions the agent attempts to reach the sum delay $D_{\text{sum delay}}$.

Reward: After executing each possible action *a* in each step, the environment grants the agent a reward $R(s, a)$ in a certain state *s* at discrete time *t*. Each state-action pair will have a value $Q(s, a)$, i.e., the expected discounted reward when starting in state *s* and selecting an action *a*. This value can be regarded as a long term reward. In general, the reward function should be related to the objective function. Our objective is to meet the sum delay $D_{\text{sum delay}}$ and the goal of RL is to get the maximum rewards. After each action, we compute the delay increased (or decreased). If decreased we get the positive reward of +1, and if increased we get the negative reward of -1. So the value of reward should be negatively correlated to the size of objective function. In our system the immediate reward can be defined as

$$Immediate.Reward = \frac{1}{delay_{\text{min}}^{\text{curr.exec}}}, \tag{14}$$

where $delay_{\text{min}}^{\text{curr.exec}}$ is the minimum delay in executing a task in current state.

The agent computes for each step and stores $Q(s, a)$ in a Q-table. We build a $M \times N$ table $Q \in R^{M \times N}$ to drive $Q(s, a)$. The simplest form of one-step Q-learning approach can be given as

$$Q(s_t, a_t) = Q(s_t, a_t) + \eta[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)], \tag{15}$$

where $\eta,\ 0 < \eta < 1$, is the learning rate and $\gamma,\ 0 < \gamma < 1$, is the discount factor, which determines the importance of the predicted future rewards. The next action is *a* and the next state is s_{t+1}.

Q-learning can be considered as a function approximator in which the values of the approximator, Q, depend on the state and action. The dynamic programming equation for computing a function approximator Q (also known as Bellman equation) is as follows

$$Q(s_t, a_t) = max_a(E[R_t + \gamma Q(s_{t+1}, a)]), \tag{16}$$

where R_t is the immediate reward received at time *t* and *E* denotes the expectation operator.

Algorithm 1 shows the process of proposed Q-learning algorithm. In the context of a femtocell network, FBS deployed with an edge cloudlet acts as an agent in the Q-learning algorithm, which means each FBS runs Algorithm 1, separately. Problem (12) can be solved by finding optimal values of computation ratio and computation resource allocation l_{ij}^{C}. Our objective is to meet the sum delay $D_{\text{sum delay}}$ by allocating optimal computing resources to an edge cloudlet. The set of actions carried out by the agent is $\mathcal{A}$ and the set of states is $\mathcal{S}$. The environment grants the agent a reward $R(s, a)$ after the algorithm takes an action $a \in \mathcal{A}$ when it is in state $s \in \mathcal{S}$ at discrete time t. After each action, we compute the delay increased (or decreased). Each state-action pair will have a value $Q(s, a)$. The agent computes for each step and stores value $Q(s, a)$ in a Q-table. This value can be considered as a long term reward. We build a $M \times N$ table $Q \in R^{M \times N}$ to drive $Q(s, a)$.

Algorithm 1 Computing resource allocation using Q-Learning algorithm.

Initialization: Initialize $Q(s_t, a_t)$
for all episodes **do**
Initialize s_t
for all steps of episode **do**
Choose a_t from set of actions
Take action a_t , observe reward R_t , and next state s_{t+1}
Update the table entry $Q(s_t, a_t)$ as in eq (15)
$Q(s_t, a_t) = Q(s_t, a_t) + \eta[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
$s_t := s_{t+1}$
Until reach expected state $s_{terminal}$
end for
end for

According to Algorithm 1, if all actions are repeatedly sampled in all states, Q-leaning approach in (15) will be updated until the value of Q converges to the optimal value. The final value of Q may be suboptimal because the number of updates are limited. Q-learning itself is a greedy policy since on each iteration it finds the action which derives the maximum Q-value. Greedy policies have sometimes the disadvantage of being vulnerable to environmental changes, and they can be trapped in a limited search area which may cause the algorithm to converge slower. The effective solution for this is to act randomly with probability ϵ (exploring) and act greedily with probability $1 - \epsilon$ (exploiting). Algorithms that try to explore and exploit fairly are called ϵ-greedy and different values for ϵ provide a trade-off between exploration and exploitation. However, as compared to the greedy policy in [33], it is demonstrated that the ϵ-greedy policy has a faster convergence rate and closer final value to the optimal one in a limited number of iterations. In this paper, we used ϵ-greedy policy.

C. *Cooperative Q-Learning*

To further enhance the search speed in Q-learning method, a cooperative Q-learning technique is proposed. Cooperation and Q-value information sharing among agents in proposed Q-learning method can reduce search time for the optimum resource allocation. In [35] and [36], a cooperative assembled learning system has been developed as a new method with different training sets in neural network (NN). The robots cooperate with each other in order to learn to share their sensory data and play the role of scout for each other. In fact, by using a multi agent RL network (MARL), agents can communicate and share

their experiences with each other, and learn from one another. Episode sharing can be used to communicate the action, state, and reward triples between the reinforcement learners. The author proved that, sharing episodes with an expert agent could improve the group learning significantly. MARL network that consists of a large number of new agents, cooperation and information sharing among these agents can reduce search time for the delay minimization solution. The time complexity of a RL algorithm depends on the structure of states and the primary knowledge of the agents [37]. The search time can be excessive if priori knowledge is not available to an agent or if agent has to adapt environment changes. Hence, providing agents with priori knowledge and decreasing the effect of state space size on learning rate has been a subject of significant research. Transferring information between agents instead of expecting agents to discover all the necessary information can be the best solution to reduce the search time. In other words, the agents search different choices in parallel by sharing their information, which decreases the search time greatly.

In an SDN based network, the centralized controller gathers information regarding the network. The nature of this information for each FBS may be different and directly related to its active time in the network. In this paper, we propose a cooperative Q-learning approach where the Q-tables of FBSs that are in the same state, i.e., the FBSs that are located in the same vicinity are shared with one another. The proposed approach reduces the communication overhead among the FBSs by sharing the useful information among them. Sharing Q-values in MARL networks for resource allocation and management is still an open research problem.

Based on the proposed Algorithm 2 for the femtocell network, the agents learn in two modes: Individual learning mode and cooperative learning mode. At first, all of the agents follow the individual learning mode. FBSs execute the proposed RL algorithm independently by initializing the Q-values of a small subset of FBSs. Agent executes *n* learning trials. Each learning trial starts from a random state and ends when the agent reaches the goal. After a specified number of individual trials, all agents switch to cooperative learning mode. In this mode, the MARL network consists of experienced FBSs and new FBS. The new FBS takes its priori knowledge from the experienced FBSs and all FBSs execute the RL algorithm. Q-tables are shared after each iteration to form a new Q-table. In cooperative learning mode, some weights are assigned by each learning agent to the other agents Q-tables based on their expertise. Then, each agent takes the weighted average of the others Q-tables and uses the resulted table as its new Q-table which can be written as

$$Q_i^{\text{New}} \leftarrow \sum_{j=1}^{n} (W_{ij} \cdot Q_j^{\text{Old}}). \tag{17}$$

Assigning Weight Strategies:

a. Learning from all agents: In this method, the measurement of agent *i*'s reliance on the experience and knowledge of agent *j* is denoted as W_{ij}. It can be said that there are some useful skills to be learned from all agents. By using all agents' experience, weight can be assigned to agent *j*'s skills by a learner *i* using this formula

$$W_{ij} = \frac{e_j}{\sum_{k=1}^{n} e_k}, \tag{18}$$

where e_j is the expertness of agent *j*, e_k is the expertness amount of agent *k* and *n* is the number of agents. In this approach, agent *j*'s knowledge will effect all learners equally i.e., $W_{1j} = W_{2j} = \cdots = W_{nj}$. Also, after each cooperation step all Q-tables become homogeneous.

b. Learning with positive weights from all agents: If $e_{\min} = min\{e_k \mid k = 1, \cdots, n\}$, $and\ c > 0$ is a constant, then $e_k - e_{\min} + c > 0$. So the following weight assigning method can be defined as

$$W_{ij} = \frac{e_j - e_{\min} + c}{\sum_{k=1}^{n} (e_k - e_{\min} + c)} > 0. \tag{19}$$

The least expert agent's weight can be written as

$$W_{i,min} = \frac{c}{\sum_{k=1}^{n} (e_k - e_{\min} + c)}. \tag{20}$$

c. Learning from expert agents: In this method the learner may use only the Q-tables of the more expert agents to minimize the amount of communication required to exchange the Q-tables. Based on learner *i*'s expertness difference with other expert agents it assigns the weights by using following formula:

$$W_{ij} = \begin{cases} 1 - \mu_i, & \text{if } i = j \\ \mu_i \frac{e_j - e_i}{\sum_{k=1}^{n} (e_k - e_i)}, & \text{if } e_j > e_i \\ 0, & \text{otherwise,} \end{cases} \tag{21}$$

where μ_i is the impressibility factor and indicates how much each agent relies on other agents. *n* is the total number of the agents, e_i and e_j are the expertness value of agents *i* and *j*, respectively.

If other agents are less expert than the learner agent *i* then their partial weights are zero. Substituting (21) in line 24 of Algorithm 2 (weighted averaging formula) results in

$$Q_i^{\text{New}} \leftarrow 1 - \mu_i \cdot Q_i^{\text{Old}} + \mu_i \cdot \sum_{j \in expert(i)} \left(\frac{e_j - e_i}{\sum_{k=1}^{n} (e_k - e_i)} \cdot Q_j^{\text{Old}}\right), \tag{22}$$

where $expert(i) = \{j \mid e_j > e_i\}$ represents the set of agents that are more expert than less expert agent.

VI. SIMULATION RESULTS

In this section, we present the simulation results to estimate the performance of proposed scheme. We compare the proposed algorithm with a random scheme and investigate the impact of task computation amount and data size on the performance of task offloading. We assume data size y_i and task computation

Algorithm 2 Weighted strategy sharing using cooperative Q-Learning scheme for agent a_i.

Initialize:
While not end of learning **do**
begin
  If in individual learning mode then
  begin learning individually
    $x_i := Current\ State()$
    $a_i := Choose\ Action()$
    Do Action (a_i)
    $r_i := receive\ an\ immediate\ Reward()$
    $y_i := move\ to\ next\ State()$
    $V(y_i) := Max_{b \in actions} Q(y_i, b)$
    $Q_i^{New}(x_i, a_i) := (1 - \beta_i) Q_i^{Old}(x_i, a_i) + \beta_i (r_i + \gamma_i V(y_i))$
    $e_i := update\ expertness(r_i)$
  end
  else cooperative learning
  begin
    for $j := 1$ *to* n do
      $e_j := get\ expertness\ of\ agent\ a_j$
    $Q_i^{New} := 0$
    for $j := 1$ *to* n do
    begin
      $W_{ij} := compute\ waits(i, j, e_1 \cdots e_n)$
      $Q_j^{Old} := get\ Q(a_j)$
      $Q_i^{New} := Q_i^{New} + W_{ij} \cdot Q_j^{Old}$
    end
  end
end

amount of q_i is generated by a probability distribution. The total computation resource of mobile user is 15 GHz and the computing resource of edge cloudlet is 30 GHz. The femto network is simulated with 5 number of FBSs where each FBS supports 10 UEs as shown is Fig. 2. The UEs can connect within a 30m radius from its serving FBS. The task processing time when the computing task is processed by the UE is 1/15 s and 1/30 s when the computation is done by the EC. We also presume that the channel gain is determined as 127 + 30·log(d) for the link of UEs and FBSs where d is the distance between UE and FBS. The other radio network parameters are set as in Table 1. For implementing the proposed Q-learning algorithm, we use OpenAI Gym [38] environment with Python. Gym provides different game environments which we can plug into our code and test an agent. The library is responsible to take care of API for delivering all the information that our agent would require, like possible actions, current state, and score. We just need to focus on the algorithm for our agent. All statistical results are averaged over 3000 independent runs.

The proposed algorithm lets the agent use the received rewards to learn, over time to take the best action in a given state. We have the reward table in our environment, that the agent will learn from. It is looking for receiving a reward by taking an action in the current state, then updating a Q value to recall if that action was valuable. First, we will initialize the Q-table to a matrix of zeros and then we can create the training algorithm that

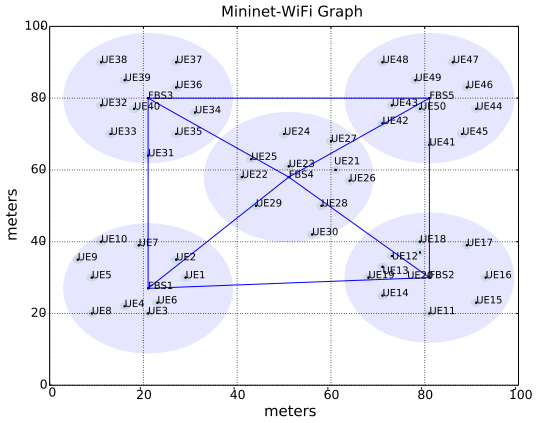

Fig. 2. Network topology.

Table 1. Simulation parameters.

Parameters	Values
Number of FBSs	5
Number of UEs per FBS	10
Channel power gain	127 + 30 · log(d)
Transmit power of UE, p_i	0.5 W
Transmission bandwidth of UE, B	10 MHz
Total battery capacity, $E_i^{\text{UE. max}}$	500 J
Gaussian channel noise, σ^2	$3 \cdot 10^{-15}$ W

will update this Q table as the agent explores the environment over thousands of episodes. We determine whether to pick a random action or to exploit the already computed Q values. We execute the prefered action in the environment to get the next state and the reward. Then, we calculate the maximum Q value for the actions corresponding to the next state, and with that, our Q value can easily be updated to the new Q value.

We implement proposed algorithms and set the machine learning parameters as in Table 2. The following values are used to perform Q-learning: Learning rate $\alpha = 0.5$, discount factor $\gamma = 0.9$. The maximum number of iterations is set to 3000 and the e-greedy algorithm is used for the first 60 percent of iterations with random $\epsilon = 0.1$. The optimal action-value function of Q-learning is learned after ξ episodes. We compare the proposed algorithm to the results of the approach with a random algorithm which selects the users randomly using traditional Q-learning algorithm and Q-learning with epsilon greedy. As the time complexity of a RL algorithm depends on state space size and the prior knowledge of the agents. The search time can be excessive if environment changes very rapidly and prior knowledge is not available to the agent. Best approach to deal with such problems is transferring information between the agents instead of expecting them discovering the mandatory information by themselves. By using the proposed cooperative Q-learning algorithm, the agents cooperate each other and share their experiences and Q-table values. This cooperation can reduce the search time for RL algorithms. In Fig. 3 we show how the total delay changes with the increasing capacity of edge cloudlet.

Table 2. Machine learning parameters.

Parameters	Values
Number of episodes, ξ	3000
Exploration rate, ϵ	0.1
Discount factor, γ	0.9
Learning rate, α	0.5
Exploration rate decay d	0.99
Minimum exploration rate, ϵ_{min}	0.020

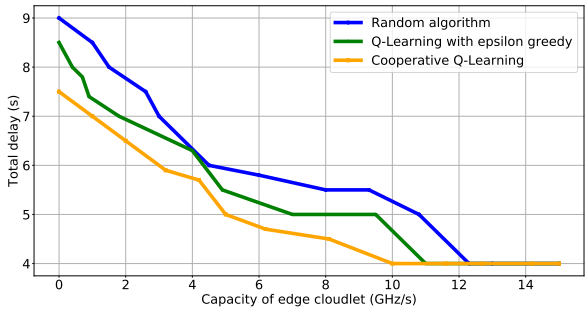

Fig. 3. Total delay vs. capacity of edge cloudlet.

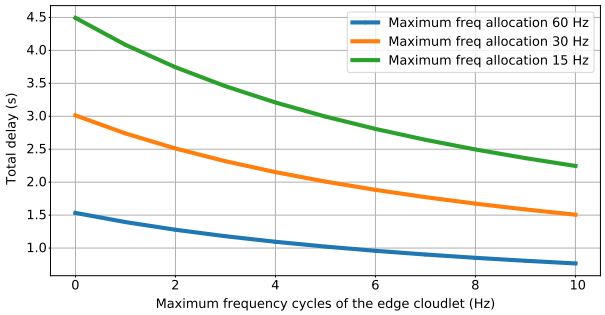

Fig. 4. Delay vs. maximum frequency cycles of the edge cloudlet.

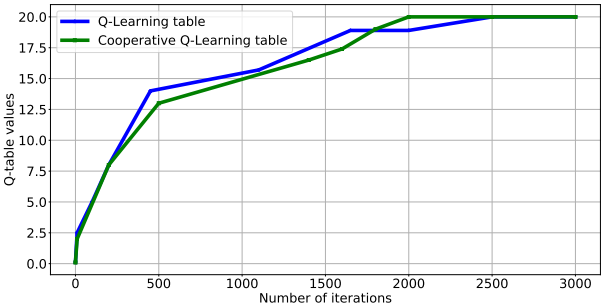

Fig. 5. Convergence of the proposed cooperative Q-learning algorithm.

We can observe that the delay in our proposed cooperative Q-learning policy decreases with the increasing computational capacity of edge cloudlet, because the execution time gets shorter if each UE is allocated more computational resource. This is due to the fact that after learning from the environment and sharing Q-table values with expert agents the system will learn to handle maximum number of users in a short time with minimum delay. Fig. 3 also demonstrates how cooperative Q-learning algorithm causes to share Q-tables of more expert agents to achieve the sum delay requirement and also minimizes the amount of communication required to exchange the Q-table values. At this stage, the proposed algorithm performs better than the Q-learning with epsilon greedy and random algorithm.

Fig. 4 represents the delay performance of the proposed algorithm under different frequency cycle requirements in the EC. Different cycles of frequency requirements play an important role to compute different tasks and have a great impact on latency performance in the system. In order to ensure the quality of service (QoS), the frequency cycles in the EC needs to be adjusted. The system assigns more frequency resources between the EC and UE, which increases the throughput of the network and minimizes the delay. For evaluating the influence of frequency cycle requirement to satisfy the QoS of UE, we set 3 different frequency cycles for distributed ECs under varied iteration numbers, where 60 Hz is considered as the optimal frequency. From Fig. 4 we can observe that due to the highest latency efficiency requirement, the algorithm needs to allocate more frequency resources. We allocate optimal frequency of 60 Hz which achieves the highest latency efficiency. The algorithm returns lowest latency efficiency due to assigning the least frequency of 15 Hz.

Fig. 5 shows the number of iterations needed till convergence for the proposed cooperative Q-learning approach. In this figure, we can see that, as time proceeds, the values of Q-tables increase until convergence to their final values. It also demonstrates that the proposed approach requires 2000 iterations to reach convergence while Q-learning approach needs 2500 iterations to reach convergence. It also shows that the proposed cooperative Q-learning algorithm goes through all iterations for each agent, and the agents share their Q-table values according to the algorithm. This indicate that the proposed approach is successful in satisfying the delay requirement of all UEs. From Fig. 5, we can also see that, tables for cooperative Q-learning and Q-learning may have different values as time increases. However, as time continues to elapse, the cooperative Q-learning table will converge earlier than the individual Q-table value. This is due to the fact that, at each iteration, the proposed algorithm selects an action based on the value of one Q-table and updates the actions Q-value in all the other Q-tables.

The results in Fig. 6 indicate the delay performance under different QoS. In this evaluation, the latency optimization performance is evaluated under optimal resource allocation to the appropriate edge cloudlet. Because when the user needs to ensure the higher QoS, the system allocates more frequency resources to the edge cloudlets, which increases the throughput of the network and minimizes the delay. We simulate with the maximum allocated frequency of 50 Hz under three different required QoS, 700 kbps, 300 kbps and 100 kbps. Fig. 6 shows that with higher QoS requirement, the overall delay of the system is reduced drastically. In contrast, Fig. 7 shows that due to random resource allocation to all edge cloudlets, the latency performance is deteriorating. This is attributed to inefficient resource allocation in the edge cloudlet, rather than selecting the

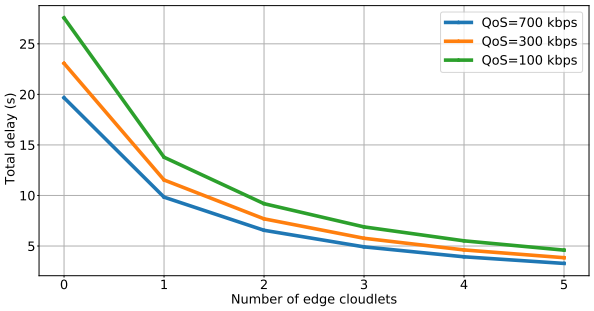

Fig. 6. Delay vs. number of edge cloudlets.

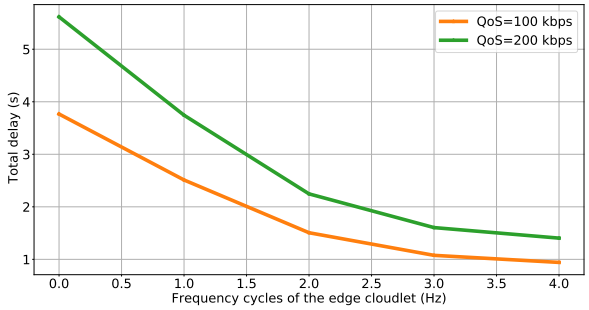

Fig. 7. Delay vs. frequency cycles of the edge cloudlets (Hz).

best EC with sufficient computing resources to provide services.

VII. CONCLUSION

In this paper we introduce a novel SDN based framework for computation offloading in MEC wireless networks. Then, we propose reinforcement learning based approaches (i.e., Q-learning and cooperative Q-learning) to solve the delay minimization problem, which takes both reward and punishment into consideration as a sign of being experienced; which is very important for a dynamic based MEC system. This approach basically shows the application of machine learning to address task offloading and resource allocation problem in MEC networks. In a multiple agent scenarios, the delay minimization in MEC is a non-convex problem that can be simplified to solve by implementing machine learning techniques instead of using conventional optimization methods. Our simulation results show that the proposed approach serves all users in a more appropriate way by sharing the learning experiences among all agents to compute UEs tasks. The proposed cooperative Q-learning scheme can achieve better performance than other baseline solutions under different system parameters to achieve delay requirements in SDN based edge network. Further, this RL based approach helps in selecting the best edge cloudlet with sufficient computing resources for providing services to users. By using this approach, we found that in most cases the agents learnt to cooperate, despite the fact that each agent was aiming to maximize its users preferences. Agents with different levels of expertness learn better in satisfying the delay requirement of UEs when they implement the weighted strategy sharing scheme using cooperative Q-learning.

REFERENCES

[1] S. Jose, "Cisco visual networking index: Global mobile data traffic forecast update," *Cisco, San Jose, CA, USA, White paper c11-741490*, Mar. 2017. [Online]. Available: http://www.cisco.com.

[2] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks Applicat.*, vol. 18, no. 1, pp. 129–140, 2013.

[3] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[4] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: Taxonomy and open challenges," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 369–392, 2014.

[5] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, and A. Neal, "Mobile-edge computing introductory technical white paper," *White paper, Mobile-edge Computing (MEC) industry initiative*, 2014.

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

[7] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[8] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[9] M. T. Beck, M. Werner, S. Feld, and S. Schimper, "Mobile edge computing: A taxonomy," in *Proc. AFIN*, 2014, pp. 48–55.

[10] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *arXiv preprint arXiv:1702.05309*, 2017.

[11] Y. Jararweh, A. Doulat, A. Darabseh, M. Alsmirat, M. Al-Ayyoub, and E. Benkhelifa, "Sdmec: Software defined system for mobile edge computing," in *Proc. IEEE IC2EW*, 2016, pp. 88–93.

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

[13] N. A. Jagadeesan and B. Krishnamachari, "Software-defined networking paradigms in wireless networks: A survey," *ACM Comput. Surveys*, vol. 47, no. 2, p. 27, 2015.

[14] M. Yang, Y. Li, D. Jin, L. Zeng, X. Wu, and A. V. Vasilakos, "Software-defined and virtualized future mobile and wireless networks: A survey," *Mobile Networks Applicat.*, vol. 20, no. 1, pp. 4–18, 2015.

[15] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 27–51, 2015.

[16] A. O. A. C. Baktir and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2359–2391, 2017.

[17] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Network Service Manage.*, vol. 13, no. 3, pp. 518–532, 2016.

[18] H. Pang and K.-L. Tan, "Authenticating query results in edge computing," in *Proc. IEEE ICDE*, 2004, pp. 560–571.

[19] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches," *Wireless Commun. Mobile Comput.*, vol. 13, no. 18, pp. 1587–1611, 2013.

[20] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. ACM MCC workshop*, 2012, pp. 13–16.

[21] B. Liang, *Mobile edge computing*, Cambridge University Press, 2017.

[22] N. Ansari and X. Sun, "Mobile edge computing empowers internet of things," *IEICE Trans. Commun.*, vol. 101, no. 3, pp. 604–619, 2018.

[23] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading decision and resource allocation for multi-user multi-task mobile cloud," in *Proc. IEEE ICC*, 2016, pp. 1–6.

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

[25] C. Xian, Y.-H. Lu, and Z. Li, "Adaptive computation offloading for energy conservation on battery-powered systems," in *Proc. IEEE ICPADS*, 2007, pp. 1–8.

[26] Z. Li, C. Wang, and R. Xu, "Computation offloading to save energy on handheld devices: A partition scheme," in *Proc. ACM CASES*, 2001, pp. 238–246.

[27] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," in *Proc. ACM MobiSys,* 2011, pp. 43–56.

[28] M. ETSI, "Mobile edge computing-introductory technical white paper," *White paper*, 2014.
[29] H. Zhang, Y. Qiu, X. Chu, K. Long, and V. C. Leung, "Fog radio access networks: Mobility management, interference mitigation, and resource optimization," *IEEE Wireless Commun.*, vol. 24, no. 6, pp. 120–127, 2017.
[30] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, 2016.
[31] J. Guo, H. Zhang, L. Yang, H. Ji, and X. Li, "Decentralized computation offloading in mobile edge computing empowered small-cell networks," in *IEEE Globecom Workshops*, 2017, pp. 1–6.
[32] Y. Pochet and L. A. Wolsey, *Production Planning by Mixed Integer Programming*, Springer Science & Business Media, 2006.
[33] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, vol. 2, no. 4, Cambridge: MIT Press, 1998.
[34] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
[35] T. Thorpe, "Multi-agent reinforcement learning: Independent vs. cooperative agents," Ph.D. dissertation, Master's thesis, Department of Computer Science, Colorado State University, 1997.
[36] Y. Liu and X. Yao, "A cooperative ensemble learning system," in *Proc. IEEE IJCNN*, 1998, pp. 2202–2207.
[37] S. D. Whitehead, "A complexity analysis of cooperative mechanisms in reinforcement learning," in *AAAI*, 1991, pp. 607–613.
[38] G. Brockman *et al.*, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

Nahida Kiran received her B.S. degree in Telecommunication & Networking from COMSATs Institute of Information Technology, Pakistan in March 2013. She acquired her Master's degree in Electronics and Communication Engineering from the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, P. R. China in 2016. She is currently pursuing her Ph.D. degree in Information and Communication Engineering from Beijing University of Posts and Telecommunications, P. R. China. Her research focus includes mobile edge computing, software defined networking, resource allocation and mobility management in cellular networks.

Chunyu Pan received the Ph.D. degree with the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing, P. R. China. She has been a Lecturer with School of Information and Communication Engineering, Beijing Information Science and Technology University, Beijing, P. R. China, since July 2019. Her research interests include software-defined cellular networks, heterogeneous wireless networks, resource allocation and mobility management in cellular networks.

Sihua Wang is currently pursuing the Ph.D. degree with the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing, P. R. China. His research interests include mobile edge computing, heterogeneous wireless networks, resource allocation and machine learning in cellular networks.

Changchuan Yin (M'98-SM'15) received the Ph.D. degree in Telecommunication Engineering from Beijing University of Posts and Telecommunications, Beijing, P. R. China, in 1998. In 2004, he held a visiting position with the Faculty of Science, the University of Sydney, Sydney, NSW, Australia. From 2007 to 2008, he held a visiting position with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA. He is currently a professor with the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications. His research interests include wireless networks and statistical signal processing. He was the co-recipient of the IEEE International Conference on Wireless Communications and Signal Processing Best Paper Award in 2009. He has served as a Technical Program Committee member for many leading IEEE conferences (e.g., ICC, Globecom, WCNC, VTC, etc.).