Q-Learning Algorithm for Joint Computation Offloading and Resource Allocation in Edge Cloud

Boutheina Dab
UPMC Sorbonne University
CNRS, LIP6
F-75005, Paris, France
boutheina.dab@lip6.fr

Nadjib Aitsaadi
University Paris-Est
LIGM-CNRS UMR 8049
LiSSi EA 3956, ESIEE Paris
F-93160, Noisy-le-Grand, France
nadjib.aitsaadi@esiee.fr

Rami Langar
University Paris-Est
LIGM-CNRS UMR 8049
F-77420, Champs-sur-Marne, France
rami.langar@u-pem.fr

***Abstract*—The advent of 5G technology along with the high proliferation of mobile devices entail an explosion of mobile traffic. Due to their resource-limitation constraint, mobile devices resort to connect to Cloud servers so as to offload computational tasks and improve, hence, resource usage. Unfortunately, the conventional Mobile Cloud Computing (MCC) solution involves high transmission latency. Recently Mobile Edge Computing (MEC) is envisioned as a promising technique for enhancing the computation capacities of mobiles and reducing latency. The key insight of MEC is to push mobile computing and storage to the network edge (i.e., base stations and access points). The main challenge of MEC solution is to find an efficient assignment of tasks with local or edge devices, while minimizing energy consumption and latency. In this paper, we propose a new joint task assignment and resource allocation approach in a multi-user WiFi-based MEC architecture. The main novelty of our work is that optimal offloading decision is jointly performed with the radio resource allocation. The objective of our scheme is to minimize the energy consumption on the mobile terminal side under the application latency constraint. To do so, we first formulate the problem as a new online Reinforcement Learning problem while considering both delay and device computation constraints. Then, we propose a new strategy based on a *Q-Learning* algorithm, named QL-Joint Task Assignment and Resource Allocation (`QL-JTAR`) to solve it. Based on extensive simulations conducted in NS3 simulator and using real input traces, we show that our approach outperforms the related prominent strategies in terms of energy consumption and delay, while ensuring near-optimal solution.**

Keywords: Mobile Edge Computing, offloading, reinforcement learning, optimization, resource allocation, IEEE 802.11ac.

I. Introduction

With the impressive increase of the Smart Mobile Devices (SMDs) popularity, a myriad of new mobile applications are emerging, such as face recognition, augmented reality and video streaming. Particularly, experts assure that the impressive proliferation of SMDs along with the advent of 5G network, will lead to the explosion of mobile traffic demand. In fact, the recent forecast of CISCO [1] expects that the number of mobile devices worldwide will reach 11.5 billion by 2019.

However, due to their limited processing power, mobile devices struggle to resist to such traffic explosion. Indeed, efficiency of mobile devices is still limited by the low evolution of battery capacity. For instance, battery capacity of iPhone has increased by only 29% since its initial release[1].

Faced with such exponential traffic growth, a promising solution proposes to take advantage of the remote Cloud resources for processing and computing of heavy applications. In this regard, during the last few years, research efforts have been devoted to Mobile Cloud Computing (MCC) solutions [2]. Nevertheless, a key limitation of MCC is the latency produced during data propagation. Indeed, servers are located far from SMDs, which entails high transmission delay.

To get rid of such limitation, a recent approach, named Mobile Edge Computation (MEC), has been proposed allowing mobile devices to use the powerful computing capability at the network edge. The main insight behind MEC approach is to bring both communication and computational capacities in close proximity to users. Actually, this concept was firstly introduced as Cloudlet [3], a nearby server to which users are connected through wireless links. Latter on, several edge cloud infrastructures have been propounded in research community [2], and industry like ETSI Mobile edge Computing [4].

To efficiently perform computation, code partition is required in order to decide which tasks should be run locally and which parts should be computed in the MEC server. Code partition decision depends on some parameters such as: i) CPU needed for each task, ii) the size of the execution output, iii) delay constraints, etc. Several MEC mechanisms have been propounded in literature. Particularly, *offloading* is one of the most relevant techniques that is orchestrated by SMDs. For instance, some offloading systems proposed in literature like MAUI [5] aim at overcoming resource limitations of mobile devices and reducing response delay.

Motivated by the benefits of mmWave wireless communications in MEC system [3], we envision, in this paper, a WiFi-based MEC architecture based on: i) ETSI MEC framework [4], and ii) IEEE 802.11ac standard [6]. In our architecture, multiple MEC servers are deployed. Then, we address the issue of joint task assignment and resource allocation in the WiFi-based MEC architecture. The main goal is to jointly optimize the task assignment and wireless resource

978-3-903176-15-7 © 2019 IFIP

[1] https://www.theverge.com/circuitbreaker/2017/6/28/15885636/iphone-10th-anniversary-hardware-specifications-comparison-apple-ios

allocation. For each incoming mobile application, the objective is to minimize the total energy on the mobile terminal side, while considering the following constraints: i) latency and ii) dependencies between tasks. To do so, we formulate the problem as a Reinforcement Learning approach where the state space corresponds to the set of tasks along with dependency links, while the actions represent the computation assignment decision. We propose a *Q-Learning*-based Task Assignment and Ressource Allocation approach, named `QL-JTAR`. The latter learns the optimal policy that minimizes the edge system's expected long-term cost (i.e., energy consumption). `QL-JTAR` proceeds as follows. **First**, initialization phase, our approach defines, for each state, a *Q-function* value that returns the cost and stands for the "Quality" of the action selected in that state. **Second**, the learning phase, the system is simulated. Particularly, in each visited state, some action is selected and the system allows transition to the next state. The *Q-function* is updated using the immediate cost generated in the transition. **Finally**, by the end of the learning phase, for each state, the action having the lowest *Q-function* value is declared to be the optimal. Accordingly, the optimal policy is determined.

Based on extensive network simulations conducted within NS3 simulator while considering the full protocol stack layers, we evaluate the performance of our proposal `QL-JTAR`. We consider the traces of real mobile applications, in the context of a football stadium. Obtained results show that our solution outperforms the related strategies in terms of energy consumption and latency, while ensuring near-optimal solution.

The outline of the paper is as follows. In section II, we give a summarized review of related work dealing with offloading strategies in MEC. Section III describes our WiFi-based MEC architecture. Next, we detail our system model in section IV. The formulation of joint offloading and resource allocation problem based on Reinforcement Learning will be presented in section V. In sections VI and VII, we, respectively, detail our proposal and present the performance evaluation results. Finally, we conclude this paper in section VIII.

II. Related work

Mobile Edge Computing related strategies can be classified into: i) **Single-User** MEC system, dealing with one user for a dedicated MEC server, and ii) **Multi-User** MEC system, where many mobile devices share one MEC server.

The single-User MEC system strategies presented in literature deal, in general, with two task models, namely: i) binary and ii) partial, task offloading models. First, the binary offloading consists in deciding whether a particular task has to be offloaded to edge or locally computed. Second, the partial task offloaded was addressed in [7] by decomposing heavy mobile applications into set of sub-tasks. The authors tackled the joint optimization of partial offloading and CPU-frequency scaling in order to minimize the energy consumption or latency. In [8], the authors formulate the task assignment as a latency minimization problem under resource utilization constraints. As in [9], they propose a new approximation algorithm to solve the problem while minimizing complexity.

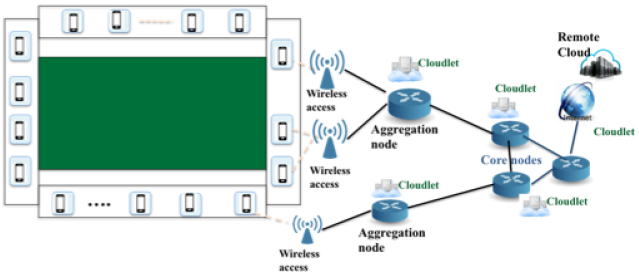


Fig. 1. WiFi-based MEC Architecture: Stadium Use case

For multi-user MEC solutions, the authors in [10] propose *MECO*, a Mobile-edge computing offloading scheme to offload intensive mobile computation for multi-user system. The authors propose to jointly optimize the radio resource allocation and computation. In [11], the authors propound a new joint radio resource and code partition in MEC approach based on relaxation technique. They investigate the call graph model to select the procedures to be computed remotely. However, the authors assume that the set of sub-tasks are independent and can be computed/offloaded simultaneously.

In this paper, we propose a new joint offloading and radio resource allocation approach to jointly: i) select the sub-tasks for remote execution and ii) allocate resources. Unlike [11], we consider applications with dependent tasks. Moreover, unlike [7] [8] [9], we consider multi-user multi-MEC server system, where each SMD can use many MEC servers.

III. WiFi-based MEC architecture

In this paper, we envision a MEC system architecture, based on ETSI framework [4], composed by MEC servers and a set of mobile users. Typically, our architecture is dedicated to the stadium and public venues use case. Indeed, ETSI argues in [12] that stadiums are considered good candidates for MEC because of the important venue services they host during sport events. Basically, our MEC system is deployed over a wireless access network having a backhauling wired infrastructure used to interconnect MEC servers. We hierarchically model the backhauling network based on a three-layers of nodes as illustrated in Fig. 1: i) Field layer containing the Access Points (APs), ii) Aggregation nodes connecting APs, and iii) Core backbone nodes connecting the MEC to the Internet. Note that APs could be based on either: i) WiFi only, ii) cellular only or iii) hybrid, wireless communication technology. In this paper, we assume that wireless network is based on WiFi technology to transmit data from SMDs to the server as argued in [5].

As depicted in Fig. 1, we deploy MEC over powerful servers, i.e., *cloudlets*, responsible of tasks computation. Cloudlets can be flexibly deployed on either field, aggregation or core level. In our MEC architecture, we assume a WiFi network based on IEEE 802.11ac standard [6].

IV. System Model

In this section, we present the task model of mobile applications. Then, we detail the computation model for the task

assignment in our hybrid WiFi-based MEC architecture.

A. Task Model

We assume that a mobile application is presented by a directed graph $\mathcal{G}$=($\mathcal{V}$, $\mathcal{E}$), where the nodes in $\mathcal{V}$ refer to the tasks, while the directed edges stand for data dependencies between tasks. Note that an edge $e = (i, j) \in \mathcal{E}$ means that the computation of task j depends on the output of task i. In other terms, the task j cannot start unless it gets results of task i. Each node i in $\mathcal{V}$ has a weight w_i that represents its computing workload, i.e., the number of CPU cycles required to compute the task i. Let α_i and β_i denote respectively the input and output data size of the task i. It is worth noting that the output of the task $(i-1)$ represent the input of the ith task, i.e., α_i=β_{i-1}. Each edge $e = (i, j) \in \mathcal{E}$ is characterized by a weight $N_{i,j}$ which specifies the size, in bits, of the program exchanged to switch the execution from task i to task j.

Let $\mathcal{N}$ denote the set of tasks constituting the ongoing mobile application $\mathcal{MA}$, and $\mathcal{M}$ the number of CPUs in the network. The root node represents the initial task of the application, while the final node is terminating the execution.

We assume that each task can be computed by only one CPU. Particularly, it can be either locally executed by the SMD (i.e., CPU 0) or remotely in one of the MEC servers, i.e., CPU $i, i \in \{1, .., \mathcal{M}\}$. Each CPU can compute a set of tasks.

In this paper, we consider, as in [9], a static scenario where all SMDs and the wireless network remain stationary during the offloading period. Note that this assumption holds for several applications, such as face recognition, natural language processing, etc. In fact, in such kind of mobile applications, usually the offloading could be achieved during a period estimated to be shorter than the timescales of SMDs mobility in the wireless networks. We sequentially perform the joint optimization of offloading decision and resource allocation for each new incoming application demand requested by an SMD.

We assume that each AP and MEC server k provides the mobile device with a fixed service rate, i.e., CPU frequency (speed), denoted by r_k expressed in cycles/sec.

B. Computation Model

We consider a joint task computation model between the SMDs and the MEC servers. Particularly, each task in graph $\mathcal{G}$ can either be locally executed or offloaded to the MEC server, based on the context of the task and the system.

Let $X \in \{0, ..., \mathcal{M}\}^{\mathcal{N}}$ denote the task assignment strategy, where the i^{th} component indicates the device on which task i is computed. Specifically, $x_i = k$ if task i is offloaded to CPU $k \in \{1, ..., \mathcal{M}\}$. Otherwise, $x_i = 0$. We denote by D_{ik} the computation delay of task i on CPU k.

1) Local Computation Energy Consumption Model: If task i of the mobile application is computed locally by the SMD, then the consumed energy consists mainly in the CPU and the screen. Note that since the latter firmly depends on the user behaviour, it is not considered [13]. Hence, the local computation time basically includes CPU operations:

$$D_{i0} = \frac{w_i}{r_0} \tag{IV.1}$$

where r_0 is the clock frequency (i.e., CPU speed) of the SMD.

The total local computation energy of the mobile device can be, thus, written as follows: :

$$\mathcal{E}_{Comp}^{Loc} = p_m^{\sigma} \cdot \sum_{i \in \mathcal{N}, x_i \in \{0\}} D_{i0} \tag{IV.2}$$

where σ is a constant ($\sigma \geq 2$).

2) Offloading Model: For mobile tasks that are offloaded to a MEC server, we assume that a software clone is already associated with each application in the cloudlet. In doing so, only the latest offloading data, such as generated data, and the computation output, need to be transmitted between the mobile devices and the MEC servers. During offloading process, three main steps are considered: i) Sending input data, ii) MEC computation, and iii) Receiving output data.

The two main objectives of our approach is to: i) minimize the mobile energy consumption and ii) satisfy the execution latency constraint. Hereafter, we will evaluate with close-formulas the above parameters.

a) Transmission energy consumption: The minimum transmission time required to send or receive $N_{i,j}$ bits of symbol duration T_b through the wireless channel is:

$$T_{i,j} = \frac{N_{i,j} \cdot T_b}{\mathcal{D}^r \cdot \mathcal{N}^S \cdot \mathcal{C}^r \cdot 8bits} \tag{IV.3}$$

where the numerator ($N_{i,j} \cdot T_b$) represents the number of bits per symbol, $\mathcal{D}^r$ represents the wireless data rate, $\mathcal{N}^S$ is the number of spatial streams, and $\mathcal{C}^r$ is the code rate.

Accordingly, if task i is locally executed while task j is offloaded to a MEC server, then the energy consumption associated to the data transmission is as follows:

$$\mathcal{E}_{i,j}^{Tx} = p_{i,j}^{tx} \cdot T_{i,j} \tag{IV.4}$$

On the other hand, if j is locally computed, then the energy needed by the SMD to decode the $N_{i,j}$ bits of the output program transmitted back by the AP is denoted by $\epsilon_{i,j}$. Note that the latter depends only on the size of the program but not on the mobile transmitted power. Let $y_{ik} \in \{0, 1\}$ be a binary variable indicating whether task i is computed on CPU k or not. The total wireless transmission energy of the mobile application on the SMD side is, hence, defined as follows:

$$\begin{aligned}\mathcal{E}^{\mathcal{TR}} = \textstyle\sum_{(i,j)\in\mathcal{E}} [&\mathcal{E}_{i,j}^{Tx} \cdot \textstyle\sum_{k=1}^{M} y_{jk} + \epsilon_{i,j} \cdot \textstyle\sum_{k=1}^{M} y_{ik} \\ &-(\mathcal{E}_{i,j}^{Tx} + \epsilon_{i,j}) \cdot \textstyle\sum_{k=1}^{M} y_{ik} \cdot \textstyle\sum_{k=1}^{M} y_{jk}]\end{aligned} \tag{IV.5}$$

The total energy consumption of a SMD is given by:

$$\mathcal{E}(X) = \mathcal{E}_{Comp}^{Loc} + \mathcal{E}^{TR} \tag{IV.6}$$

b) Total offloading Delay: The total delay includes both the: i) computation and ii) transmission (sending and receiving), delays. The time needed to compute the task $i \in \mathcal{V}$ locally (i.e., x_i=0) is given by the equation IV.1. Hence, the total local computation time is computed as follows:

$$\mathcal{T}^l = \sum_{i \in \mathcal{V}, x_i \in \{0\}} D_{i0} \tag{IV.7}$$

Note that when a component computation is delegated to a MEC server, the mobile device remains in idle state and its wireless network interface is turned off during execution. Let D_{ik}^{Comp} denote the completion time of the i^{th} task on the MEC server with CPU k. D_{ik}^{Comp} depends on both the: i) computation time and ii) waiting time in the queue. The latter incarnates the queueing time of the ongoing residual workload in device k. Formally, D_{ik}^{Comp} is given by:

$$D_{ik}^{Comp} = D_{ik} + T_{ik} \tag{IV.8}$$

where the computation time D_{ik} is defined by:

$$D_{ik} = \frac{w_i}{r_k} \tag{IV.9}$$

and T_{ik} represents the waiting delay defined as follows:

$$T_{ik} = \frac{w_k^{res}}{r_k} + \frac{\sum_{j \in V, j \neq i} w_j \cdot y_{jk}}{r_k} \tag{IV.10}$$

where r_k represents the fixed service rate (i.e., CPU frequency) provided by the MEC server k. The latter is much higher than the mobile CPU frequency. y_{jk} represents a binary variable indicating whether task j, a predecessor of task i, is computed by CPU k or not. The second term corresponds to the waiting delay in the queue of the device k. Then, the total completion time required to compute task i remotely (i.e., $x_i \in \{1, .., \mathcal{M}\}$) or locally (i.e., $x_i = 0$) is defined as follows:

$$\mathcal{T}^{Comp} = \sum_{i \in \mathcal{V}} D_{ix_i}^{Comp} \tag{IV.11}$$

Computation offloading requires the transmission of the input program from one site to the other, e.g., the SMD to the AP, and vice versa. The induced total transmission (sending/receiving) delay is hence defined as follows:

$$\begin{aligned} \mathcal{T}^{TR} = \sum_{(i,j) \in \mathcal{E}} [\mathcal{T}_{i,j} \cdot \sum_{k=1}^{M} y_{jk} + \gamma_{i,j} \cdot \sum_{k=1}^{M} y_{ik} \\ -(\mathcal{T}_{i,j} + \gamma_{i,j}) \cdot \sum_{k=1}^{M} y_{ik} \cdot \sum_{k=1}^{M} y_{jk}] \end{aligned} \tag{IV.12}$$

where $\gamma_{i,j}$ is the time required to get the data back from the AP. Note that this time does not depend on the transmitted power but only on the size of the program, $N_{i,j}$.

V. Problem Formulation

In this section, we formulate the joint offloading and resource allocation in MEC as an online **Reinforcement Learning** (*RL*) problem [14] [15], in order to minimize the system cost (i.e., energy consumption).

A. Reinforcement Learning Formulation

The MEC system state changes from one time slot to the other. The time slot is defined as the period during which a task is computed. Hence, the problem can be cast as Markovian Decision Problem (MDP). The latter can be solved by either: i) iteratively solving a linear system of Bellman equations [14], or ii) using the Bellman transformation in an iterative way to compute the optimal value function. In this regard, Reinforcement Learning has recently emerged as value-iteration approach that provides optimal or near-optimal solutions to large MDPs [14]. Basically, MDP is defined by four elements: i) the state space $\mathcal{S}$, ii) the action space $\mathcal{A}$, iii) the cost function, and iv) the policy π.

1) State Space: Let $\mathbb{C}_h$ denote the set of available wireless channels in the Wi-Fi-based MEC system. We define a state of the MEC system by the tuple $s = (i, e, ch, m)$, where $i \in \mathcal{V}$,$e \in \mathcal{E}, ch \in \mathbb{C}_h, m \in \{0, .., \mathcal{M}\}$. The state space is defined by the set of states for all the task graph, $\mathcal{S} = \{s = (i, e, ch, m)\}$.

2) Action Space: The actual actions taken for each task $i \in \mathcal{V}$ in a sate $s \in \mathcal{S}$ by the edge system are $a_{i,s} \in \{0, .., \mathcal{M}\}$, where $a_{i,s} \geq 1$ refers to an offloading action to a MEC server, while $a_{i,s} = 0$ corresponds to a local computation action. Accordingly, the action space is defined by: $\mathcal{A} = \{a_{i,s} \in \{0, .., \mathcal{M}\}, i \in \{0, ..., |\mathcal{V}|\}, s \in \{0, ..., |\mathcal{S}|\}$.

3) Cost function: We define the immediate cost of taking action $a_{i,s}$ for task $i \in \mathcal{V}$ on state $s \in \mathcal{S}$ as follows:

$$\psi(s, a_{i,s}) = \begin{cases} C_o(s, a_{i,s}) & \text{if } a_{i,s} \geq 1 \\ C_l(s, a_{i,s}) & \text{if } a_{i,s} = 0 \end{cases}$$

where $C_o(s, a_{i,s}) = \mathcal{E}_{s,a_{i,s}}^{Tx}$ represents the offloading cost, and $C_l(\mathcal{S}, a_{i,s}) = \mathcal{E}_{Comp}^{Loc}(i)$ is the local computation cost.

We define the expected long-term cost (named delayed cost) as the expected sum of the component immediate costs:

$$\psi^{\pi}(\mathcal{S}) = \sum_{i \in \mathcal{V}, s \in \mathcal{S}} \psi(s, a_{i,s}) \tag{V.13}$$

4) Policy: The policy is a mapping function $\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. It corresponds to the joint offloading and resource allocation policy in the MEC system. The policy maps each state to an action. The agent, represented by the MEC manager process, aims at optimizing the policy π to minimize the edge system's expected long-term cost. Thus, the optimal offloading policy, denoted by $\pi^{\star}$, should minimize the system cost given by:

$$\pi^{\star} = argmin_{\pi} \sum_{i \in \mathcal{V}, s \in \mathcal{S}} \psi(\mathcal{S}, a_{i,s}) \tag{V.14}$$

B. Joint Task Assignment and Resource Allocation Optimization Problem

The main objective is to find the task assignment strategy X that minimizes the total energy consumption on the mobile side, while considering latency constraint. Formally, the objective function of our optimization problem is given by:

$$\begin{aligned} &\text{minimize} \sum_{(i,j) \in \mathcal{E}} [\mathcal{E}_{i,j}^{Tx} \cdot \sum_{k=1}^{M} y_{jk} + \epsilon_{i,j} \cdot \sum_{k=1}^{M} y_{ik} \\ &-(\mathcal{E}_{i,j}^{Tx} + \epsilon_{i,j}) \cdot \sum_{k=1}^{M} y_{ik} \sum_{k=1}^{M} y_{jk}] \\ &+ \sum_{i \in \mathcal{V}, x_i \in \{0\}} p_m^{\sigma} \cdot D_{i0} \end{aligned} \tag{V.15}$$

To ensure the latency constraint, the total computation and transmission delay should be less than the maximum latency dictated by the application. Formally, this is ensured by the following constraint:

$$\mathcal{T}^{Comp} + \mathcal{T}^{TR} \leq \mathcal{L} \tag{V.16}$$

where $\mathcal{L}$ denotes the maximum latency of the application, assumed to be equal to the local execution delay.

Note that the task assignment vector X indicates for each task i either it should be computed locally (i.e., $x_i = 0$) or

remotely (i.e., $x_i = k, k \in \{1, .., \mathcal{M}\}$). Hence, x_i is an integer variable that should satisfy the following constraint:

$$x_i \in \{0, .., \mathcal{M}\}, \forall i \in \mathcal{V} \tag{V.17}$$

Moreover, each task i of the mobile application, $\mathcal{MA}$, should be assigned to only one CPU (i.e., the local SMD or one MEC server). Hence, single CPU computation is guaranteed with the following constraints:

$$\begin{array}{l} \sum_{k=0}^{M} y_{ik} = 1, \forall i \in \mathcal{V} \\ y_{ik} \in \{0,1\}, \forall k \in \{0, ..., \mathcal{M}\},\ \forall i \in \mathcal{V} \end{array} \tag{V.18}$$

In addition, to ensure the computation dependency between tasks, we denote by $\mathcal{L}_i$ the list of predecessor nodes of task i on the call graph $\mathcal{G}$. We define a binary variable $z_i \in \{0,1\}$ indicating whether the task is assigned or not yet. To guarantee that each task i is assigned only after the allocation of all its predecessor tasks, we add the following constraint as follows:

$$\begin{array}{l} z_i = \sum_{k=0}^{M} y_{ik},\ \forall i \in \mathcal{V} \\ z_i \in \{0,1\}, \forall i \in \mathcal{V} \\ z_i \leq z_j, \forall j \in \mathcal{L}_i,\ \forall i \in \mathcal{V} \end{array} \tag{V.19}$$

Formally, the joint task assignment and resource allocation (JTAR) optimization problem is formulated as follows:

$$\begin{array}{lll} Minimize & \sum_{(i,j)\in\mathcal{E}} [\mathcal{E}_{i,j}^{Tx} \cdot \sum_{k=1}^{M} y_{jk} + \epsilon_{i,j} \cdot \sum_{k=1}^{M} y_{ik} & \\ & -(\mathcal{E}_{i,j}^{Tx} + \epsilon_{i,j}) \cdot \sum_{k=1}^{M} y_{ik} \sum_{k=1}^{M} y_{jk}] & \\ & + \sum_{i\in\mathcal{V}} p_m^{\sigma} \cdot D_{i0} \cdot y_{i0} & \\ subject\ to: & \mathcal{T}^{Comp} + \mathcal{T}^{TR} \leq \mathcal{L} & \\ & \sum_{k=0}^{M} y_{ik} = 1, \forall i \in \mathcal{V} & \\ & y_{ik} \in \{0,1\}, \forall k \in \{0, ..., \mathcal{M}\},\ \forall i \in \mathcal{V} & \\ & z_i = \sum_{k=0}^{M} y_{ik},\ \forall i \in \mathcal{V} & \\ & z_i \in \{0,1\}, \forall i \in \mathcal{V} & \\ & z_i \leq z_j, \forall j \in \mathcal{L}_i,\ \forall i \in \mathcal{V} & \end{array}$$

Note that JTAR problem is an Integer Programming problem since the variables y_{ik} and z_i are integer.

VI. Proposal: QL-JTAR

To solve the aforementioned *JTAR* problem based on $\mathcal{RL}$ technique, we propose a new approach using the *Q-Learning* algorithm [14], named QL-JTAR. Particularly, *Q-Learning* is a value-based $\mathcal{RL}$ algorithm. Its goal is to learn, in an online way, a policy which tells the agent what action to take under which condition. Note that the online learning process does not require knowing the transition probabilities ahead of time. The main strength of $\mathcal{RL}$ lies in its ability to obtain near-optimal solutions of complex large-scale MDPs. Specifically, given a policy π, the objective is to learn the *state-action value* function, named *Q-function*, $\mathcal{Q}(s,a)$ of each state-action pair. In order to compute the actual values of each state, we make use of the model-free learning [14]. The value of *Q-function* is the expected sum of the discounted costs for an agent starting at state s, taking action a, and then following policy π. Our approach records for each trial the *Q-function* value. After a finite number of iterations, these values will converge.

The optimal *Q-function* is defined as the best *Q-function* that can be learned by any possible policy. A policy that selects for every state the action with the smallest (optimal) Q-value, is optimal. Note that a policy that minimizes a *Q-function* in this way is said to be **greedy** in that *Q-function*. Therefore, to find an optimal policy, we first start by finding $\mathcal{Q}^{\star}$, and then we compute a greedy policy in $\mathcal{Q}^{\star}$.

It is worth noting that our QL-JTAR algorithm requires the updating of *Q-function* whenever the system visits a new state. For each state-action pair, the so-called *Q-function* is stored. Our approach proceeds as follows. First, **initialization phase**, the *Q-function* values are initialized to suitable numbers in the beginning, as explained in section VI.A. Second, **learning phase**, the system is simulated using the *Q-Learning* algorithm. Particularly, in each visited state s, some action a is selected and the system allows transition to the next state. The immediate cost $\mathcal{C}(s,a)$ that is generated in the transition is recorded as the feedback. The latter is used to update the *Q-function* for the action selected in the previous state. If the feedback is **good**, the *Q-function*, $\mathcal{Q}(s,a)$, of that particular action and the state in which the action was selected is decreased using the advanced version Relaxed-SMART algorithm [15], first introduced in [14]. Otherwise, i.e., the feedback is **poor**, the $\mathcal{Q}(s,a)$ is punished by increasing its value. Afterwards, the same reward-punishment policy is carried out in the next state. This process is iteratively repeated for a large number of transitions. By the end of the learning phase, for each state s the action $a^{\star}$ having the lowest *Q-function* value is declared to be the optimal action for s. Accordingly, the optimal policy is determined.

Hereafter, we will detail the steps of QL-JTAR algorithm.

A. Initialization Phase

The *Q-function* are Initialized to some arbitrary values (e.g., 0). For the iteration count k, set to 1, let $\mathcal{C}^k$ denote the average cost in the k^{th} iteration of the algorithm. Set $\mathcal{C}^1$ to 0. Let i denote the first state. Let $\mathcal{A}(i)$ denote the set of actions allowed in state i (i.e., offloading or local computation). We denote by ρ^k and τ^k the two learning rates, that should be positive values typically less than 1. Note that the learning rates should also be a function of k. Our approach must ensure that τ^k converges to 0 faster than ρ^k. To satisfy these conditions, we set the step sizes to $\rho^k = \frac{\log(k)}{k}$ and $\tau^k = \frac{90}{(100+k)}$ as in [15]. Accordingly, as k tends to infinity, $\frac{\tau^k}{\rho^k}$ should tend to zero.

Then, we set the long-term cost and the long-term transition time to zero. Let ITERMAX denote the number of iterations for which the algorithm is run, initialized to a large number. Finally, as in [15], we set η, a scaling constant needed, to 0.99.

B. Learning Phase

The steps of the learning phase are as follows.

. **Step** 1 (***Q-function* Update**): QL-JTAR determines the action associated to the lowest *Q-function* value in state i. The determined action is called the **greedy** action. Then, for each iteration count k, the greedy action is selected with probability $(1-p(k))$. Commonly, $p(k)$ is given by:

Algorithm 1: QL−JTAR pseudo-algorithm

Inputs: $\mathcal{S}$, $\mathcal{A}$, $\mathcal{MA}$
Output: $\pi^\star$ /* optimal task assignment policy */
Initialization:
$\mathcal{Q}(s,a) \leftarrow 0, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$, $k \leftarrow 1$, $\mathcal{C}^1 \leftarrow 0$
$i \leftarrow First\_State$, $\mathcal{A}(i) \leftarrow$ set of actions allowed in state i
$\rho^k = \frac{\log(k)}{k}$, $\tau^k = \frac{90}{(100+k)}$
$total\_cost \leftarrow 0$, $total\_time \leftarrow 0$, $\eta \leftarrow 0.99$
repeat
  $\mathcal{Q}_{min}(s) \leftarrow \min_{a \in \mathcal{A}(s)} \mathcal{Q}(s,a)$
  $GreedyAction \leftarrow$ The action associated to $\mathcal{Q}_{min}(s)$
  $p(k) = \frac{G_1}{G_2+k}$
  $RandomNumber \leftarrow$ Generate random number in $[0,1]$
  if *RandomNumber* $\leq (1-p(k))$ **then**
    $a \leftarrow Greedy\_Action$, $\phi \leftarrow 0$
  else
    $a \leftarrow Exploratory\_Action$, $\phi \leftarrow 1$
  $j \leftarrow Next\_State(s,a)$
  $\mathcal{Q}(s,a) \leftarrow (1-\rho^k)\mathcal{Q}(s,a) + \rho^k[\mathcal{C}(s,a,j) - \mathcal{C}^k t(s,a,j)$
  $+\eta \min_{b \in \mathcal{A}(j)} \mathcal{Q}(j,b)]$
  if $\phi = 1$ **then**
    Go to Step 32
  else
    $total\_cost \leftarrow total\_cost + \mathcal{C}(s,a,j)$
    $total\_time \leftarrow total\_time + t(s,a,j)$
    $\mathcal{C}^k \leftarrow (1-\tau^k)\mathcal{C}^k + \tau^k[\frac{total\_cost}{total\_time}]$
  $k \leftarrow k+1$, $i \leftarrow j$
  if $i = Last\_State$ **then**
    $i \leftarrow First\_State$
until $k > ITERMAX$;
for $i \in \mathcal{S}$ **do**
  $a^\star \leftarrow$ The action associated to $\mathcal{Q}_{min}(s)$
  $\pi^\star \leftarrow \pi^\star \cup \{s, a^\star\}$
Stop

$p(k) = \frac{G_1}{G_2+k}$, where $G_2 \geq G_1$, and G_1 and G_2 are large positive constants, e.g., 1000 and 2000 respectively. Then, one of the other actions is chosen with a probability $p(k)$. The non-greedy actions are called exploratory actions, and selecting an exploratory action is called exploration. Note that the probability of exploration will decay with the number of iterations k. Let denote the selected action by a. If a is a greedy action, then set $\phi = 0$; otherwise, set $\phi = 1$. Next, our approach simulates action a. To do so, let j be the next state, and $C(i,a,j)$ denote the transition cost. Let $t(i,a,j)$ denote the transition time, i.e., the computation/offloading time induced from state i to state j. Then, the state-action value function, $\mathcal{Q}(i,a)$, will be updated as follows:

$$\mathcal{Q}(i,a) \leftarrow (1-\rho^k)\mathcal{Q}(i,a) + \rho^k[\mathcal{C}(i,a,j) - \mathcal{C}^k t(i,a,j) + \eta \min_{b \in \mathcal{A}(j)} \mathcal{Q}(j,b)]$$

where $\min_{b \in \mathcal{A}(j)} \mathcal{Q}(j,b)$ denotes the minimum *Q-function* value of the state j.

. **Step** 2 **(Average Cost Update)**: If $\phi = 1$, i.e., the action a was non-greedy, go to Step 3. Otherwise, update the total cost and total transition time respectively as follows:

$$total\_cost \leftarrow total\_cost + \mathcal{C}(i,a,j)$$
$$total\_time \leftarrow total\_time + t(i,a,j)$$

Then, update the average cost as follows:

$$\mathcal{C}^k \leftarrow (1-\tau^k)\mathcal{C}^k + \tau^k[\frac{total\_cost}{total\_time}] \quad \text{(VI.20)}$$

. **Step** 3 **(Check for Termination)**: Increment k by 1. Set $i \leftarrow j$. If $k <$ ITERMAX, return to Step 1. Otherwise, go to Step 4.

. **Step** 4 **(Outputs)**: For each state i, declare the action for which $\mathcal{Q}(i,.)$ is maximum to be the optimal action, in order to generate a policy, and STOP.

The QL−JTAR is summarized in the pseudo Algorithm 1.

VII. Performance evaluation

In this section, we provide series of network simulation results to assess the performance of our proposed joint offloading and resource allocation solution QL−JTAR. Hereafter, we describe the simulation set up and define the performance metrics considered to evaluate our proposal. Then, we analyze the results and discuss the effectiveness of our proposal.

A. Simulation Environment and methodologies

1) Experiment Design: We conduct our simulations within NS3[2] network simulator, which is a discrete-event driven network simulation platform based on C++ language and widely used by research community. To realize WiFi links, we make use of IEEE 802.11ac standard.

We set the propagation parameters based on the IEEE 802.11ac [6]. We configure the physical layer with free-space propagation model. Besides, we build WiFi-based stadium MEC system. The geographical dimensions of the the stadium correspond to the dimensions of a football stadium, i.e., $200m \times 100m$, and thus with a total perimeter of $600m$. It forms a grid composed of 12 wireless cells, each cell corresponds to a wireless communication range of IEEE 802.11ac, i.e., $50m$. On each cell, we deploy 6 Access Points (APs), making use of the 6 available wireless channels, and a set of mobile devices. Each SMD is associated to the less heavy AP. Moreover, the APs of each cell are connected to one aggregation MEC server based on high-speed wired links. The aggregation MEC servers are connected to: i) each other based on ring topology as argued in [16], and ii) a core Cloudlet node, via 10 Gbps links. Similarly to [11], we assume that the propagation delay of wired links is set to $1ms$. Finally, we implement our proposed solution based on C++ language and CPLEX[3] solver.

[2]https://www.nsnam.org
[3]http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer

TABLE I
AVERAGE NETWORK METRICS

	$\mathbb{D}_a$ (second)	$\mathbb{E}_a$ (Joule)
QL-JTAR	$93.46 \pm 0.09\%$	$9.36 \pm 0.04\%$
JOR-MEC	$89.32 \pm 0.24\%$	$4.03 \pm 0.12\%$
Full-Local	$4002.23 \pm 3.81\%$	$94.44 \pm 0.0\%$
Full-Cloud	$164.39 \pm 0.05\%$	$4.27 \pm 0.001\%$

TABLE II
AVERAGE RUNNING TIME METRIC

	QL-JTAR	JOR-MEC
$\mathbb{T}^{ex}$ (seconds)	$0.00026 \pm 0.000005\%$	$0.028 \pm 0.00007\%$

We compare QL-JTAR to the related strategies, namely: i) JOR-MEC, corresponding to the optimal Joint Offloading and Resource allocation exact solution, ii) Full-Cloud, referring to a full Cloud computation and ii) Full-Local representing a total local mobile computation.

2) Simulation setup: We assume that application demands arrival on each cell of the stadium follows a Poisson process with arrival rate λ_A set to 5 requests per second. Moreover, the traffic generated by each mobile device follows a Constant Bit Rate (CBR) model. We consider in our simulations a realistic call graph corresponding to a face recognition application [9] running on SMDs. The offloading data size of the tasks follows a uniform distribution with mean ranging from 100 Mbis to $\mathcal{N}_{max}$. The latter varies between different values during simulations. We set the CPU speed to 10^8 cycles/sec for mobile devices, and 10^{10} cycles/sec for MEC server, as in [9]. We make use of TCP protocol to transmit traffic of tasks from one site to another. We run the simulations for 100 application requests for each cell, and thus for 1200 demands in all the stadium. Note that each performance value of the implemented strategies is equal to the average of 5 simulations. Furthermore, our simulation results are always presented with confidence intervals corresponding to a confidence level of 95%.

B. Performance metrics

Hereafter, we define the performance metrics used to evaluate our approach:

- $\mathcal{T}^{Tot}$: corresponds to the cumulative completion delay of all the completed application demands in the network.
- $\mathcal{E}^{Tot}$: represents the cumulative energy consumption (computation and transmission) on the mobile side.
- $\mathcal{T}^{a}$: corresponds to the average completion delay of all the completed application demands in the network.
- $\mathcal{E}^{a}$: represents the average energy consumption (computation and transmission) on the mobile side.
- $\mathbb{T}^{ex}$: refers to the running time of the proposed approach.

C. Simulation Results

To assess the efficiency of our joint computation offloading and ressource allocation approach, we consider the call graph of realistic face recognition application. Basically, our simulations include two scenarios. In the first scenario, *Offloading-Comparison scenario*, we set the maximum exchanged data $\mathcal{N}_{max}$ to 250 $Mbps$, and the transmit power of mobile devices to 10 mW. We calculate the cumulative/average metrics as well as the running time $\mathbb{T}^{ex}$ for the four approaches: i) QL-JTAR, ii) JOR-MEC, iii) Full-Cloud, and iv) Full-Local. In the second scenario, $\mathcal{N}_{max}$*-Variation scenario*, we evaluate the performance of our proposed solution while varying the maximum exchanged data size $\mathcal{N}_{max}$.

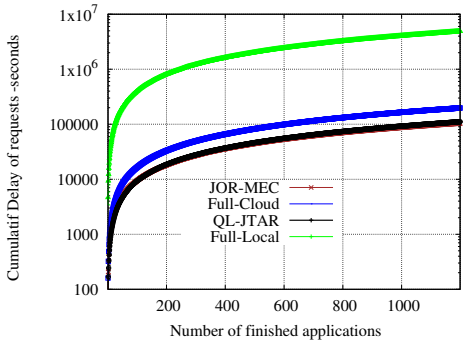
1) Offloading-Comparison Scenario: Fig. 2(a) shows that our proposal QL-JTAR reduces the total completion delay by 43.8% compared to the Full-Cloud approach and by 97% compared to the Full-Local method. On the other hand, QL-JTAR ensures a near-optimal delay, as the gap with the optimal solution is approximately equal to 6.4%. Obtained results corroborate those illustrated in TABLE I which show that the average delay is alleviated compared to Full-Cloud and Full-Local methods. Note that QL-JTAR increases $\mathbb{D}_a$ by only 4.4% compared to the optimal solution JOR-MEC.

Fig. 2(b), illustrates the cumulative energy for each request. It is straightforward to see that QL-JTAR reduces the total energy consumption by approximately 89% compared to Full-Local approach. In contrast, *Full-Cloud* achieves a lower cumulative energy by the end of simulation. This can be explained by the fact that all tasks are computed in the MEC side and hence, mobile computation energy is not considered. The optimal solution alleviates the cumulative energy by approximately 36% compared to *Full-Cloud*. These results approve those illustrated in TABLE I that shows that: QL-JTAR alleviates $\mathbb{E}_a$ by roughly 90% compared to Full-Local, and the energy by 56% compared to Full-Cloud .

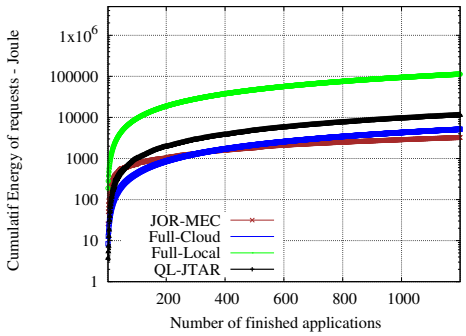
Although the optimal solution outperforms our approach in terms of delay and energy, TABLE II shows that QL-JTAR impressively reduces the computation time compared to the exact solution. Indeed, $\mathbb{T}^{ex}$ is roughly 100 times lower thanks to the reinforcement learning algorithm.

2) N_{Max}-Variation Scenario: Fig. 3(a) shows that the larger the value of $\mathcal{N}_{max}$ is, the higher is the average delay $\mathcal{T}^{a}$ of the ended application demands. This is due to the increase in the amount of exchanged data from one side to the other, which induces higher transmission latency. Moreover, we notice that the average completion delay is dramatically reduced compared to *Full-Cloud* and *Full-Local*. In addition, it is clear that QL-JTAR reduces the average delay in a close way to the optimal solution. Indeed, the average delay is impressively alleviated with a gap of 6.7% compared to optimal when $\mathcal{N}_{max} = 150$ $Mbps$ and with a gap of 4.1% for $\mathcal{N}_{max} = 400$ $Mbps$. This is can be explained by the fact that our approach minimizes both the delay and energy as shown in *Q-function* given in formula (VI.20), while JOR-MEC only minimzes energy and considers delay as constraint.

Similarly, Fig. 3(b) depicts the variation of $\mathcal{E}^{a}$ when $\mathcal{N}_{max}$ ranges from 150 $Mbps$ to 400 $Mbps$. We remark that the average energy consumption increases as the size of exchanged data increases which corroborate the above results. We notice, in addition, that the larger $\mathcal{N}_{max}$, the closer to optimal our approach QL-JTAR becomes. Indeed, while the average energy is reduced with a gap of 73% compared to optimal

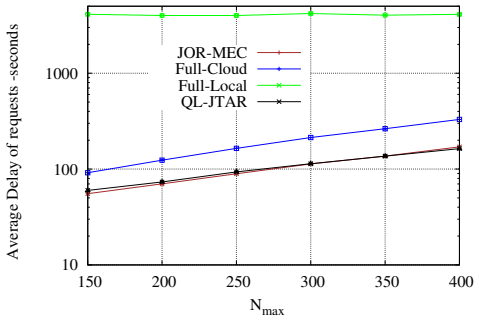


(a) $\mathcal{T}^{Tot}$

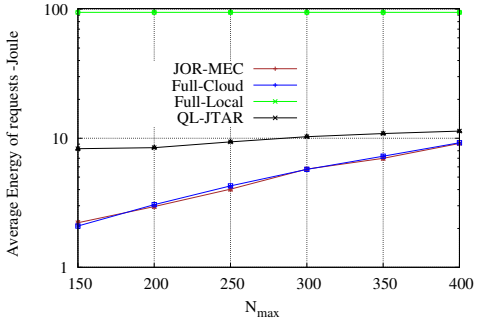


(b) $\mathcal{E}(X)$

Fig. 2. Cumulative completion delay and energy spent for processing



(a) Average Delay



(b) Average Energy

Fig. 3. Average metrics per different maximum size N_{Max}

when $\mathcal{N}_{max} = 150\ Mbits$, it is impressively alleviated with a gap of 19.5% for $\mathcal{N}_{max} = 400\ Mbps$. Therefore, our approach performs well for heavy mobile applications.

VIII. Conclusion

In this paper, we addressed the problem of joint computation offloading and resource allocation in Mobile Edge Cloud. We envisioned a MEC architecture based on ETSI system and using wireless IEEE 802.11ac standard. We formulated the problem based on reinforcement learning technique. To do so, we proposed a novel joint task assignment and resource allocation solution based on *Q-Learning* algorithm, named `QL-JTAR`. The objective of our proposal is to minimize energy consumption on the mobile terminal side, while considering the application delay constraint. The conducted simulations in NS-3 simulator for a real foot-ball stadium MEC use case, show that our proposal outperforms the related strategies in terms of energy consumption and latency. Moreover, the obtained results show that `QL-JTAR` ensures near-optimal solution, while reducing the computation time.

ACKNOWLEDGEMENT

This work was supported by the FUI PODIUM project (Grant no. 10703).

References

[1] "Cisco visual networking index: Forecast and methodology, 2015-2020," CISCO, Tech. Rep., 2016.

[2] H. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, December 2013.

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, October 2009.

[4] "Mobile edge computing (MEC); framework and reference architecture, ETSI GS MEC 003," ETSI Industry Specification Group (ISG), Tech. Rep., March 2016.

[5] E. Cuervo, A. Balasubramanian, D. k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," *ACM Mobisys*, June 2010.

[6] "802.11ac in-depth, white paper," Aruba Networks, Tech. Rep., September 2016.

[7] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, October 2016.

[8] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Transactions on Mobile Computing*, November 2017.

[9] P. Lorenzo, S. Barbarossa, and S. Sardellitti, "Joint optimization of radio resources and code partitioning in mobile edge computing," *IEEE Transactions on Signal Processing*, February 2016.

[10] C. You, K. Huang, H. Chae, and B. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, March 2017.

[11] T.-Q. Dinh, J. Tang, Q. Duy, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, August 2017.

[12] "Mobile edge use cases and deployment options - white paper," Juniper networks, Tech. Rep., July 2016.

[13] Z. Jiang and S. Mao, "Energy delay tradeoff in cloud offloading for multi-core mobile devices," *IEEE Access*, November 2015.

[14] A. Gosavi, "Reinforcement learning for long-run average cost," *European Journal of Operational Research*, June 2004.

[15] ——, "A tutorial for reinforcement learning," Missouri University of Science and Technology, Tech. Rep., February 2017.

[16] A. Ceselli, M. Premoli, and S. Secci, "Mobile edge cloud network design optimization," *IEEE Transactions on Networking*, June 2017.