Maximizing Mobiles Energy Saving Through Tasks Optimal Offloading Placement in two-tier Cloud: a Theoretical and an Experimental Study

Houssemeddine MAZOUZI^a, Khaled BOUSSETTA^{a,b}, Nadjib ACHIR^a

^a*Laboratory L2TI, Institut Sup Galilée, Université Paris 13, Sorbonne Paris Cité 99
Avenue J-B Clement, 93430 Villetaneuse, France*

^b*Laboratory Agora, CITI, Inria/INSA Lyon, 56 Boulevard Niels Bohr, 69100
Villeurbanne, France*

Abstract

In this paper, we focus on tasks offloading over two tiered mobile edge computing environment. We consider several users with energy constrained tasks that can be offloaded over edge clouds (cloudlets) or on a remote cloud with differentiated system and network resources capacities. We investigate offloading policy that decides which tasks should be offloaded and determine the offloading location on the cloudlets or on the cloud. The objective is to minimize the total energy consumed by the users. We formulate this problem as a Non-Linear Binary Integer Programming. Since the centralized optimal solution is NP-hard, we propose a distributed linear relaxation heuristic based on Lagrangian decomposition approach. To solve the sub-problems, we also propose a greedy heuristic that computes the best cloudlet selection and bandwidth allocation following tasks' energy consumption. We

*Corresponding author

Email addresses: mazouzi.houssemeddine@univ-paris13.fr (Houssemeddine MAZOUZI), khaled.boussetta@univ-paris13.fr (Khaled BOUSSETTA), nadjib.achir@univ-paris13.fr (Nadjib ACHIR)

Preprint submitted to Computer Communications

May 8, 2019

compared our proposal against existing approaches under different system parameters (CPU resources), variable number of users and for six applications, each having specific traffic pattern, resource demands and time constraints. Numerical results show that our proposal outperforms existing approaches. In addition to the theoretical approach, we evaluate our offloading policy using real experiments. In this case, we setup a real testbed composed of client terminal, offloading server located either at the edge or at a remote Cloud. We also implemented our proposal as an offloading middleware on both the client and the offloading server. Using this testbed, we were able to evaluate our offloading decision policy for multi-users context with three real Android OS applications, with different traffic patterns and resource demands. We also discuss the performance of our proposal for each application and we analyze the multi-users effect.

Keywords:

computation offloading; mobile cloud computing; mobile edge computing; cloudlet; Lagrangian decomposition; Offloading Middleware.

1. Introduction

In recent years, mobile devices have undergone a major transformation, moving from small devices with limited capabilities to important everyday accessories with important capabilities. This recent advances in hardware and software mobile technology have led to an exponential growth in mobile application markets. Unfortunately, even as mobile applications become more and more intensive, the computing power of mobile devices remains limited compared to what we can find in data-centers and cloud. Furthermore,

9 because the limited weight and size and therefore the life of the battery, a
10 powerful approach to improving the performance of mobile applications and
11 reducing the shortage of mobile device resources is required. One possible ap-
12 proach is to enable mobile devices to offload some of their intensive workloads
13 to remote high-performance virtual machines. Unfortunately, even though
14 clouds have rich computing and storage resources, they are generally geo-
15 graphically far away from users. In this case, this approach may suffer from
16 significant and fluctuating delays on the Internet. This finding is particularly
17 problematic for some mobile applications, such as augmented reality or cloud
18 gaming, which requires a reduced response time.

19 To reduce this long access delay, an emerging tendency is to push the cloud
20 to the network edge, mainly located within existing wireless Access Points
21 (APs), ADSL box or Base Stations (BSs). This proximity gives users the
22 opportunity to offload their tasks to this Edge cloud or cloudlets. This new
23 paradigm is known as "**Mobile Edge Computing (MEC)**". A cloudlet
24 can be seen as a small data center, and because of this geographical proximity
25 between users and cloudlets, the access delay on the task offloading can be
26 greatly reduced, compared to remote clouds, and thus significantly improving
27 user experiences. In this paper, we focus on multi-user MEC, where users
28 offload their applications to edge servers or cloudlets. In this case, both the
29 wireless bandwidth and the computing resources must be shared among the
30 users.

31 The work presented in this paper is an extension of our previous work [1].
32 In [1], we present a computation offloading policy for multi-users multi-
33 cloudlets environment, named Efficient cloudlet Selection Offloading policy

(ECESO). Our objective is to determine which users should offload their tasks and to which cloudlet in order to minimize the overall energy consumed by the users. Basically, ECESO assigns each user to the best cloudlet in order to reduce the energy consumption of all users, according to the available network and system resources. We formulate this problem as a Binary Integer Programming (BIP) and we propose a distributed linear relaxation heuristic based on Lagrangian decomposition approach. Our policy consists of two decision levels:

- The local offloading decision level that concerns the users associated with the same access point (AP), in order to solve the offloading subproblem of this AP.
- The global offloading decision level ensures that the offloading solution given by the local offloading decision level complies with the cloudlet resource constraints.

In addition to this theoretical approach, we added in this paper a complete experimental approach in order to evaluate our proposal. Our objective was to evaluate our proposed policy with real Android mobile applications, we also designed and implemented a computation offloading middleware for Android-based terminals. This middleware was integrated on both the mobile terminal and the offloading server. Basically, we integrated our offloading policy to the mobile terminal via a client offloading middleware in order to decide if the task should be executed locally or offloaded to the edge/remote cloud. In the remote virtual machine, we also integrated a server offloading middleware to ensure that the offloaded tasks are correctly executed. Using

58 this testbed we were able to run multi-users offloading experiments in local
59 edge and in the cloud, with the aim to compare the observed performances
60 to the placement decision made by our proposal ECESO.

61 The rest of paper is organized as follows: Section 2 presents related work,
62 and section 3 describes the modeled system. Problem formulation and solving
63 are detailed in Section 4. Theoretical and experimental evaluation of our
64 offloading policy is discussed in section 5. Finally, a conclusion is drawn in
65 Section 6.

66 2. Related Work

67 Several works were proposed to explore computation offloading in order
68 to improve the performance of the mobile devices. Some work focused on
69 the wireless bandwidth allocation in order to take offloading decision, such
70 as Meng-Hsi Chen et al. [2], Xu Chen et al. [3], Songtao Guo et al. [4, 5],
71 and Keke Gai et al. [6]. The work presented by Meng-Hsi et al. is one of
72 the first works supporting multi-user computation offloading in mobile cloud
73 computing. It decides which task must be performed in the remote cloud
74 and which task must be performed locally. Then, it allocates the wireless
75 bandwidth to each offloaded task in order to reduce the energy consumption
76 of the mobile device. Xu Chen et al. policy was designed to a single cloudlet
77 mobile-edge environment. Each user tries to offload its tasks, accordingly
78 with the available wireless bandwidth to reduce the energy consumption.
79 Another offloading approach for multi-users was presented by Songtao Guo
80 et al. The offloading policy decides which tasks should be offloaded and
81 allocates the wireless bandwidth to each offloaded task. Then, it allocates

82 the local processor frequency. Lastly, Keke Gai et al. propose a scheduler to
83 assign the tasks between the local mobile device and the remote cloud in order
84 to save energy consumption. In a multi-cloudlet scenario, the computational
85 capacity of each cloudlet is limited, unlike these heuristics, the computation
86 offloading policy must select the best cloudlet to each user with the purpose
87 to achieve high performance.

88 More recently, many works focus on cloudlets placement heuristics in the
89 MEC environment. The main goal is to find how many cloudlets are needed
90 and where place them, such as Mike Jia et al. [7, 8], Hong Yao et al. [9],
91 and Longjie Ma et al. [10]. Mike Jia et al. heuristic tries to find the best
92 cloudlets placement in a large network, then select a cloudlet to perform
93 the computation tasks of each AP. The K-median clustering based on user
94 density is used to place the cloudlets. Then each AP is statically assigned to a
95 cloudlet. Similarly, Hong Yao et al. was designed to support heterogeneous
96 cloudlets environment. Finally, Longjie Ma et al. was introduced to find
97 the minimal number of cloudlets that must be placed to improve the user
98 experience quality. However, the density of mobile-users are dynamic and
99 changes over time. So, static assignment of the APs to cloudlets may decrease
100 the performance of the computation offloading. To confirm this assumption,
101 our heuristic ECESO consider dynamic cloudlet selection and the wireless
102 bandwidth allocation with the aim of minimizing energy consumption of
103 mobile devices.

104 Anwesha Mukherjee et al. [11, 12] and Mike Jia et al. [13] focus on the
105 dynamic cloudlet selection in order to reduce the offloading cost. Anwesha
106 Mukherjee et al. designed a multi-level offloading policy to optimize the

107 energy consumption. The users offload to the nearest cloudlet in the first step.
 108 According to the resource availability in this cloudlet, it can perform the tasks
 109 or offload the task to another cloudlet and so on. Mike Jia et al. introduced a
 110 heuristic to balance the load between the cloudlet. Its main goal is to migrate
 111 some tasks from overloaded cloudlets to underloaded cloudlets to reduce the
 112 execution time. These works propose dynamic cloudlet selection heuristics,
 113 but they do not consider the wireless bandwidth in a multi-user environment.
 114 In our previous work [14], we introduce D2M-ECOP a new offloading policy
 115 for multi-cloudlet MEC environment. D2M-ECOP focuses on selecting the
 116 best cloudlet dynamically to perform the tasks of each user. This proposition
 117 achieves high perform in multi-cloudlet MEC environment. However, it does
 118 not consider the offloading to the remote cloud in case of overloaded of the
 119 cloudlet, which can affect the performance of the computation offloading in
 120 such scenario.

121 All the offloading policies presented above focus on reducing the offloading
 122 cost and try to offload the tasks to a predetermined offloading server, a
 123 remote cloud or cloudlet. Consequently, the performance of computation
 124 offloading can be decreased due to the dynamic density of users in such
 125 environment. Therefore, designing a new offloading policy is mandatory.
 126 The new policy must consider many offloading servers for whom a user can
 127 offload its tasks, and compute optimal task placement by considering two-tier
 128 MEC environment.

129 Other works of the literature have proposed computation offloading plat-
 130 forms for mobile application. Eduardo Cuervo et al. [15] implement Maui
 131 platform for Windows Phone terminals. This platform uses interface and

132 annotations programming paradigm [16] to indicate which task to offload.
 133 Byung-Gon Chun et al. [17] have proposed offloading platform named Clonecloud,
 134 which uses Android Virtual Machine to offload tasks on the remote cloud.
 135 Clonecloud uses threads-based offloading strategy to offload tasks in order
 136 to reduce the completion time of the application. Similarly, Jose Benedetto
 137 et al. [18] design MobiCOP, which is an Android computation offloading
 138 platform based on Google cloud computing platform. Unfortunately, all the
 139 platforms described above have been designed to make the offloading pos-
 140 sible but not to choose where to offload. In addition, they do not consider
 141 offloading to a multi-cloudlet MEC environment. In this paper, we developed
 142 a new computation offloading platform dedicated to multi-cloudlet MEC en-
 143 vironment. Using this platform, we set up a real experimentation in order
 144 to validate our offloading policy.

145 **3. System description and modeling**

146 This section presents the MEC system modeling. It describes the compu-
 147 tation offloading tasks model, and the network communication model. Then,
 148 it details the offloading cost considered in our offloading problem. Table 1
 149 presents all the variables used to modeling our multi-user multi-cloudlet com-
 150 putation offloading problems.

Table 1: Notations and definitions used in the problem modeling

Symbol	Notations and Definitions
K	the number of cloudlets available in the network.
M	the number of APs in the network.
N_m	the number of users associated to the AP m .
$f_{m,n}$	the local computing capacity of the n^{th} user of the m^{th} AP.
F_k	the computing capacity of the cloudlet k .
c_k	the computing resource allocation on cloudlet k .
$dw_{m,n}$	the amount of data to download by the n^{th} user of the m^{th} AP from the MEC.
$up_{m,n}$	the amount of data uploaded to the MEC from the n^{th} user of the m^{th} AP.
$B_{m,n}^{up}$	the allocated upload data rate for the n^{th} user of the m^{th} AP.
$B_{m,n}^{dw}$	the allocated download data rate for the n^{th} user of the m^{th} AP.
$P_{m,n}^{tx}$	power consumption when the Wi-Fi interface is transforming data.
$P_{m,n}^{rx}$	power consumption when the Wi-Fi interface is receiving data.
$P_{m,n}^{Idle}$	power consumption when the Wi-Fi interface is in Idel state
$t_{m,n}$	the maximum tolerated delay according the QoS of the task of the n^{th} user of the m^{th} AP.
$T_{m,n,k}^t$	the communication time when n^{th} user of the m^{th} AP offload to cloudlet k .
$T_{m,n}^l$	the local processing time for n^{th} user of the m^{th} AP.
$T_{m,n,k}^e$	the remote processing time for n^{th} user of the m^{th} AP in cloudlet k .
$Z_{m,n}^l$	the local energy consumption for n^{th} user of the m^{th} AP.
$Z_{m,n,k}^e$	the remote energy consumption for n^{th} user of the m^{th} AP in cloudlet k .
$\gamma_{m,n}$	the computational resource required by the task of the n^{th} user of the m^{th} AP.
λ_m	the Lagrangian multiplier of the subproblem m .
$x_{m,n}^k$	the offloading decision variable for the task of n^{th} user of the m^{th} AP in the cloudlet k .
$y_{m,n}$	the category to which belong the tasks (static or dynamic offloading decision task).

Figure 1: MEC environment

152 We consider the MEC environment illustrated in Figure 1. The infras-
 153 tructure is composed of M APs, K deployed cloudlets and one remote cloud.
 154 In the remainder, we will refer to the cloud as the $(K + 1)$ th offloading
 155 server. Similarly to [8], we assume that the number of cloudlets is less than
 156 the number of access points ($K < M$).

157 Let $\mathcal{M}$ denote the set of APs. The remote cloud and the cloudlets consti-
 158 tute a set, denoted $\mathcal{K}$, of offloading servers. All these servers offer computa-
 159 tion resources to perform offloaded tasks. In each server $k \in \mathcal{K}$, the resources
 160 are characterized by a fixed capacity, denoted F_k , of computational resource

units. A computational resource unit is expressed in Ghz and is defined as the number of cycles per second allocated to perform a task. We denote c_k the number of cycles per second allocated by servers k to perform any given offloaded task. Similarly to [4, 3], we consider that $c_k, \forall k \in \mathcal{K}$, is fixed and does not change during the computation. We also assume that the number of computation resources units, F_k , is limited in cloudlets while it is infinite in the cloud. Formally, $\forall k \in \{1, 2, \dots, K\}, F_k \ll F_{K+1} = \infty$.

3.2. Tasks Requirements

Let consider that the system is observed at a given time. Extension to continuous observation time is possible by discretizing the time into contiguous observation intervals. For each observation time, we assume that each AP $m \in \mathcal{M}$ is associated to a set of users, denoted $\mathcal{N}_m$ of size N_m .

Let (m, n) denote the n^{th} user associated with the m^{th} AP. At observation time, we assume that each user $(m, n), \forall m, n \in (\mathcal{M}, \mathcal{N}_m)$, have exactly one task, denoted $r_{m,n}$. This task is characterized by the number of CPU cycles needed for its computation, denoted in the following as $\gamma_{m,n}$. The purpose of this work is to decide if a task should be performed locally, on the user's terminal or remotely in one of the $\mathcal{K}$ servers. To this end, and similarly to [19], we distinguish two offloading decision tasks:

1. The *static offloading decision task*
2. The *dynamic offloading decision task*

For the first category, the tasks are always offloaded. The offloading decision is taken when the application is designed. In this case, the tasks must be performed remotely on the cloudlets or cloud either because they

185 require some specific hardware/software environment (e.g. GPU, operating
 186 systems) which are not available on the user's terminal or in order to fulfill
 187 some application's constraints (e.g. security issues). Accordingly, for static
 188 offloading decision task, our objective is to select in which server $k \in \mathcal{K}$ the
 189 task must be offloaded.

190 For the second category, the offloading decision is taken at runtime. These
 191 tasks can be executed either locally or offloaded to one of the $k \in \mathcal{K}$ offloading
 192 servers. In this case, our objective is to decide if a *dynamic offloading decision*
 193 *task* has to be executed locally (on the user's device) or if it should be
 194 offloaded, and if so, to which offloading server $k \in \mathcal{K}$.

195 Finally, in order to differentiate between the two categories, we associate
 196 to each task $r_{m,n}$ a binary variable $y_{m,n}$, equal to 0 if $r_{m,n}$ is a *dynamic*
 197 *offloading decision task* and to 1 if $r_{m,n}$ is a *static offloading decision task*.

198 3.3. Local Processing time

199 In the case where a task, $r_{m,n}$, is performed locally, the execution time of
 200 the task only depends on the computing capabilities of the user's terminal.
 201 Indeed, the transmission time is equal to zero. In the following, we assume
 202 that the user's device can allocate, at a given observation time, a computation
 203 capacity, denoted as $f_{m,n}$, to perform the task locally. We also assume that
 204 this computing capacity remains unchanged throughout the execution of the
 205 task. This capacity is expressed as the number of cycles per second. We can
 206 therefore deduce the local processing time of the task $r_{m,n}$ as follows:

$$T_{m,n}^l = \frac{\gamma_{m,n}}{f_{m,n}} \quad (1)$$

207 3.4. Remote Processing time

208 In the case where the task $r_{m,n}$ is offloaded on server k then the following
209 steps are required:

- 210 1. In order to perform task $r_{m,n}$ remotely on server k a given amount of
211 data must be uploaded from the user's terminal to server k , denoted
212 hereinafter as $up_{m,n}$ and expressed in bits. The time required to trans-
213 mit this data depends on the bandwidth available for the user's (m,n)
214 terminal at the AP m and also to the delay from the AP m to the
215 server k on the backhaul network. Lets denote by $B_{m,n}^{up}$ the upload
216 bandwidth allocated to the user m,n in the AP m to transmit the data
217 required for the task $r_{m,n}$. We will detail in section 3.5 how to compute
218 this bandwidth when a given number of users associated to a same AP
219 must offload their tasks. Accordingly, we can then derive the transmis-
220 sion time to upload the data of an offloaded task $r_{m,n}$ from user (m,n)
221 terminal to its AP m as follows:

$$T_{m,n}^{up} = \frac{up_{m,n}}{B_{m,n}^{up}} \quad (2)$$

222 Similarly, let $B_{m,k}^{bh}$ denotes the end to end backhaul bandwidth between
223 the AP m and the offloading server k . We can express the transmission
224 time of the data associated with an offloaded task $r_{m,n}$ from AP m to
225 server k as follow:

$$T_{m,n}^{bh,k} = \frac{up_{m,n}}{B_{m,k}^{bh}} \quad (3)$$

- 226 2. This uploaded data is then used by the offloaded server k to perform

227 the task. The remote computing time of task $r_{m,n}$ can be expressed as
 228 the ratio of the CPU cycles required for the task's computing ($\gamma_{m,n}$) to
 229 the number of cycles per second allocated at server k (c_k):

$$T_{m,n}^{e,k} = \frac{\gamma_{m,n}}{c_k} \quad (4)$$

230 3. Finally, when the task's computation is finished, the server k returns
 231 back the results to the user's terminal. We denotes by $dw_{m,n}$ the
 232 amount or results data returned back to the user (m, n). This quantity
 233 is also expressed in bits. We can thus derive the transmission time of
 234 task $r_{m,n}$ from server k to AP m as follow:

$$T_{m,n}^{k,bh} = \frac{dw_{m,n}}{B_{m,k}^{bh}} \quad (5)$$

235 In the same way, We can also express the transmission time to download
 236 the remote computation result of the offloaded task $r_{m,n}$ from AP m
 237 to user (m, n) terminal as follows:

$$T_{m,n}^{dw} = \frac{dw_{m,n}}{B_{m,n}^{dw}} \quad (6)$$

238 Here, $B_{m,n}^{dw}$ denotes the allocated bandwidth to transmit the offloaded
 239 task results from the AP m to user (m, n).

240 Finally, we can then compute the completion of task $r_{m,n}$ when the latter
 241 is offloaded on server k as follows:

$$T_{m,n}^k = T_{m,n}^{up} + T_{m,n}^{bh,k} + T_{m,n}^{e,k} + T_{m,n}^{k,bh} + T_{m,n}^{dw} \quad (7)$$

242 3.5. Shared wireless access bandwidth

243 As introduced in the last section, the available bandwidth in upload and
 244 download for each user in an AP depends on the number of concurrent trans-
 245 missions, which means in our case the number of offloaded tasks. In the
 246 following we denote by π_m the number of tasks that are offloaded through
 247 AP $m \in \mathcal{M}$ to a server $k \in \mathcal{K}$.

248 In this paper, we consider that access points use 802.11n technology.
 249 However, this work can be easily extended to cellular technologies by con-
 250 sidering existing models from the literature. As we will emphasize in section
 251 20, the key point here is the fact that the available bandwidth is inversely
 252 proportional to the number of offloaded tasks, which makes our optimization
 253 problem non-linear.

254 In order to estimate the bandwidth at each AP, we adopt the Bianchi
 255 analytical model of WiFi channels [20, 21] where the characteristics of DCF
 256 mechanisms in 802.11n are considered. The parameters are summarized in
 257 Table 2.

Table 2: The characteristics of DCF mechanisms in IEEE 802.11n

Parameter	Value
MAC header	272 bits
PHY header	128 bits
ACK	112 bits + PHY header
W(wireless bandwidth)	150 mbps
CW (Contention Window)	15
R (Maximum backoff counter)	7
ϱ (slot time)	9 μs
ϕ (propagation time)	1 μs
SIFS, DIFS	10 μs , 28 μs

258 Using the Bianchi model [20, 21], the bandwidth of an AP where π_m users
 259 are competing for a transmission can be expressed by the following formula:

$$B(\pi_m) = \frac{E(\text{Payload information transmitted in a slot time})}{E(\text{length of a slot time})} \quad (8)$$

$$= \frac{ps.E}{(1 - pb).\varrho + ps.\overline{T}_s + (pb - ps).\overline{T}_c} \quad (9)$$

260 Where

- 261 • ps denote the probability that a successful transmission occurs in a slot
 262 time.

$$ps = \pi_m \tau (1 - \tau)^{\pi_m - 1}$$

263 Here, τ denotes the stationary probability that one mobile device trans-
 264 mits a packet in a slot time. It can be expressed as:

$$\tau = \frac{1}{1 + \frac{1 - p}{2(1 - p^{R+1})} [\sum_{j=0}^R p^j \cdot (2^j CW - 1) - (1 - p^{R+1})]} \quad (10)$$

265 Where p is the probability of a collision during the transmission of a
 266 packet. We can express it as:

$$p = 1 - (1 - \tau)^{\pi_m - 1} \quad (11)$$

267 Fixed point method can be used to solve equations 10 and 11 to deter-
 268 mine the value of τ .

- E is the average time to transmit the packet payload information of size d . It can be computed as follows:

$$E = \frac{d}{W} * \frac{CW}{CW - 1} \quad (12)$$

- pb denotes the probability that the channel is busy. It can be computed as follows:

$$pb = 1 - (1 - \tau)^{\pi_m} \quad (13)$$

- 269 • $\bar{T}_s$ is the average time the channel is sensed busy because of a successful
 270 transmission. Let T_{MPDU} , T_{ACK} , SIFS and DIFS denote the time to
 271 transmit the MPDU (including MAC header, PHY header), the time
 272 to transmit an ACK, the SIFS time and the DIFS time, respectively.

$$\bar{T}_s = (T_{MPDU} + SIFS + T_{ACK} + DIFS) * \frac{CW}{CW - 1} + \phi$$

273 Here, T_{MPDU} , T_{ACK} and DIFS denote the time to transmit the MPDU
 274 (including MAC header, PHY header), the time to transmit an ACK,
 275 and the DIFS time, respectively.

- 276 • $\bar{T}_c$ is the average time the channel is sensed busy by each station during
 277 a collision.

$$\bar{T}_c = T_{MPDU} + SIFS + T_{ACK} + DIFS + \phi$$

278 Using equation 9 we can compute the allocated bandwidth to transmit
 279 the input data of offloaded task $r_{m,n}$ from user (m, n) to its AP m as follows:

$$B_{m,n}^{up} = \frac{B(\pi_m)}{\pi_m} \quad (14)$$

280 In the same way, we can obtain the allocated bandwidth to transmit the
 281 offloaded task results from the AP m to user (m, n) as follows:

$$B_{m,n}^{dw} = \frac{B(\pi_m)}{\pi_m} \quad (15)$$

282 3.6. Completion time constraint

283 Let $T_{m,n}$ denotes the total processing time of task $r_{m,n}$. From the above
 284 system description, one can see that $T_{m,n}$ depends on the processing time and
 285 eventually, in case of offloading, upload and download transmission times.
 286 Precisely, if a task $r_{m,n}$ is performed locally, then $T_{m,n} = T_{m,n}^l$. Otherwise, if
 287 a task $r_{m,n}$ is offloaded on server k the $T_{m,n} = T_{m,n}^k$.

288 Hence, to integrate applications' Quality of Services requirements, we
 289 associate to each offloadable task a time constraint threshold. Precisely, we
 290 define a maximum completion time threshold, denoted $t_{m,n}$ to any task $r_{m,n}$,
 291 $\forall(m, n) \in (\mathcal{M}, \mathcal{N}_m)$.

292 The completion time is a hard constraint for any task, $r_{m,n}$, $\forall(m, n) \in$
 293 $(\mathcal{M}, \mathcal{N}_m)$. In other words, the optimal offloading policy must satisfy the
 294 following constraint:

$$T_{m,n} < t_{m,n} \quad (16)$$

295 As stated before, the purpose of our offloading policy is to determine

296 which tasks should be offloaded and to which server in order to satisfy the
 297 completion time constraint. In addition to this constraint, our purpose is to
 298 determine the optimal offloading policy which minimizes the overall energy
 299 consumed by the user's terminals. In the following sections we will detail
 300 energy consumption models for both, local and remote tasks processing.

301 3.7. *Energy consumption: local processing*

302 Following the model proposed in [22] of power consumption due to tasks
 303 processing on portable devices, we can then derive the total amount of energy
 304 consumed to process task $r_{m,n}$ locally as follows:

$$\mathcal{Z}_{m,n}^l = \kappa * (f_{m,n})^3 * T_{m,n}^l = \kappa * (f_{m,n})^2 * \gamma_{m,n} \quad (17)$$

305 where κ is the effective switched capacitance, which depends on the chip
 306 architecture, and is used to adjust the processor frequency. Similarly to
 307 [22], we set $\kappa = 10^{-9}$.

308 3.8. *Energy consumption: Remote Processing*

309 The total amount of energy consumed by the user's device to perform the
 310 task remotely is equal to the energy used when the device 1) turns the radio
 311 in the transmission mode to send the data to the remote server, 2) turn the
 312 radio in idle mode to wait the task completion and finally 3) turn the radio
 313 in the reception mode in order to receive the result data from the remote
 314 server. The consumed energy can thus be expressed as follows:

$$\mathcal{Z}_{m,n}^k = P_{m,n}^{tx} * T_{m,n}^{up} + P_{m,n}^{idle} * (T_{m,n}^{bh,k} + T_{m,n}^{e,k} + T_{m,n}^{k,bh}) + P_{m,n}^{rx} * T_{m,n}^{dw} \quad (18)$$

315 where $P_{m,n}^{tx}$ is the power consumption when the radio interface is in transmis-
 316 sion mode, $P_{m,n}^{rx}$ is the power consumption when the radio interface is in the
 317 reception mode and $P_{m,n}^{idle}$ is the power consumption when the radio interface
 318 in idle mode. As is commonly assumed [22], we suppose that

$$P_{m,n}^{idle} \leq P_{m,n}^{rx} \leq P_{m,n}^{tx} \quad (19)$$

319 4. Problem Formulation and Solving

320 As introduced earlier, the objective of this paper is to propose an efficient
 321 offloading policy that decides which tasks should be offloaded and to which
 322 offloading server (cloudlets or cloud), while minimizing the total energy con-
 323 sumed by the mobiles. Given our system description and according to the
 324 QoS and offloading servers' resources capabilities constraints, our problem
 325 can be formulated as follows:

$$\begin{aligned}
& \text{Minimize } \sum_m^M \sum_n^{N_m} \mathcal{Z}_{m,n} \\
& \text{Subject to:} \\
& C1 : \sum_{k=1}^{K+1} x_{m,n}^k \leq 1, \forall m \in \mathcal{M}, (m, n) \in \mathcal{N}_m \\
& C2 : y_{m,n} - \sum_{k=1}^{K+1} x_{m,n}^k \leq 0, \forall m \in \mathcal{M}, (m, n) \in \mathcal{N}_m \\
& C3 : T_{m,n} \leq t_{m,n}, \forall m \in \mathcal{M}, (m, n) \in \mathcal{N}_m \\
& C4 : \sum_m^M \left(\sum_n^{N_m} x_{m,n}^k * c_k \right) \leq F_k, \forall k \in \mathcal{K} \\
& C5 : x_{m,n}^k \in \{0, 1\}, \forall m \in \mathcal{M}, (m, n) \in \mathcal{N}_m, k \in \mathcal{K}
\end{aligned} \tag{20}$$

326 As indicated above, our objective is to minimize the total amount of
 327 energy consumed by the mobiles. Here $x_{m,n}^k$ is the offloading decision of the
 328 task of the user (m, n) to the offloading server k , which means that $x_{m,n}^k = 1$
 329 if the user (m, n) offloads its task to the offloading server k , and 0 otherwise.
 330 $\mathcal{Z}_{m,n}$ is the amount of energy consumed by the task of the user n on the AP
 331 m , and can be computed as following:

$$\mathcal{Z}_{m,n} = \left(1 - \sum_{k=1}^{K+1} x_{m,n}^k \right) * \mathcal{Z}_{m,n}^l + \sum_{k=1}^{K+1} x_{m,n}^k * \mathcal{Z}_{m,n}^k$$

332 Constraint $C1$ ensures that each task is assigned at most to one offloading
 333 server. Constraint $C2$ guarantee that any static offloading decision task must

334 be assigned to exactly one offloading server, and a dynamic offloading decision
 335 task may be assigned to at most one offloading servers. The next constraint
 336 *C3* ensures that the QoS required by the task, in terms of completion time,
 337 must be less than a given threshold. The processing time of task $r_{m,n}$ can be
 338 expressed as following:

$$T_{m,n} = (1 - \sum_{k=1}^{K+1} x_{m,n}^k) * T_{m,n}^l + \sum_{k=1}^{K+1} x_{m,n}^k * T_{m,n}^k$$

339 The next constraint *C4* shows that it is not possible to exceed the offload-
 340 ing capacity of the offloading server. Finally, constraint *C5* ensures that the
 341 decision variable, $x_{m,n}^k$, is a binary variable.

342 **Theorem 1.** The problem defined by equations 20 is a Non-Linear Binary
 343 Integer Problem (NLBIP) with an exponential function and constraints. It
 344 is an NP-hard problem.

345 *Proof.* Let us consider a special case where the same number of users are
 346 associated with each AP and all tasks are static offloading decision. So, all
 347 the tasks must be offloaded to the cloudlets and the bandwidth allocated
 348 to each user is known in advance. Thus, the special case is Linear Binary
 349 Integer Problem (LBIP). In fact, this special case can be easily reduced to
 350 the General Assignment Problem (GAP) with assignment restrictions, which
 351 is NP-hard as shown in [23]. Since the special case is NP-hard, the problem
 352 20 is also NP-hard. $\square$

353 One possible approach to resolve the above problem is to use some de-
 354 composition techniques such as Lagrangian relaxation. Thus, we introduce

the Lagrangian multipliers $\lambda = [\lambda_k, k \in \mathcal{K}]^T$ on the constraint C5, since it is considered as a complicating constraint [24]. Here, λ_k denotes the price of all the tasks performed by the k^{th} offloading server. The Lagrangian function is given by:

$$L(X, \lambda) = \sum_m^M \sum_n^{N_m} (\mathcal{Z}_{m,n} + \sum_k^{K+1} \lambda_k x_{m,n}^k * c_k) - \sum_k^{K+1} \lambda_k F_k$$

In this case, the Lagrange dual problem for the primal problem (20) is then given by:

$$\max_{\lambda} \min_X L(X, \lambda)$$

We can see that the Lagrange dual problem is separable into two levels. The first level is the inner minimizing and consists of M subproblems for the M APs. The second level is the outer maximization and represents the master problem that consider the global variable and constraint C4.

4.1. Local Computation Offloading Decision Heuristic

As introduced in the last section, we decompose the Lagrange Dual problem into M subproblems. Each subproblem concerns one AP and aims to offload the task which belongs to the users associated to that AP. This subproblem can be formulated as following:

$$\begin{aligned} & \textbf{Minimize} \sum_n^{N_m} (\mathcal{Z}_{m,n} + \sum_k^{K+1} \lambda_k \cdot x_{m,n}^k \cdot c_k) \\ & \textbf{Subject to:} \\ & \text{constraints C1 - C3 and C5} \end{aligned} \tag{21}$$

368 From the last formulation, we can observe that we need to compute the
 369 bandwidth allocated to each user. Unfortunately, according to equation 8 this
 370 bandwidth depends on the number of users that offload their tasks (π_m). To
 371 overcome this dependency problem, we use a branching heuristic. Basically, it
 372 consists of finding the optimal value of π_m , that gives the minimum offloading
 373 cost of the subproblem 21. We can easily derive a lower bound for π_m , since
 374 the minimum number of tasks that should be offloaded by the users are the
 375 tasks that are considered as *static offloading decision tasks*. Similarly, we can
 376 also derive an upper bound for π_m , which corresponds to the total number
 377 of tasks (N_m) belonging to the users of the AP m . Consequently, we have
 378 to add one additional constraint C6 to the subproblem formulation (21), as
 379 following:

$$C6 : \sum_n^{N_m} \sum_k^{K+1} x_{m,n}^k = \pi_m \quad (22)$$

380 To solve our subproblem, we propose a distributed greedy heuristic to se-
 381 lect which user should offload its task and to which offloading server (cloudlet).
 382 Our proposed heuristic is illustrated in the algorithm 1. Basically, we start
 383 our algorithm by finding the best offloading server for all static offloading
 384 tasks, that minimize the Lagrangian cost $\mathcal{Z}_{m,n}^k + \lambda_k \cdot c_k$ under the constraints
 385 C1 – C3 and C5 – C6. There after, since the wireless bandwidth at the AP
 386 maybe not enough to offload all the remaining dynamic offloading decision
 387 tasks, we propose to define an order to determine which task must be of-
 388 floaded at first. To do this, we compute for each dynamic offloading decision

task an offloading priority defined as following:

$$a_{m,n} = \mathcal{Z}_{m,n}^l - \min_{k \in \mathcal{K}}(\mathcal{Z}_{m,n}^k) \quad (23)$$

This offloading priority depicts the potential gain, in terms of energy saving, between a local execution or an offloading of the task. The idea here is that when $a_{m,n}$ increases, the offloading of the task is preferred since more energy is saved at the mobile. Finally, once the number of the offloading task is equal to the current offloading capacity (π_m), the remaining tasks are assigned to being performed locally by the users.

Algorithm 1 *ECESO offloading heuristic*

Output: output the offloading decisions $\mathcal{X}$ and cost $\mathcal{Z}$;

- 1: **for** each value of π_m **do**
 - 2: allocate bandwidth using equation 14 and 15 ;
 - 3: offload each static offloading decision task to the offloading server k that minimizes $\mathcal{Z}_{m,n}^k + \lambda_k c_k$ under constraints $C1 - C3$ and $C5 - C6$.
 - 4: compute $a_{m,n}$ for every dynamic offloading decision task ;
 - 5: Sort dynamic offloading decision tasks in decreasing order of $a_{m,n}$;
 - 6: offload each dynamic offloading decision task to the offloading server k that minimizes $\mathcal{Z}_{m,n}^k + \lambda_k c_k$ under constraints $C1 - C3$ and $C5 - C6$. Otherwise, the task must be performed locally.
 - 7: update the best offloading cost $\mathcal{Z}$ and decisions $\mathcal{X}$;
 - 8: **end for**
-

4.2. Global Offloading Decision and Lagrangian Adjustment Heuristics

The outer level of the Lagrangian dual problem refers to the global offloading decision problem. It ensures a feasible offloading solution of the primal problem. As known, the optimal solution of the Lagrange dual requires an exhaustive search of all solutions' space and Lagrange multiplier

401 values which is a difficult task in general. Consequently, we need to adopt an
 402 alternative approach. In this work, we use the Subgradient-based heuristic
 403 proposed in [24]. This heuristic has three steps, first solve the subproblems,
 404 using our proposed heuristics, for the current value of λ . Then, we check if
 405 the obtained solution is feasible or not. If so, we use a Lagrangian Adjust-
 406 ment Heuristic (LAH) to get a feasible solution using local searches. The idea
 407 of LAH is to check if every offloading server respect the constraint $C4$. When
 408 an offloading server does not respect this constraint, LAH tries to migrate
 409 some tasks offloaded from this offloading server to other offloading servers
 410 that respects all constraints. Finally, we update the Lagrange multipliers as
 411 following:

$$\lambda_k(t+1) = \lambda_k(t) + \theta(t) * \left(\sum_m^M \left(\sum_n^{N_m} x_{m,n}^k * c_k \right) - F_k \right) \quad (24)$$

where $\theta(t)$ is the update step. In this work, we use Held and Karp formula [24]
 to update this step as following:

$$\theta(t) = \eta(t) * \frac{\mathcal{Z}^* - \mathcal{Z}(t)}{\sum_{k=1}^{K+1} \left(\sum_m^M \sum_n^{N_m} x_{m,n}^k * c_k - F_k \right)^2} \quad (25)$$

412 where $\eta(t)$ is a decreasing adaptation parameter $0 < \eta(0) \leq 2$, $\mathcal{Z}^*$ is the best
 413 obtained solution of the problem 20 and $\mathcal{Z}(t)$ refers to the current solution

414 of the Lagrangian Dual. We have:

$$\eta(t+1) = \begin{cases} \vartheta * \eta(t) & \text{if } \mathcal{Z}(t) \text{ did not increase} \\ \eta(t) & \text{otherwise} \end{cases} \quad (26)$$

415 As suggested in [24], we set the values of $\vartheta = 0.9$ and $\eta(0) = 2$. The
 416 master problem repeats these steps until the stop conditions, which are the
 417 maximum number of iterations It_{max} and the maximum error ε ($\theta(t) < \varepsilon$).

418 5. Performance Evaluation

419 5.1. Numerical Assessment

420 5.1.1. System Parameters

421 In this section, we evaluate the performance of our offloading policy by
 422 evaluating several performance metrics, i.e. the average number of offloaded
 423 tasks and the energy saving from the offloading under different settings. The
 424 MEC environment consists of a metropolitan area, which is composed of 20
 425 APs and four cloudlets already deployed among the network. In addition, two
 426 network topologies are considered. The ring topology, in which the cloudlets
 427 are equidistantly deployed in the AP, i.e: cloudlet 1 is collocated with the
 428 AP 1, cloudlets 2 with the AP 6, cloudlet 3 with the AP 11 and cloudlet
 429 4 with the AP 16. The second topology is generated by GT-ITM [25] tool,
 430 where the cloudlets are randomly deployed. In order to get a better un-
 431 derstanding of the offloading policies performances, we consider real mobile
 432 applications. Table 3 illustrates the characteristics of the used applications,

where γ indicates the computing resources required to perform the applications, up represents the data that must be transmitted to the remote server, dw the data that should be received from the remote server and t the maximum tolerated delay according to the QoS required by the application. The first three applications are static offloading decision tasks, and the remaining applications are dynamic offloading decision tasks [19].

Table 3: The characteristic of the real-world applications used for our tests.

Application	$\gamma_{(m,n)}$ (Giga CPU cycles)	$up_{(m,n)}$ (Kilobyte)	$dw_{(m,n)}$ (Byte)	$t_{(m,n)}$ (Second)
static offloading decision tasks				
FACE	12.3	62	60	5
SPEECH	15	243	50	5.1
OBJECT	44.6	73	50	13
dynamic offloading decision tasks				
Linpack	50	10240	120	62.5
CPUBENCH	3.36	80	80	4.21
PI BENCH	130	10240	200	163

In addition, we consider two cloudlets configurations, in which we use real-world setting, used by the public cloud provider such as: Amazon AWS and Microsoft azure [3, 4, 10], to simulate the behavior of ECESO real-world scenarios. In the first configuration, each cloudlet has a computing capacity of 1000 Giga cycles/s, and allocates 10 Giga cycles to perform every offloaded task ($F_k = 1000$ and $c_k = 10$). In the second configuration, we consider heterogeneous cloudlets, where cloudlets 1 and 2 have a computing capacity of 500 Giga cycles/s and cloudlets 3 and 4 have a computing capacity of 1500 Giga cycles/s. Both upload and download bandwidths of each AP are

448 set to 150 mbps and the bandwidth allocated to each user is estimated using
 449 the parameter settings used in Bianchi model [21]. Regarding the backhaul
 450 network, we use the parameters presented in [26]. Moreover, as in [8], we
 451 assume a homogeneous users distribution in the network. As in [22], we
 452 also consider that the power consumed in transmission mode is equal to the
 453 power consumed in the reception mode and is equal to ten times the power
 454 consumed in idle mode. We set $P_{m,n}^{idle}$ to 100 *mWatts*. The local computing
 455 capability of each user was randomly chosen from $F_{m,n} \in [0.8, 1, 1.2]$ Giga
 456 cycles/s. Finally, we consider that each user randomly chooses an application
 457 from those described in the table 3.

458 In order to evaluate the performances of our proposal, we propose to
 459 compare it with the following offloading policies:

- 460 • **DOTA** [10, 8]: In **DOTA**, each AP is associated with the nearest
 461 cloudlet. In this case, all users connected to this AP offload their tasks
 462 to the same cloudlet. When a cloudlet is overloaded, the tasks are
 463 migrated to the remote cloud.
- 464 • **CBL** [13, 7]: Using **CBL** we also associate each AP to the nearest
 465 cloudlet. Thus, all users connected to that AP have to offload their
 466 tasks into that same cloudlet. However, unlike **DOTA**, when the
 467 cloudlet is overloaded, the tasks are migrated to another cloudlet.
- 468 • **FCO** [2, 3]: In this policy, all users offload their tasks to the cloud.

469 In order to compare these offloading policies, we also define comparison
 470 metrics depicting the gain of a given offloading policy $\mathcal{P}$. This gain represents

the benefit of the offloading policy $\mathcal{P}$ compared to case where the task is offloaded to the cloud (i.e. **FCO** policy). We formulate the gain as following:

$$\text{gain of } \mathcal{P} = 100 * \frac{(\text{cost of FCO} - \text{cost of } \mathcal{P})}{\text{cost of FCO}}$$

5.1.2. Results Analysis

In Figure 2, we plot the gain of our policy (ECESO) compared to the gain of DOTA and CBL, when considering a network topology following the configuration 1 with homogeneous cloudlet characteristics. As expected, we can observe that the gain decreases when the number of users increases. This is mainly due to the fact that the backhaul cost increases when the number of offloaded task increases. We can also observe that the performances of ECESO, DOTA and CBL are almost equivalent, except when the number of users exceeds 300, where we notice that our approach is slightly better. This is due to the fact that ECESO tries to maximize the bandwidth allocated to each user and offload in priority the tasks with the greatest impact on the cost. Consequently, less tasks are offloading compared to the other offloading policies.

In Figure 3, we compare the performances of ECESO, BCL and DOTA when heterogeneous cloudlets are deployed, for both ring and GT-ITM topologies. As we can see, when few number of users are considered (< 100), the performances of the three policies are equivalent. However, when the number of users increases ECESO outperforms both BCL and DOTA, for both configurations of network topologies. This is due to the fact that when we increase the number of users both cloudlets 1 and 2 cannot support all the offloaded tasks. In this case, DOTA policy start to migrate the overloaded tasks to the

Figure 2: Offloading gain over homogeneous cloudlets configuration

494 remote cloud, which adds more offloading costs. On the other hand, CBL
 495 tries to migrate the overloaded tasks from cloudlets 1 and 2 to cloudlets 3
 496 and 4 and also some additional offloading cost but less than DOTA. How-
 497 ever, using ECESO, for each task, we can select the best cloudlet at the
 498 offloading decisions step, which reduce the additional offloading cost due to
 499 the migration of tasks introduced in DOTA and CBL. Finally, we notice that
 500 the effect of the network topology on the performances of ECESO is more
 501 important than the DOTA and CBL. This is due to the fact that ECESO
 502 uses the topology to select the cloudlets. However, DOTA and CBL assign
 503 statically the AP to the cloudlet.

Figure 3: Offloading gain over heterogeneous cloudlets.

Figure 4: Comparison of offloaded tasks to cloudlets and the remote cloud under different users and cloud computing capacity (c_{cloud} in Giga cycles/s).

504 Finally, in Figure 4 we investigate how each application behaves accord-
 505 ing to the amount of resources allocated to tasks on the cloudlet and on the
 506 cloud. Precisely, we fix the computing capacity allocated to each user in the
 507 cloudlets to 10 Giga cycles/s and we vary this capacity between 10 to 20
 508 Giga cycles/s in the cloud [3, 4, 10]. As we can see, not all the applica-
 509 tions have the same behavior according to the amount of resources allocated
 510 on the cloud. Basically, all **static offloading decision tasks** (FACE, SPEECH,
 511 and OBJECT) are always offloaded to the cloudlets or the cloud have the
 512 same computation computing 10 Giga cycles/s. However, where the remote
 513 cloud has greater computing capacity than cloudlets ($c_{cloud} > c_{cloudlets}$) all
 514 the tasks are offloaded to the remote cloud. Figure 4(b), 4(c) and 4(d) illus-
 515 trate the performances of the ECESO policy for *dynamic offloading decision*
 516 *tasks*, CPUBENCH, PIBENCH and Linpack, respectively. The ratio of the
 517 offloading tasks decreases when the number of users in the network increases,
 518 for example, for CPUBENCH applications 100% of tasks are offloaded where
 519 the number of users is not greater than 200, but only 30% are offloaded where
 520 1000 users are in the network. This decreasing of the ratio of offloaded tasks
 521 is due to the wireless bandwidth in the AP. We also note that the ratio of
 522 the offloaded tasks depends on the application characteristics when the ap-
 523 plications require a lot of computing resources and transmit a huge amount
 524 of data (Linpack and PIBENCH) the ratio of offloaded tasks is lower. As a
 525 result, choose the placement of the tasks, remote cloud or cloudlet, depends
 526 on many factors, in the figures we noticed that the computing resource allo-
 527 cation in the remote cloud, the computing resource allocation in the cloudlets
 528 and the characteristics of the application affect directly the placement of the

529 offloaded tasks.

530 5.2. Experimental Assessment

531 5.2.1. Testbed

532 In addition to the numerical results presented in the previous section,
533 we evaluated our proposal using real experiments. In this case, we setup a
534 testbed composed of several hardware components, as illustrated in figure 5.
535 The first component consists of the client device acting as a mobile terminal,
536 the second component is the offloading server located either at the edge Cloud
537 or the remote Cloud and finally the last component is a network topology to
538 connect the client device to the offloading server.

539 For the client device, even if our testbed can be used with real Android
540 mobile terminal, we decided to consider an Android client using a Raspberry
541 Pi 3 device. The main idea is to have a controlled experimental environment,
542 since we need to measure the energy consumed by the device and also to con-
543 trol the bandwidth for the client. We used a Raspberry Pi 3 Model B (RPi3)
544 powered by a Quad Core Broadcom BCM2837 64bit ARMv8 processor at
545 1.2 GHz and 1GB LPDDR2 of RAM at 900 MHz. On this client device, we
546 installed Android 8 "Oreo" as operating system and also our client offloading
547 middleware. Finally, we developed and implemented three **dynamic offloading**
548 **decision tasks** applications: Linpack, CPUBENCH and PI BENCH. Each of
549 these bench applications was implemented as a fragment within a common
550 Java Android application. The size of the APK source file is 5427.2 Kilobytes.

551 The second component of our testbed is the offloading server. As shown
552 in the figure 5, the offloading server can be located at the edge Cloud or at
553 the remote Cloud. For the edge Cloud, we used a desktop PC powered by an

Figure 5: An Overview Architect of Computation Offloading Platform

554 Intel I7-6700 8 cores CPU at 3.40 GHz and 16 GB DDR3 of RAM with an
555 Ubuntu 18.04.2 LTS as operating system. In order to run native Android
556 applications, we also installed, using VirtualBox, a virtual machine with
557 Android-x86 distribution as operating system. Android-x86 [27] is an open
558 source project to port Android on x86 platform. Finally, we also implemented
559 and installed on this distribution our server offloading middleware. For the
560 remote Cloud, we deploy our offloading server on AWS Amazon Cloud using
561 the `t2.medium` instance. This instance is powered by high-frequency Intel
562 Xeon processors with 2 vCPU and a memory of 4 GB. As for the edge Cloud,
563 we also installed our server offloading middleware on this virtual machine.

564 To connect the Android client device to the offloading server, we have set
565 up a network topology composed of two Cisco routers. The first router is
566 connected to the client via a switch (LAN) and the second router allows our
567 testbed to be connected to the edge Cloud through a local network and to
568 the remote Cloud via an Internet connection. To connect the two routers, we
569 use a serial link allowing us to control the speed at which the data, in bits
570 per second (bps), is sent between the two routers. The main objective is to
571 control the bandwidth between the offloading client and the offloading server.
572 In our experiments, we used the bandwidth values offered by the serial link
573 which are: *1.2Kbps*, *4.8Kbps*, *9.6Kbps*, *38.4Kbps*, *72Kbps*, *125Kbps*, *500Kbps*,
574 *5.3Mbps* and *8Mbps*.

575 We also implemented and deployed our offloading middleware. This mid-
576 dleware implementation is based on client-server architecture. On both the
577 client and the server, our middleware is integrated to Android operating sys-
578 tem as a new service which must be executed in the background by the An-

579 droid OS. Inspired by the offloading platforms from literature [15, 18, 17, 28],
580 our offloading middleware uses component-based design pattern [29]. For in-
581 stance, the client offloading middleware has three main functions: the *profiler*
582 *function*, the *offloading decider function* and the *communication function*.
583 The objective of the profiler is to collect the information related to the appli-
584 cation (i.e. task), such as the hardware usage and the network bandwidth.
585 This information is stored in a local database in order to be used by a second
586 function, which is the offloading decider function. The main objective of the
587 offloading decider function is to decide whether the task should be executed
588 locally or offloaded to the edge or to the remote Cloud. We implemented
589 our offloading policy within this function. Finally, the last function is the
590 communication function, which is in charge of handling the communications
591 between the client offloading middleware and the remote offloading middle-
592 ware. Basically, in case of offloading, this function sends both the APK source
593 code and the input data corresponding to the offloaded task to the offloading
594 server.

595 In addition to the client offloading middleware, we also developed a re-
596 mote offloading middleware. The main objective of this middleware is to
597 execute the task offloaded from the client, and also to return the results ob-
598 tained at the end of the execution. This remote offloading middleware is
599 also composed of three main functions: *task handler function*, *monitor func-*
600 *tion* and *communication function*. The main objective of the task handler
601 function is to load and execute the APK source code of each received task.
602 The task handler is also in charge of maintaining an isolated execution en-
603 vironment between the received tasks by executing each task in a separated

604 thread. At the end of the task execution the task handler gathers the results
605 and, using the communication function, sends it back to the client offloading
606 middleware. Finally, monitor function saves information (i.e. logs) related
607 to the task execution.

608 5.2.2. Discussion

609 The first purpose of the experiments was to characterize the benchmark
610 applications: Linpack, CPU-Bench and PI-Bench. Each of these applications
611 was parameterized in order to study the system under three CPU resource
612 requirement situations: 1) *Light*, 2) *Medium* and 3) *Full*. Precisely, Linpack
613 is a software that performs matrix calculations. To generate *light* processing
614 requirements, we parameterized Linpack with 17 square matrix sizes, ranging
615 from 1 to 17. *Medium* and *full* configurations were generated by multiplying
616 the above matrix sizes by a factor of 40 and 70, respectively. CPU-Bench is
617 a software that generates a random floating point and a random integer of a
618 predefined size n . To generate light, medium and full configurations, we set
619 $n = 10^6, 10^7$ and 10^8 , respectively. Finally, three CPU resource requirement
620 situations for Pi-Bench are obtained by computing π with an approximation
621 of $10^3, 10^4$ and 10^5 decimal places. All the shown results are average values
622 obtained after executing 10 runs with the same experiments setting.

623 Using our testbed, we measured the number of CPU cycles ($\gamma_{m,n}$), the
624 quantity of uploaded data ($up_{m,n}$) and the quantity of downloaded data
625 ($dw_{m,n}$) of the three bench applications, under *Light*, *Medium* and *Full* con-
626 figurations. The results are shown in Table 4. One can observe that all
627 the measured parameters are different from those reported from the liter-
628 ature [19] in table 3, except downloaded data for Linpack. New versions

629 and the possibility to execute these applications by setting some parameters
630 might explain such a gap. Notice also that due to our implementation of
631 these three bench applications as fragments within the same Java Android
632 application, the quantity of the downloaded data is always equal to the size
633 of the APK source file (5427.2 bytes). Among the three bench applications,
634 Linpack is the one that generates the lowest number of CPU cycles. Under
635 *Full* configuration, it is also the one that leads to the highest number of CPU
636 cycles.

Table 4: The characteristic of the bench apps

Application	$\gamma_{m,n}$ (Giga CPU cycles)	$up_{m,n}$ (Kilobyte)	$dw_{m,n}$ (Byte)
Linpack			
Light	0.2033	5427,2	120
Medium	547.406	5427,2	120
Full	2909.11	5427,2	120
PIBENCH			
Light	2.203	5427,2	56
Medium	45.539	5427,2	56
Full	1310.882	5427,2	56
CPUBENCH			
Light	8.696	5427,2	30
Medium	293.613	5427,2	30
Full	745.435	5427,2	30

637 Figure 6 shows the energy consumption measured in our testbed for Lin-
638 pack *easy*, when it is executed locally on the mobile terminal, on the edge
639 with 1 or 2 vCPU and on the clouds located at Paris or at the Ohio, both
640 allocating 2 vCPU. The x axis represents the bandwidth of the serial link

Figure 6: Linpack *easy*: Energy Consumption

641 that we have varied from 1.2 Kbps to 8 Mbps. Notice that logarithmic scale
 642 is used for x and y axis. As one can see, when the offered bandwidth is
 643 restricted to less than 500Kbps, the lowest energy consumption is obtained
 644 for local execution. Above 500kbps, offloading Linpack *easy* on the edge with
 645 2vCPU offers better energy performances. Starting from 5.3Mbps offloading
 646 on the cloud located at Paris outperforms the local execution. These exper-
 647 imental results prove that, even for applications that require light process-
 648 ing resources, offloading could be energetically beneficial when the available
 649 bandwidth to the remote server is sufficiently high.

650 Another situation fostering the remote execution is observed when the
 651 task's processing needs is important. This is clearly illustrated for Linpack
 652 *medium* in figure 7. We can notice a certain hierarchy in the performances:
 653 first the closest edge, then the remote cloud, and lastly the local execution.

Figure 7: Linpack *medium*: Energy Consumption

Figure 8: Round Trip Time between the terminal and different locations

654 A noticeable discrepancy can be observed in figure 7 between the clouds at
 655 Paris and at Ohio. We believe that this is related to differentiated network
 656 latency from our lab to these two clouds. The average Round Trip Time
 657 (RTT) measured from the terminal toward different remote servers is drawn
 658 in figure 8. We can see that the RTT for the edge and the cloud in Paris
 659 are very close to each other, while the RTT to the cloud in Ohio increases
 660 significantly for larger bandwidth values.

Figure 9: Linpack *full*: Energy Consumption

661 As shown in Figure 9, the observations made for Linpack *medium* are
 662 reinforced for the *full* variant. Because Linpack *full* is a CPU-intensive ap-
 663 plication, it is obvious that its completion time is mainly dominated by the
 664 computation duration. The transmission delay is proportionally negligible.
 665 This explains why the effect of differentiated RTT between Paris and Ohio
 666 is hardly observable in figure 9. However, the performance gap between the

edge and the clouds is very apparent and almost insensitive to the bandwidth, when the latter reaches $72Kbps$. We can deduce that the local edge offers higher CPU capacity than those provided at the clouds, despite the fact that both have allocated 2vCPU resources. This discrepancy illustrates the impact of the heterogeneity of the hardware and the software technologies, which are used in a multi-tier mobile edge computing infrastructure.

Figure 10: CPUBench *easy*: Energy Consumption

Energy consumption for CPU-Bench *easy*, *medium* and *full* are shown in figures 10, 11 and 12, respectively. The general observations regarding Linpack holds for CPUBench. However, for CPU-Bench *easy* one can see in figure 10 that when the bandwidth is restricted to less than $125Kbps$ the cloud in Paris and the local edge have almost the same performances, with a very slight advantage for the cloud at Paris. This can be explained by the fact that transmission duration over networks that exhibit important

Figure 11: CPUBench *medium*: Energy Consumption

Figure 12: CPUBench *full*: Energy Consumption

delays represent a significant proportion part with respect to the completion time of an offloaded light task. As shown in figure 8 when the bandwidth is restricted to less than $125Kbps$ the Round Trip Time from the terminal toward the local edge is elevated and quite similar to the RTT to the cloud at Paris.

Figures 13, 14 and 15 are relative to Pi-Bench. Compared to the previous bench applications one can notice for *easy* and *medium* cases that the edge with 2vCPU outperforms significantly the edge with 1vCPU. We believe that this is due to the multi-threaded nature of this application. The computation time is lower in 2vCPU compared to 1vCPU thanks to the ability of the former edge to process in parallel the computation of this multi-threaded application. The advantage of offloading such multithreaded application in a multi-CPU sever is clearly shown in figure 15 for the CPU-intensive case. The clouds in Paris and Ohio, where 2vCPU resources are allocated, outperform the 1vCPU edge.

Completion time for PiBench *easy*, *medium* and *full* are shown in Figures 16, 17 and 18, respectively. One can remark that this completion time curves have the same shape and follow the same hierarchy than those observed in energy consumption figures 14 and 15. Actually, in all our experiments we noticed that the consumed energy on the terminal is quasi-stationary during the execution time of an offloaded task. Consequently, the consumed energy is proportional to the completion time of an offloaded task. This observation holds for Linpack and CPU-Bench, as well¹. The corollary to this finding is that in case of offloading, the remote location that minimizes

¹To save space, we choose not to show the completion time for Linpack and CPU-Bench

Figure 13: PiBench *easy*: Energy Consumption

Figure 14: PiBench *medium*: Energy Consumption

Figure 15: PiBench *full*: Energy Consumption

704 the completion time is also the one that optimize the consumed energy. This
 705 statement has been validated in all the experiments that we have run, for the
 706 three bench applications, under different CPU and network configurations.

707 When a task is executed locally, the measured energy is also quasi-
 708 stationary during the processing time. However, as illustrated in figures 6,
 709 7, 9, 10, 11, 12 13, 14 and 15 the difference of consumed energy between
 710 local and remote execution is of several order of magnitude. Yet, in almost
 711 all the experiments that we have run, we noticed that the decision among
 712 local and offloading that minimize the completion time is also the one that
 713 optimize the consumed energy. The unique observed exception case to this
 714 rule is Pi-Bench *easy* when the bandwidth is restricted to 9.6Kbps. Com-
 715 paring figure 16 to 13, we can remark that when the bandwidth is restricted
 716 to 9.6Kbps the local execution minimize the completion time, while energy

consumption is minimized when Pi-Bench *easy* is offloaded on the 2vCPU
edge.

Figure 16: PiBench *easy*: completion time

From figures 6, 7, 9, 10, 11, 12 13, 14 and 15 we can easily identify the
locations that optimize the consumed energy for different applications and
network configurations. The right column in table 5, shows the location that
minimizes the consumed energy that we derived from experimental results
for the three bench applications under different processing loads and network
configurations. The left column indicates the optimal locations obtained
by our proposed framework ECESO. To fit with the applications that we
evaluated in our testbed, we applied the measured parameters indicated in
table 4 to ECESO. We can see that in the majority of the cases the theoretical
solution match with the placement derived from the experimentation results.
The few mismatches are indicated in red. For Linpack, the ECESO placement

Figure 17: PiBench *medium*: completion time

Figure 18: PiBench *full*: completion time

730 solution fits with the best placement observed *via* experiments in 22 cases
731 among 27 tested configurations. Thus compared to experiments, our proposal
732 have determined the optimal task execution location in 81.4% of the cases.
733 To quantify the under-performance of ECESO in terms of consumed energy,
734 we calculate the relative difference of the experimentally consumed energy
735 among: 1) the case where task’s computational location is computed by our
736 proposal ECESO (left column in table 5) 2) the case where optimal placement
737 solution is derived from experiments observations (right column in table 5).
738 Averaging over the 27 possible cases, the under-performance of ECESO in
739 terms of consumed energy with respect to an optimal placement derived from
740 experimental results, is limited to 1.962%.

741 The near-optimality of ECESO, is also confirmed for Pi-Bench and CPU-
742 Bench. For Pi-Bench, the optimal placement is achieved by ECESO in
743 85.18% of the studied cases, with an average under-performance of 2.57% of
744 consumed energy. For CPU-Bench optimal placement is obtained by ECESO
745 in 70.37% of the cases with an average under-performance of 3.76% of con-
746 sumed energy.

747 Finally, in the last set of experiments, we assessed the effect of multi-users
748 context on offloading performances. We incrementally varied the number of
749 users. However, due to the constraints on the available infrastructure nodes,
750 at the time when we ran our experiments, we fixed the allocated resources
751 both at the cloud and the edge to 1vCPU and we limited the number of
752 mobile terminals to ten. Yet, we believe that the general tendencies, which
753 we will comment hereafter, still hold for larger scale.

754 To save space, we also choose to discuss for the multi-users context case

Table 5: Offloading decisions for each application: ECESO *vs* Experiments

Bandwidth	ECESO			Experimentation		
	Easy	Medium	Full	Easy	Medium	Full
Linpack						
1 200	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Local</i>	<i>1vCPU</i>	<i>1vCPU</i>
4 800	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>
9 600	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>
38 400	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>
72 000	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Local</i>	<i>2vCPU</i>	<i>1vCPU</i>
125 000	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Local</i>	<i>2vCPU</i>	<i>1vCPU</i>
500 000	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>
5 300 000	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>
8 000 000	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>
PiBench						
1 200	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Local</i>	<i>1vCPU</i>	<i>2vCPU</i>
4 800	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Local</i>	<i>2vCPU</i>	<i>2vCPU</i>
9 600	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>
38 400	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>
72 000	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>
125 000	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>
500 000	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>
5 300 000	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>
8 000 000	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>
CPUBench						
1 200	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Local</i>	<i>Paris</i>	<i>2vCPU</i>
4 800	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Paris</i>	<i>2vCPU</i>
9 600	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Paris</i>	<i>2vCPU</i>	<i>2vCPU</i>
38 400	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Paris</i>	<i>1vCPU</i>	<i>1vCPU</i>
72 000	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>Paris</i>	<i>2vCPU</i>	<i>2vCPU</i>
125 000	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>1vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>
500 000	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>
5 300 000	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>
8 000 000	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>	<i>2vCPU</i>

only the results of CPUBench. Our choice is motivated by the fact that following the one-user experiments results (see table 5)), CPUBench is the only application in our benchmark for which we observed three offloading decisions: local, edge and cloud.

Figure 19: Multi-users completion time of CPUBench *easy* in the cloud

The completion time of CPU-Bench when it is offloaded on the cloud is represented in figures 19, 20, and 21, for *easy*, *medium* and *full* mode, respectively. Each curve in those figures is associated to an access bandwidth, between 1.2 kbps to 8Mbps. The x -axis represents the number of offloading users, while the y -axis represents in logarithmic scale the completion time in milliseconds. One can see that the completion time increases significantly with the number of users, especially for bandwidth capacities less than 500kbps. The contention among several users on the access network increases the transmission delay, especially when the bandwidth capacity is

Figure 20: Multi-users completion time of CPUBench *medium* in the cloud

Figure 21: Multi-users completion time of CPUBench *full* in the cloud

quite low. This effect is observed even for CPU-Bench in *full*, despite the fact the processing delay *versus* transmission delay is quite important for this mode. Compared to offloading on the edge, which is shown in figures 22 23 and 24, one can see that the completion time in the edge increases slightly with the number of users, but the slope is much lower than for the cloud.

Figure 22: Multi-users completion time of CPU-Bench *easy* in 1vCPU edge

Completion time of 10 concurrent CPU-Bench users with local, edge and cloud execution locations is shown in figures 25, 26, and 27, for *easy*, *medium* and *full* mode, respectively. The *x*-axis represents in logarithmic scale the access bandwidth, which varies between 1.2 Kbps to 8Mbps, while the *y*-axis represents in logarithmic scale the completion time in milliseconds.

As shown in figure 25, local execution is the best placement for CPU-Bench in *easy* mode when bandwidth capacity is extremely constrained: less than 9,6Kbps. Indeed, in such condition, the transmission delay is very high

Figure 23: Multi-users completion time of CPU-Bench *medium* in 1vCPU edge

Figure 24: Multi-users completion time of CPU-Bench *full* in 1vCPU edge

781 and is proportionally much important than the processing delay, even when
 782 the application is executed in the edge or the cloud. Faster execution in the
 783 edge or in the cloud can not compensate the large transmission time when
 784 the access bandwidth is extremely low and shared among many (10) users.
 785 For such case, it is not worth it to offload and it is better to execute the task
 786 locally on the mobile terminal. For bandwidth capacity larger than 10Kbps,
 787 the experiments results show that the best decision is to offload on the edge.
 788 This result is interesting because it indicates that even when 10 users are
 789 contending on a quite limited bandwidth capacity of about 10Kbps, it still
 790 worth it to offload a computational intensive task such as CPU-Bench to
 791 the edge. Figures 25, 26, and 27 show that in almost all the cases the best
 792 offloading location is the edge. The only exception is for *easy* mode with a
 793 bandwidth access set to 5.3Mbps. As shown in figure 8, for higher bandwidth
 794 capacity (larger than few Mbps) the transmission delay to the edge is almost
 795 similar to the cloud. The difference is less than 10 milliseconds. In these
 796 cases, the completion time is almost the same on the edge and on the cloud,
 797 especially when the computational resources required by a task are low, which
 798 is the case for CPU-Bench in *easy* mode.

799 In table 6, we compare the best offloading decision of CPU-Bench with 10
 800 users, which we observed through experiments (figures 25, 26, and 27) to the
 801 decision obtained by our proposed algorithm ECESO. Each line is associated
 802 to a given bandwidth capacity at the access network. Thus combining with
 803 the three modes of CPU Bench, we have in total 27 cases. Table 6 show that
 804 the decision of ECESO matches with the best placement observed through
 805 experiments in 66.7% of the cases. The mismatches are due to the conser-

Figure 25: 10 users completion time of CPU-Bench *easy*

Figure 26: 10 users completion time of CPU-Bench *medium*

Figure 27: 10 users completion time of CPU-Bench *full*

806 vative nature of our algorithm. For example, for extremely low bandwidth
807 capacity (low than 3.2 Kbps) ECESO recommends to execute the task locally
808 rather than offload it to the edge. Similarly, for *easy* mode with 5.3 Mbps
809 bandwidth access case that we discussed above, our algorithm suggests to
810 offload the task on the edge, while experiments have shown that offloading
811 on the cloud can also minimize the completion time. The conservative nature
812 of ECESO is mainly inherent to our modelling, which induces some assump-
813 tions that appears to be conservative compared to real system behaviors, as
814 observed in experiments.

815 6. Conclusion

816 In this paper, we propose a new computation offloading policy in two-
817 tier MEC environment. The proposed offloading policy decides which users

Table 6: Offloading decision for CPUBench with 10 users: ECESO *vs* Experiments

Bandwidth	ECESO			Experimentation		
	Easy	Medium	Full	Easy	Medium	Full
1 200	<i>Local</i>	<i>Local</i>	<i>Local</i>	<i>Local</i>	<i>1vCPU</i>	<i>1vCPU</i>
4 800	<i>Local</i>	<i>Local</i>	<i>Local</i>	<i>Local</i>	<i>1vCPU</i>	<i>1vCPU</i>
9 600	<i>Local</i>	<i>Local</i>	<i>Local</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>
38 400	<i>Local</i>	<i>Local</i>	<i>Local</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>
72 000	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>
125 000	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>
500 000	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>
5 300 000	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>Paris</i>	<i>1vCPU</i>	<i>1vCPU</i>
8 000 000	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>	<i>1vCPU</i>

818 should offload, to which offloading server and allocated the wireless band-
 819 width accordingly. First, we formulate the problem as a Non-Linear Bi-
 820 nary Integer Program (NLBIP). Then, we propose an efficient heuristic to
 821 solve the problem using Lagrangian decomposition approach. The proposed
 822 heuristic uses a branching algorithm to maximize the bandwidth allocation
 823 and minimize the energy consumption. The numerical results show perfor-
 824 mance improvements in terms of the energy consumption compared to exist-
 825 ing offloading policies under different scenarios. Finally, we implemented our
 826 offloading policy on a real testbed. Using this testbed, we were able to eval-
 827 uate our offloading decision policy for three real Android OS applications,
 828 with different traffic patterns, resource demands and multi-users context.

829 References

- 830 [1] H. Mazouzi, N. Achir, K. Boussetta, Maximizing mobiles energy sav-
 831 ing through tasks optimal offloading placement in two-tier cloud, in:
 832 Proceedings of the 21st ACM International Conference on Modeling,

- 833 Analysis and Simulation of Wireless and Mobile Systems, MSWIM '18,
834 ACM, New York, NY, USA, 2018, pp. 137–145.
- 835 [2] M.-H. Chen, B. Liang, M. Dong, Joint offloading decision and resource
836 allocation for multi-user multi-task mobile cloud, in: 2016 IEEE Inter-
837 national Conference on Communications (ICC), IEEE, 2016, pp. 1–6.
- 838 [3] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation of-
839 floading for mobile-edge cloud computing, IEEE/ACM Transactions on
840 Networking 24 (5) (2016) 2795–2808.
- 841 [4] S. Guo, B. Xiao, Y. Yang, Y. Yang, Energy-efficient dynamic offloading
842 and resource scheduling in mobile cloud computing, in: 35th Annual
843 IEEE International Conference on Computer Communications, INFO-
844 COM 2016, San Francisco, CA, USA, April 10-14, 2016, Vol. 2016-July,
845 IEEE, 2016, pp. 1–9.
- 846 [5] S. Guo, J. Liu, Y. Yang, B. Xiao, Z. Li, Energy-efficient dynamic compu-
847 tation offloading and cooperative task scheduling in mobile cloud com-
848 puting, IEEE Transactions on Mobile Computing 18 (2) (2019) 319–333.
- 849 [6] K. Gai, M. Qiu, H. Zhao, Energy-aware task assignment for mobile
850 cyber-enabled applications in heterogeneous cloud computing, Journal
851 of Parallel and Distributed Computing 111 (2018) 126–135.
- 852 [7] M. Jia, J. Cao, W. Liang, Optimal cloudlet placement and user to
853 cloudlet allocation in wireless metropolitan area networks, IEEE Trans-
854 actions on Cloud Computing PP (99).

- 855 [8] Z. Xu, W. Liang, W. Xu, M. Jia, S. Guo, Efficient algorithms for ca-
856 pacitated cloudlet placements, *IEEE Transactions on Parallel and Dis-*
857 *tributed Systems* 27 (10) (2016) 2866–2880.
- 858 [9] H. Yao, C. Bai, M. Xiong, D. Zeng, Z. Fu, Heterogeneous cloudlet de-
859 ployment and user-cloudlet association toward cost effective fog comput-
860 ing, *Concurrency and Computation: Practice and Experience* 29 (16).
- 861 [10] L. Ma, J. Wu, L. Chen, Dota: Delay bounded optimal cloudlet de-
862 ployment and user association in wmans, in: *Proceedings of the 17th*
863 *IEEE/ACM International Symposium on Cluster, Cloud and Grid Com-*
864 *puting*, IEEE Press, 2017, pp. 196–203.
- 865 [11] A. Mukherjee, D. De, D. G. Roy, A power and latency aware cloudlet
866 selection strategy for multi-cloudlet environment, *IEEE Transactions on*
867 *Cloud Computing* PP (99) (2016) 1–14.
- 868 [12] D. G. Roy, D. De, A. Mukherjee, R. Buyya, Application-aware cloudlet
869 selection for computation offloading in multi-cloudlet environment, *The*
870 *Journal of Supercomputing* (2016) 1–19.
- 871 [13] M. Jia, W. Liang, Z. Xu, M. Huang, Cloudlet load balancing in wire-
872 less metropolitan area networks, in: *Computer Communications, IEEE*
873 *INFOCOM 2016-The 35th Annual IEEE International Conference on*,
874 Vol. 2016-July, 2016, pp. 1–9.
- 875 [14] H. Mazouzi, N. Achir, K. Boussetta, Dm2-ecop: An efficient compu-
876 tation offloading policy for multi-user multi-cloudlet mobile edge com-

- 877 puting environment, *ACM Trans. Internet Technol.* 19 (2) (2019) 24:1–
878 24:24.
- 879 [15] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu,
880 R. Chandra, P. Bahl, Maui: making smartphones last longer with code
881 offload, in: *Proceedings of the 8th International Conference on Mobile*
882 *Systems, Applications, and Services (MobiSys 2010)*, San Francisco, Cal-
883 ifornia, USA, June 15-18, 2010, ACM, pp. 49–62.
- 884 [16] R. C. Martin, *Agile software development: principles, patterns, and*
885 *practices*, Prentice Hall, 2002.
- 886 [17] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, Clonecloud: elastic
887 execution between mobile device and cloud, in: *European Conference*
888 *on Computer Systems, Proceedings of the Sixth European conference on*
889 *Computer systems, EuroSys 2011*, Salzburg, Austria, April 10-13, 2011,
890 ACM, pp. 301–314.
- 891 [18] J. I. Benedetto, G. Valenzuela, P. Sanabria, A. Neyem, J. Navon,
892 C. Poellabauer, Mobicop: A scalable and reliable mobile code offloading
893 solution, *Wireless Communications and Mobile Computing* 2018.
- 894 [19] Y. Gao, W. Hu, K. Ha, B. Amos, P. Pillai, M. Satyanarayanan, Are
895 cloudlets necessary?, *School Comput. Sci.*, Carnegie Mellon Univ., Pitts-
896 burgh, PA, USA, Tech. Rep. CMU-CS-15-139.
- 897 [20] I. Tinnirello, G. Bianchi, Y. Xiao, Refinements on ieee 802.11 distributed
898 coordination function modeling approaches, *IEEE Transactions on Ve-*
899 *hicular Technology* 59 (3) (2010) 1055–1067.

- 900 [21] G. Bianchi, Performance analysis of the iee 802.11 distributed coordi-
901 nation function, *IEEE Journal on selected areas in communications*
902 18 (3) (2000) 535–547.
- 903 [22] A. Carroll, G. Heiser, et al., An analysis of power consumption in a
904 smartphone., in: *USENIX annual technical conference*, Vol. 14, Boston,
905 MA, 2010, pp. 21–21.
- 906 [23] S. O. Krumke, C. Thielen, The generalized assignment problem with
907 minimum quantities, *European Journal of Operational Research* 228 (1)
908 (2013) 46–55.
- 909 [24] J. Tang, C. Yan, X. Wang, C. Zeng, Using lagrangian relaxation de-
910 composition with heuristic to integrate the decisions of cell formation
911 and parts scheduling considering intercell moves, *IEEE Transactions on*
912 *Automation Science and Engineering* 11 (4) (2014) 1110–1121.
- 913 [25] M. Thomas, E. W. Zegura, Generation and analysis of random graphs to
914 model internetworks, Tech. rep., Georgia Institute of Technology (1994).
- 915 [26] J. Oueis, E. Calvanese-Strinati, A. De Domenico, S. Barbarossa, On
916 the impact of backhaul network on distributed cloud computing, in:
917 *Wireless Communications and Networking Conference Workshops (WC-*
918 *NCW)*, 2014 IEEE, IEEE, 2014, pp. 12–17.
- 919 [27] OSBoxes, [Android x86](https://www.osboxes.org/android-x86/).
920 URL <https://www.osboxes.org/android-x86/>
- 921 [28] L. López, F. J. Nieto, T.-H. Velivassaki, S. Kosta, C.-H. Hong, R. Mon-
922 tella, I. Mavroidis, C. Fernández, Heterogeneous secure multi-level re-

- 923 mote acceleration service for low-power integrated systems and devices,
924 Procedia Computer Science 97 (2016) 118–121.
- 925 [29] F. Foukalas, Y. Ntarladimas, A. Glentis, Z. Boufidis, Protocol reconfigu-
926 ration using component-based design, in: IFIP International Conference
927 on Distributed Applications and Interoperable Systems, Springer, 2005,
928 pp. 148–156.