Cloud and Edge Computation Offloading for Latency Limited Services

IVANA KOVACEVIC1, (Member, IEEE), ERKKI HARJULA1, (Member, IEEE), SAVO GLISIC2, (Senior Member, IEEE), BEATRIZ LORENZO3, (Member, IEEE), AND MIKA YLIANTTILA1, (Senior Member, IEEE)

1Centre for Wireless Communications, University of Oulu, 90014 Oulu, Finland
2Worcester Polytechnic Institute, Worcester, MA 01609, USA
3Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, USA

Corresponding author: Ivana Kovacevic (ivana.kovacevic@oulu.fi)

This work was supported in part by the Academy of Finland 5GEAR project, in part by the Academy of Finland 6Genesis Flagship under Grant 318927, in part by the Academy of Finland Digihealth project, and in part by the NSF SII National Center for Wireless Spectrum Research under Grant NSF 2037782.

ABSTRACT Multi-access Edge Computing (MEC) is recognised as a solution in future networks to offload computation and data storage from mobile and IoT devices to the servers at the edge of mobile networks. It reduces the network traffic and service latency compared to passing all data to cloud data centers while offering greater processing power than handling tasks locally at terminals. Since MEC servers are scattered throughout the radio access network, their computation capacities are modest in comparison to large cloud data centers. Therefore, offloading decision between MEC and cloud server should minimize the usage of the resources while maximizing the number of accepted delay critical requests. In this work we formulate the joint optimization of communication and computation resources allocation for computation offloading (CO) requests with strict latency constraints. We show that the global optimization problem is NP-hard and propose an efficient heuristic solution based on the single user optimal solution. Simulation results are presented to show the effectiveness of the proposed algorithm, compared to optimal and baseline solution where tasks are allocated in the order of arrival, with different system parameters. They show that our algorithm performs close to the optimal in terms of resource utilization and outperforms the baseline algorithm in terms of acceptance rate.

INDEX TERMS Cloud computing, multi access edge computing (MEC), computational offloading (CO), end-to-end latency, limited-latency services, joint resource allocation.

I. INTRODUCTION

Emerging applications, such as augmented and virtual reality, face recognition and language processing, are becoming more computationally demanding. At the same time, upcoming wearables such as AR (augmented reality) glasses, various IoT devices such as medical instrumentation, implants and sensors have limitations in terms of available energy from batteries and computational capacities. While CO is recognised as a potential solution to this problem, the use of existing distant cloud services is becoming impractical especially for the upcoming delay critical applications [1], [2]. Multi-access Edge Computing (MEC) concept is aiming at bringing external computational resources closer to the users. European Telecommunications Standards Institute (ETSI) has introduced MEC as part of forthcoming 5G networks [3]. MEC servers are located in the access network, which significantly reduces communication delay [4]. However their relatively dense deployment and cost-effectiveness imply the limitations in physical size and available computational capacity [1]. Nevertheless, MEC servers are not solely beneficial for the delay critical services. Offloading any type of tasks, including latency tolerant tasks, to MEC instead of Cloud servers has potential to improve security by keeping the data within the bounds of mobile operator’s network [4] and decreases the amount of traffic in the core network, which in turn decreases overall use of computational and networking resources and energy on the system level.

The conflicting demands of offloading as much computational tasks as possible to nearby capacity-constrained MEC servers, while securing enough resources for the
delay-critical applications that cannot be offloaded to distant cloud, require sophisticated decision-making algorithms. In this paper, we propose joint allocation of wireless (communication) and computational resources for MEC and Cloud server offloading with strict delay requirements for multiple computational tasks. We refer to computational infrastructure where tasks are dynamically assigned to MEC or Cloud server as two-tier infrastructure [5].

Delay-critical applications have different requirements in terms of data throughput, latency and reliability, and the need for further classification of low latency traffic is recognised. Furthermore, it should be noted that even the definition of communication latency itself is not unique, but instead it depends on the use cases [6]. Latency requirements can be expressed by stochastic measures, such as that the expected value and the variance of the latency must remain under a predefined threshold [7], [8], or that the latency should be under a certain threshold with certain reliability. In the literature, the former case is also referred to as probabilistic latency while the latter as deterministic latency [9]. In this paper, we consider the deterministic latency definition and focus on the computational tasks that have strict requirement to be completed within given time period. This definition is appropriate for use cases such as VR (virtual reality), AR, and real-time control, where, for example, images have to be processed before human eye can detect the lagging. Number of emerging IoT applications is expected to fall into this category of computational offloading. Delay critical and high reliability application are significant for development of telemedicine services. For example remote monitoring of the patients through a number of wearable sensors and implants require reliable and efficient external data processing.

In summary, our paper presents the following major contributions:

- Global optimal resource allocation algorithm for two-tier CO that jointly minimizes computation and communication resource usage, while maximizing the number of accepted tasks and meeting the strict latency requirements in the two-tier computational offloading architecture (2TCOA)
- Efficient heuristic solution for dynamic CO resource allocation decision depending on the instantaneous network conditions and computational demands, including the trade-off between selection of low-latency MEC and high capacity cloud server. Heuristic resource allocation (HRA) algorithm, based on the single task optimal allocation is designed to emulate the optimal allocation.
- Extensive simulation results that demonstrate the performance of our proposed scheme.

The rest of the paper is organized as follows: Section II presents related works on CO to multi-access edge and cloud servers. Section III provides a general system model and formulates the problem of optimal resource allocation for 2TCOA for latency limited traffic. In order to design the efficient heuristic solution we analyse the performance of the optimal algorithm in Section IV. Heuristic solution is introduced in Section V. Section VI provides simulation results illustrating the performance of our schemes. Finally, Section VI concludes the paper.

II. RELATED WORK

Resource allocation for cloud and edge CO is extensively studied in the literature. Decision criteria whether to offload the computational task or perform it locally at the end-user device is considered in [10]–[13]. Work in [10] proposes joint physical resource block and MEC computation resources allocation with consideration of interference, in order to minimize the overall consumption of the entire system in terms of time and energy. Since this problem is non-convex, they propose a solution based on graph coloring. Similarly in [11], a multi-user CO problem for MEC in a multi-channel wireless interference environment is studied by using game-theory approach. Here, a potential game is used to determine the bandwidth assignment, while cloud computing capability is determined according to the subscribed contract. MEC servers are assumed to have unlimited capacity which is more appropriate for cloud servers. Another game-theoretic approach assumes that users are not rational but have subjective perceptions under uncertain wireless environment [13]. They formulate users’ decision-making of whether to offload or not as a prospect theory based non-cooperative game. The optimal offloading probability and transit power that minimize delay and energy consumption are calculated in [12] using Interior Point Method to solve non-linear optimization problem. The energy delay tradeoff for MEC offloading in [14] is set as mixed integer programming problem and solved by convex approximation.

Resource allocation for cloudlet offloading, a similar problem to MEC offloading, is studied in [15]–[17]. Cloudlet technology has been introduced to deploy mobile cloud services at the network edge. Cloudlet is a server that has direct wireless access such as WiFi. In a computation-intensive environment, Cloudlets can efficiently process the computationally intensive tasks. Authors in [15] propose heuristic solution for computational latency minimization for cloudlet without considering communication delay. Integer Programming formulation that minimizes the total cost of providing services, while taking into account probability of resource availability, is presented in [16]. The minimization of delay in a multi-cloudlet system is studied in [17]. They solve two mixed Integer Linear Programming (ILP) problems, to first select the appropriate cloudlet, and then to allocate the resources. Nowadays, Cloudlets are considered inadequate due to the limited wireless coverage [18]. On the other hand, MEC has superior offloading techniques provided by the mobile network with low-latency and high-bandwidth [18], wide coverage and better security.

Majority of studies on CO considers latency minimization without strict latency constraints. Work in [19] proposes the joint optimization of the radio resources and the computational resources in MIMO multicell system in order to
minimize the overall users’ energy consumption, while meeting strict latency constraints. While this work focuses on one-tier problem, i.e. offloading to MEC, our work studies two-tier offloading architecture, namely offloading to MEC and cloud server.

Several works study MEC co-operation [20]–[22]. The co-operation of mobile cloud providers, in terms of resource sharing, is studied in [20]. In [21], the authors propose collaborative task offloading. Price based dynamic resource management for co-operation between cloudlets is presented in [22].

A few papers, such as [23]–[25] and [26], consider two-tier offloading architecture. In these works, learning algorithms are utilized to allocate resources. Semi-Markov decision process criterion is used in [23], where the optimization problem is solved using linear programming. They oversimplify the two-tier problem with an assumption that offloading to distant cloud only occurs when there is no cloudlet coverage. In addition no explicit delay constraints are considered. Two virtual machine allocation methods based on semi-Markov decision process are proposed in [24] to balance the tradeoff between the high cost of providing services by the remote cloud and the limited computing capacity of the local fog. Model-based planning method and model-free reinforcement learning (RL) method are used to allocate virtual machines. The authors distinguish the high and low-priority services. Only high-priority services can access the cloud while low priority services can be accepted if there is free space in the fog. Delay constraints or bandwidth allocation are not considered. Reinforcement learning solution is proposed in [25] for selecting appropriate collaborative edge servers and allocating corresponding portion of the computing task to individual edge servers as well as the radio bandwidth resource. A simple scheme is adopted that offloads task to the cloud only when edge servers are occupied. They also minimize the average service latency, and although maximum tolerated latency is introduced as a constraint. This problem is different compared to ours, where over-provisioning of resources is not possible since strict latency requirement is met for each task in order to maximize the number of accepted tasks. The problem of two-tier offloading is addressed in these works by setting predefined policies. In our work, the decision whether to execute a task at a MEC server or a cloud data-center is dynamic, and depends on the instantaneous network conditions and computational demands.

Finally, a few recent works consider resource allocation for dynamic two-tier computational offloading with explicit latency constraints. In [26], a heuristic algorithm that jointly allocates wireless bandwidth and computational resources to mobile devices, is proposed to minimize the energy consumption of the system. In contrast, our work aims to minimize the total use of overall resources which is equivalent to maximizing the number of performed tasks. Furthermore, joint optimization of two-tier computation offloading decisions and computation resource allocations for vehicular network is considered in [27]. In contrast to our work they do not consider the transmission rate allocation.

III. SYSTEM MODEL

A. COMPUTATIONAL OFFLOADING ARCHITECTURE

We consider a two-tier computation offloading model presented in Fig. 1. A number of users (UE), connected to base stations, have access to external computation servers for offloading computational tasks. Each Base Station (BS) is connected to a MEC server, which consists of a number of interconnected physical machines. The placement of MEC servers is flexible, i.e., a server can directly connect to a BS via high speed fibers, or to an edge switch so that multiple BSs can share the computing resources of the same MEC server. MEC servers are placed in the access network, and while the communication latency between BS and MEC is low, the computational capacity of MEC is typically limited when compared to a cloud server. Computational requests can also be forwarded from BS through a core network to a distant cloud server. The communication latency between the UE and cloud servers is typically high, while the capacity of a cloud server is typically also high and easily scalable. Therefore, for the purpose of our analysis, its capacity can be considered unlimited.

We consider a MEC server $M$ is directly connected to a base stations $B$ as shown in Fig. 2. A set of UEs connected to the BS $B$, generate $\mathcal{N} = \{1, \ldots, n, \ldots, N\}$ limited latency tasks. The available bandwidth at the moment of the allocation, i.e. transmission rate in packets per second, is $R$. The minimum possible bandwidth to be assigned to one UE is $R_{\text{min}}$. Each UE $n$ has a computational request with the profile $Q^n(D^n, J^n, p^n_1, p^n_2)$ where $D^n$ is the maximum allowed delay, which accounts for total communication and computational delay of task execution. In order to complete a computational task $n$, $J^n$ number of instructions, i.e. CPU cycles, have to be executed. We denote with $p^n_1$ and $p^n_2$ the number of packets that have to be transmitted in order to forward the task to the server, and the computation result back to the UE, respectively. Wireless bandwidth is typically scarce compared to the optic links capacities, so here we focus on the wireless (communication) resources allocation. We assume that the transmission rates per session on the optical links are constant and known at the time of task
allocation. Limited latency communication is intolerant to queuing delays, and therefore network slices handling this type of traffic have to provide constant rates on each link of the route. For task \( n \) the delay between BS \( B \) and MEC \( M \) is denoted as \( d_{nB,M}^p \), while the delay between MEC server and cloud server is denoted as \( d_{nM,C}^p \), where \( d_{nB,M}^p \ll d_{nM,C}^p \). The total computational capacity of a MEC server available in the moment of allocation, in terms of CPU cycles per second, is \( F \). The computational rate assigned to a task \( n \) at a MEC server and cloud server is \( f_{nM}^p \) and \( f_{nC}^p \) CPU cycles per second, respectively. A list of notations used throughout the paper is given in Table 1.

### Table 1. Table of notations.

| Symbol | Meaning | Symbol | Meaning |
|--------|---------|--------|---------|
| \( N, n \) | Set and index of tasks | \( N \) | Total number of tasks |
| \( R \) | Available bandwidth | \( R_{\text{min}} \) | minimum bandwidth per task |
| \( Q^n \) | Profile of task \( n \) | \( D^n \) | maximum delay for task \( n \) |
| \( f^n \) | Number of instructions of task \( n \) | \( n \) | Number of packets of task \( n \) to receive |
| \( p^n \_B \) | Number of packets of task \( n \) to send | \( p^n \_C \) | Number of packets of task \( n \) to receive |
| \( d_{B,M}^p \) | Delay between BS and MEC for \( n \) | \( d_{M,C}^p \) | Delay between MEC and Cloud for \( n \) |
| \( F \) | Computational capacity of MEC | \( f_{M}^{*} \) | Computational rate of task \( n \) at MEC |
| \( f_{C}^{*} \) | Computational rate of task \( n \) at Cloud | \( r^n \) | allocated bandwidth to task \( n \) |
| \( D^n_{M} \) | Delay of task \( n \) at MEC | \( D^n_{C} \) | Delay of task \( n \) at Cloud |
| \( N_{\text{max}} \) | Max number of requests for B | \( a^n \) | allocation indicator and its set |
| \( f_M \) | Vector of computational rates at MEC | \( f_C \) | Vector of computational rates at Cloud |
| \( r \) | Vector of transmission rates | \( r_f \) | Vector of computational rates at Cloud |
| \( A_M \) | Number of tasks offloaded to MEC | \( f_M^{*} \) | Single task optimal computation rate at MEC |
| \( f_C^{*} \) | Single task optimal computation rate at Cloud | \( r_f^{*} \) | Single task optimal transmission rate at MEC |
| \( L_M \) | Number of links between BS and MEC | \( L_C \) | Number of links between MEC and Cloud |
| \( P_R \) | Price of the wireless resources | \( P_M, P_C \) | Price of computational resources at MEC and Cloud |

\( d_{n1}^{B,M} = L_M/r_s + (p^n_1 - 1)/r_s \) [28]. Similarly, relaying the computational result back to the UE needs \( d_{n2}^{B,M} = L_M/r_s + (p^n_2 - 1)/r_s \), and thus the total delay between BS and MEC for relaying the task \( n \) is \( d_{n}^{B,M} = (2L_M + p^n_1 + p^n_2 - 2)/r_s \). The execution time of a task \( n \) is \( J^n / f_M^n \) and the total delay of offloading a task \( n \) to MEC has to be less than the delay threshold:

\[
D^n_{M} = \frac{p^n_1 + p^n_2}{f_M^n} + \frac{J^n}{f_M^n} + d_{n}^{B,M} \leq D^n
\]  

The delay of a task offloaded to the cloud server can be calculated in a similar way taking into account additional delay between the MEC and Cloud server \( d_{n}^{M,C} = (2L_C + p^n_1 + p^n_2 - 2)/r_s \) where \( L_C \) is the number of links/hops on the route between MEC and Cloud server. The condition for a delay guarantee in the Cloud can be expressed as:

\[
D^n_{C} = \frac{p^n_1 + p^n_2}{f_C^n} + d_{n}^{M,B} + d_{n}^{M,C} + \frac{J^n}{f_C^n} \leq D^n
\]

### B. LIMITED LATENCY COMPUTATION OFFLOADING

We assume that the bandwidth allocated to UE with task \( n \) at BS, \( B \), provides the rate of \( r^n \) packets per second. The time necessary to offload a task to the BS is \( p^n_1 / r^n \), while relaying the computational result back to UE takes \( p^n_2 / r^n \).

We assume that the transmission rate of the limited-latency slice relaying the task between the BS and MEC server is \( r_s \) packets per second. The delay of sending a total of \( p^n_1 \) packets over \( L_M \) links between BS and MEC is
period, delay requirements $D^n$ of these tasks have to be reduced for the amount of time the task is waiting to be allocated.

The overall usage of the network resources is minimized if the following two criteria are met: 1) there is no overprovisioning of communication and computational resources for any task $n$, i.e. $D^n_M = D^n$ or $D^n_C = D^n$ and 2) the execution of a task at MEC is favoured over forwarding it to the cloud. This reduces the load in the backbone network and reduces the demand for processing power, since time frame for execution is not decreased due to network delay. Therefore, the objective of the global resource allocation optimization problem is to minimize the amount of the allocated communication bandwidth and computational rate at MEC and cloud servers. We define an indicator $a^n = 0$, if request $n$ is offloaded to MEC, and $a^n = 1$ if it is offloaded to cloud server. The vector of these indicators is denoted as $a$. The vectors of the variables related to assigned computational rates to MEC and cloud server are denoted by $\{M\}$ and $\{C\}$, respectively, while $\nabla$ is the vector of assigned transmission rates. The price of using $r$ units of the rate is $P(r)$, while prices of computational resources at MEC and Cloud are $P_M(f)$ and $P_C(f)$, respectively. The global optimization problem for CO decision and resource allocation can be expressed as:

$$\begin{align*}
    & \text{minimize} & \sum_{n=1}^{N} P_M(f^n_M) + P_C(f^n_C) + P_r(r^n) \\
    & \text{subject to} & (1-a^n)D^n_M + a^n D^n_C \leq D^n & n \in N \\
    & & \sum_{n=1}^{N} f^n_M \leq F \\
    & & \sum_{n=1}^{N} r^n \leq R \\
    & & r^n \geq R_{min} & n \in N \\
    & & a^n \in \{0, 1\} & n \in N \\
    & & 0 \leq \{M\} \leq \{C\} \leq \infty
\end{align*}$$

(3)

The execution of a task at the Cloud always requires more resources than the execution at the MEC server, i.e. $f^n_M + r^n_M < f^n_C + r^n_B$ where $r^n = r^n_M$ in order to satisfy (1) and $r^n = r^n_C$ in order to satisfy (2). Therefore, the objective function aims at maximizing number of tasks assigned to the MEC server. The first constraint guarantees that maximum allowed delay of each request is not exceeded. The second constraint ensures that the total allocated computational rate does not exceed the available capacity at MEC server. Next, the third and fourth constraint limit the total allocated bandwidth at BS to the available amount and bandwidth of a single UE to be greater than the minimum permitted. Each request can be allocated only to one server, MEC or cloud, so indicator $a^n$ takes value 0 or 1. Finally, allocated computational rates have to be positive numbers.

The minimization in (3) is a mixed integer programming problem, since indicators $a$ are binary variables, and therefore problem (3) is NP-hard. The relaxation of the integer variables to $a^n \in [0,1]$ is not suitable for this problem, as the execution of a task cannot be simply divided between MEC and Cloud servers. The optimal solution can be obtained iteratively by testing different combinations of these binary variables. At each step, a value for server association vector $a$ is picked and a non-integer subproblem is solved. The problem does not have a solution for every value of vector $a$. In order to analyze the subproblem we assume the indicator vector $a$ is given. We also aim at minimizing the total amount of resources used in the network, so we set the price of all resource units to one. Next, we substitute equations (1) and (2) in the first constraint of optimization problem (3):

$$\begin{align*}
    & \text{minimize} & \sum_{n=1}^{N} f^n_M + f^n_C + r^n \\
    & \text{subject to} & (1-a^n) \left( p^n_1 + p^n_2 + d^n_B + \frac{J^n}{f^n_M} \right) \leq D^n & n \in N \\
    & & a^n \left( p^n_1 + p^n_B + d^n_B + \frac{J^n}{f^n_C} \right) \leq D^n & n \in N \\
    & & \sum_{n=1}^{N} f^n_M \leq F \\
    & & \sum_{n=1}^{N} r^n \leq R \\
    & & r^n \geq R_{min} & n \in N \\
    & & 0 \leq \{M\} \leq \{C\} \leq \infty
\end{align*}$$

(4)

This optimization problem is quadratic, since the first and second constraints are not linear functions of variables $r^n, f^n_M$ and $f^n_C$. In order to linearize them, we have to introduce auxiliary variables $\bar{r^n}, \bar{f^n}_M$ and $\bar{f^n}_C$, and additional conditions. We can rewrite them as:

$$\begin{align*}
    & a^n \left( (p^n_1 + p^n_2 + d^n_B + \frac{J^n}{f^n_M}) \right) \leq D^n \\
    & a^n \left( (p^n_1 + p^n_B + d^n_B + \frac{J^n}{f^n_C}) \right) \leq D^n \\
    & r^n = \bar{r^n} M + 1; \quad f^n_M = \bar{f^n}_M M + 1; \quad f^n_C = \bar{f^n}_C C + 1
\end{align*}$$

(5)

Hyperbolic constraints (5c) can be rewritten as a second-order cone program (SOCP) constraint. We use the fact that any relation of the form $z^2 \leq xy$ can be transformed into

$$\begin{align*}
    & \frac{2z}{x+y} \leq \sqrt{x+y} \\
    & \frac{z}{x-y} \leq \sqrt{x+y}
\end{align*}$$

(6)

where $\| \cdot \|_2$ is $l_2$ norm. The optimization problem in (4) is then equivalent to:

$$\begin{align*}
    & \text{minimize} & \sum_{n=1}^{N} f^n_M + f^n_C + r^n \\
    & \text{subject to} & \left( \begin{array}{c}
            r^n \bar{r^n}_M \\
            2 \bar{r^n}_M \end{array} \right) \leq \left( \begin{array}{c}
            r^n \\
            \bar{r^n}_M
\end{array} \right) & n \in N
\end{align*}$$

(5a); (5b) $n \in N$
Algorithm 1 Optimal Resource Allocation

1: IF $N > N_{\text{max}}$
2: select $N_{\text{max}}$ requests; $N = N_{\text{max}}$
3: END
4: Initialize: $A_M = N$, $A_M^* = \text{Null}$, $i = 0$
5: WHILE $A_M^* = \text{Null}$
6: $A_M = A_M - i$
7: Generate all combinations for $a$ with $A_M$ zeros and $N - A_M$ ones
8: FOR $i = \text{Combination}_1$ TO $\text{Combination}_{\text{max}}$
9: Solve equation (7) for $a$
10: IF there is Solution:
11: $A_M^* = A_M$
12: $\text{Solution}_{\text{min}} = \text{min}\{\text{Solution}, \text{Solution}_{\text{min}}\}$
13: END
14: END
15: $i = i + 1$
16: END

The transformed sub-problem in optimization (7) is SOCP, and it has an efficient solution [29]. The objective to minimize $f_M^n + f_C^n + r^n$, together with the first and second constraint of problem (4) ensures that there is no resource over-provisioning. For the optimal solution we need $D_M^n = D^n$ or $D_C^n = D^n$ depending on the selected server, i.e. $d^n$. In summary, the optimum allocation can be found by the following Algorithm 1.

The algorithm first checks if all requests can be allocated to the MEC servers, as this is the most desirable outcome. If this is not possible, it investigates possibilities for offloading 1, 2, 3... requests to the Cloud server. The algorithm does not have to go through all the variations to find the minimum that solves (3). The biggest $A_M$ that has the solution for (7) is optimal for (3). This means that if there is an optimal solution for some $A_M^*$, all the solutions for $A_M^* + x$, where $x = 1, 2, 3...$ will require more resources because more tasks are offloaded to the Cloud server, and execution at the Cloud server requires more resources. For some $A_M^*$ there can be more than one possible allocation depending on the selected combination for vector $a$. Algorithm 1 finds which combination of $a$ for $A_M^*$ gives the minimal objective function value $\text{Solution}_{\text{min}}$. This solution is the optimal solution of the original problem (3). Although Algorithm 1 is more efficient than checking solutions of (7) iteratively for all combinations of $a$, it is still not efficient for large number of requests $N$. Nevertheless, we use this solution as a benchmark for comparing the heuristic algorithm’s performance. In order to construct efficient sub-optimal algorithm we analyse the optimal allocation characteristics in the rest of the section.

D. CHARACTERISTICS OF THE OPTIMAL SOLUTION

In this sub-section we present and discuss the performance of the optimal resource allocation solutions for several simple examples. We choose the examples that show certain characteristics of optimal solution and typical tendencies in algorithm’s output. We are particularly interested in the effects that each parameter of the request profile have on the allocation decision.

1) SINGLE TASK OPTIMAL ALLOCATION

First we investigate what is the optimal resource allocation in terms of minimizing the resource usage when the bandwidth and computational resources are not scarce. We look at a single task example were $F$ and $B$ are unlimited, to find the optimal split between two degrees of freedom: computation and communication delay. The communication rate that needs to be allocated to task $n$ in order to execute the task with delay constraint $D^n$, can be expressed as a function of allocated bandwidth. Bandwidth allocation $r^n$ of request $n$ can take one of two values: $r^n_M$ if assigned to MEC and $r^n_C$ if assigned to Cloud server. For MEC case we get from (1):

$$f_M^n = J^n / (D^n - p^n_1 + p^n_2 - d^n_{B,M})$$ (8)

The computation rate has to be a positive number, so we can obtain the minimum bandwidth requirement. For the MEC allocation the following condition must hold:

$$D^n - p^n_1 + p^n_2 - d^n_{B,M} > 0$$

$$r^n_M > (p^n_1 + p^n_2) / D^n - d^n_{B,M}$$ (9)

For the Cloud server allocation

$$r^n_C > (p^n_1 + p^n_2) / D^n - d^n_{B,M} - d^n_{M,C}$$ (10)

In order to find the analytical expression for single task optimal MEC allocation we need to find the minimum value of $f^n_M + r^n_M$ i.e. Eq. (8) defined by:

$$d(f^n_M + r^n_M) = (D^n - d^n_{B,M}) r^n_M - (p^n_1 + p^n_2)^2$$ (11)

As result, the optimum values for $f^n_M$ and $r^n_M$ are

$$r^n_M = \sqrt{f^n_M (p^n_1 + p^n_2)}$$ (12)

$$f^n_M = J^n / (D^n - p^n_1 + p^n_2 - d^n_{B,M})$$ (13)
Similarly, the optimum resource allocation for Cloud server is:

\[ r_C^n = \frac{J^n(p_1^n + p_2^n)}{D^n - d_{B,M}^n - d_{M,C}^n} \]  

(14)

\[ f_C^n = \frac{J^n}{(D^n - d_{B,M}^n - d_{M,C}^n)} \]  

(15)

For the illustration purposes, we consider an example of one request with delay limited to \( D = 5 \) ms for execution of \( J = 100 \) MI (Million Instructions) and transmission of \( p_1 + p_2 = 20 \) packets. The number of the links between BS and MEC is \( L_M = 1 \), and between MEC and Cloud server is \( L_C = 30 \). The transmission rate of the network slice is \( r_s = 40000 \) packets/s. The round-trip delay between BS and MEC is \( d_{M,M}^n = 0.5 \) ms, and between MEC and Cloud \( d_{M,C}^n = 2 \) ms. The possible allocation pairs \( (f_M, r_M) \) for MEC and \( (f_C, r_C) \) for Cloud offloading are shown in Fig. 3. Both computation rate \( f_M \) and \( f_C \) are expressed in Million Instructions Per Second (MIPS). Both functions \( f_M + r_M \) and \( f_C + r_C \) have a minimum, that is marked in the figure together with the corresponding minimum values of allocation pairs \( (f_M, r_M) \) and \( (f_C, r_C) \). We refer to these solutions as single task optimal solution for MEC and Cloud allocation.

2) MULTIPLE TASK OPTIMAL SOLUTION

In the first example we consider \( N = 5 \) requests with delay limited to \( D = 5 \) ms \( \forall n \). We set other task parameters same as in the previous example. The available bandwidth capacity is \( R = 100 \) packets/second and the vacant MEC computational capacity is \( F = 100 \) MIPS. Each request has \( J^n = 100 \) MI to execute and \( p_1^n + p_2^n = 20 \) packets to send. Round-trip delay between the BS and the MEC server is \( d_{B,M}^n = 0.5 \) ms, and between the MEC server and the Cloud center is \( d_{M,C}^n = 2 \) ms. The optimal server resource allocations \( f_M^n, f_C^n \) are shown in Fig. 4, while the optimal bandwidth allocation is presented in Fig. 5 for each task. For the optimal solution hyperbolic constrains in (7) hold with equality. Therefore solution of Algorithm 1 is the optimal solution of (3).

Results show that three requests are allocated to the MEC server and two requests are allocated to the Cloud. Since requests have equal demands, equal number of resources are allocated at each server. In case the delay requirement is the same, the services executed on the Cloud server require more computational power and bandwidth than requests executed at MEC server. This is due to additional delay through the network \( d_{M,C}^n \). We can see that in an optimal solution the task is allocated to the MEC servers whenever possible. The allocated bandwidth and computational resource \( (r^n, f_M^n) \) of tasks \( n = 1, 3, 4 \) are equal to the single task optimal MEC allocation, while \( (r^n, f_C^n) \) of tasks \( n = 2, 5 \) are equal to the single task optimal Cloud allocation, as presented in Fig. 3. Note that the request profiles \( Q_n \) of tasks in this example are the same as in the single task optimal example. In this example, since bandwidth is not the limiting factor, optimal resource allocation is a collection of sinle-task optimal allocations.

Next, we consider the example with the same parameters, where the request \( n = 2 \) has higher number of packets to transmit, \( p_1^2 + p_2^2 = 40 \) packets. The optimal computational
capacity and bandwidth allocation are presented in Fig. 6 and Fig. 7 for each task.

Similarly to the previous example, three tasks are allocated to MEC. However, we can see that in contrast to the previous example, the request 2 is now allocated to the MEC server. The request with higher number of packets needs more resources to be executed at both MEC and Cloud servers, as compared to the other requests. This time the request 2 is placed at the MEC server because it is more efficient from the resource point of view to send one of the less demanding requests to the Cloud. We can also see that both bandwidth and computational capacity allocations of \( n = 2 \) are higher, compared to other tasks \((n = 3, 4)\) allocated to MEC. This is because the excess delay, caused by higher number of packets, can be compensated by lower computation delay and faster transmission. The allocations \((r^n, j^n_c)\) are close to the single task optimal. However, in this example the bandwidth is the limiting factor \(\sum_{n} r^n = R\). Therefore, the tasks allocated to the cloud have slightly less bandwidth than in the single optimal case, while this additional delay is compensated with the higher allocation computational rate at the Cloud server.

In the third example, we consider the same parameters as in the first example except that request \( n = 2 \) has a higher number of instructions \( J_2 = 200 \) MI. The optimal solution is presented in Fig. 8 and Fig. 9.

The request with the higher number of instructions \( n = 2 \) is assigned to the Cloud server since the high number of instructions requires higher computational capacity to complete the task in a limited time. If allocated to a MEC server it would occupy more capacity than the other requests. Therefore, a smaller number of requests would be allocated to MEC server which is typically less efficient, especially when both bandwidth and MEC capacity are scarce. Again, the bandwidth is the limiting factor and the allocations of all tasks are slightly different than the single tasks optimal.

Finally, in the fourth example, we investigate the effect of delay constraint on the optimal allocation. In addition to parameters set in the first example, we set \( D^5 \) to 4.5ms.
The requests with the stricter delay limit require more resources to be completed in time than the others. The aim of the algorithm is to place shorter-delay requests to MEC because the delay between the MEC and the Cloud server further decreases the delay budget, in turn leading to even higher resource consumption. Results in Fig. 10 and 11 show that only two short-delay requests are assigned to MEC. Due to the lack of MEC resources, one short delay request is assigned to the Cloud server. As expected compared to the other request assigned to Cloud server, it requires more bandwidth and computational resources.

IV. HEURISTIC LIMITED LATENCY CO ALGORITHM

Due to inefficiency of finding the optimal resource allocation, in this section we propose a heuristic algorithm for limited latency CO problem. Based on the properties and trends of the optimal solution, discussed in the previous section, we construct a computationally simple algorithm that achieves close to optimal performance.

Equations (12)-(15) show that the limited-latency computational task has the unique resource allocation pair \( (r^*_M, f^*_M) \), which minimizes the total resource consumption in the network. The analysis of the optimal solutions showed that when resources are not limited, all the requests have the single task optimal allocation. If both MEC capacity and bandwidth are sufficient for such allocation, all requests are allocated to the MEC server for execution. If the MEC server capacity becomes the limiting factor, while the bandwidth is sufficient, some tasks need to be allocated to the Cloud server. The optimal solution shows that both MEC and Cloud-allocated requests have the single task optimal allocation.

If the bandwidth is scarce, the allocated rate \( r^n \) is lower than the single-task optimal \( r^*_M \), and the computational rate \( f^*_M \) or \( f^*_C \) is higher than single task optimal \( f^*_M \) or \( f^*_C \), in order to compensate for excess communication delay. In this case optimal allocation cannot be predicted in advance. However, based on these examples, we can see that the bandwidth is fully utilised. If the bandwidth is scarce, it is not possible to tell if the MEC capacity is sufficient, based on the knowledge of the single-task optimal allocation.

In our Heuristic Resource Allocation (HRA) algorithm, we assume that the requests are allocated one by one. So far we identified 3 areas, in which the optimal solution has different characteristics. In the first case, both MEC and bandwidth resources are sufficient and all requests are allocated with the single-task optimal allocation. This is easy to replicate by the proposed heuristic algorithm, where one by one request is allocated \( (r^*_M, f^*_M) \) if \( \sum r^*_M > F \) and \( \sum f^*_M > R \).

The second case is when the bandwidth is sufficient and the MEC resources are scarce, i.e. \( \sum r^*_M < F \). The sufficiency of the bandwidth cannot be checked in advance, i.e. before it is determined which requests are sent to the Cloud server. First, the heuristic algorithm allocates the task one by one with \( (r^*_M, f^*_M) \) to MEC, and when MEC runs out of capacity, allocates them to the Cloud. However, the allocation of the requests in the order of arrival is suboptimal. Instead, the requests with shorter delays should be processed first, because the goal is to allocate them to the MEC server. Example 4 also shows that the shorter-delay request is allocated to MEC in the optimal solution. It is preferred for the requests with high number of instructions to be executed at the Cloud server, as shown in Example 3 in Fig. 8 and Fig. 9. This is because they require a lot of computational resources and can block the MEC resources from many other tasks. Finally, the requests with higher number of packets should be processed before those with less packets, in order to increase their chance of being admitted to MEC, because it is not efficient to send large amount of data out of the local network.

The heuristic algorithm should first sort the requests by the ascending order of delay. If two requests have the same delay requirements, the one with the lower number of instructions \( J \) is allocated before the higher one. If also the number of instructions is the same for both, the requests with the higher number of packets \( p1 + p2 \) should be processed first. Sorting the resources does not guarantee that the optimal solution would be achieved, but it increases efficiency compared to the algorithm working on a FIFO principle.

The allocation in these two cases, i.e. when the bandwidth is sufficient, can be handled with the single algorithm, as shown in Fig. 12. We refer to this part of the HRA as Procedure 1. It sorts the requests, aiming at sorting requests approximately by the amount of resources needed, and does the single-task optimal allocation. It first tries to assign the task to the MEC server, and if vacant resources are not sufficient, to the Cloud server.

In the third case, the bandwidth resources are scarce \( \sum r^*_M < R \), while MEC resources are either sufficient \( \sum f^*_M > F \) or scarce \( \sum f^*_M < F \). This means that for some \( n \) \( r^n < r^*_M \). However, it is not possible to know how much smaller the bandwidth allocation \( r^n \) is from the optimal. On the other hand, since the bandwidth allocation decreases, the computational rate allocation increases compared to \( f^*_M \), so there is no guarantee that all tasks can be executed at the MEC server, i.e. that the MEC resources are sufficient. This means that the cases where MEC capacity is sufficient and scarce cannot be distinguished and need to utilize the same heuristic procedure. Again, the first step is to sort the requests. The whole available bandwidth is divided between
the tasks. Then, requests are processed one by one. MEC rate \( f_M^i \) is calculated based on (13). If vacant MEC resources are sufficient, the request is assigned to MEC, otherwise \( f_C^i \) is calculated and the request is assigned to the Cloud. We refer to this part of HRA as Procedure 2, Fig. 12.

If the bandwidth is sufficient, the resource allocation algorithm selects the Procedure 1. If the bandwidth is not sufficient, it selects the Procedure 2. However, as mentioned earlier, when MEC capacity is scarce, it is not possible to allocate the request if bandwidth is sufficient since we do not know which tasks will be allocated to the Cloud. The optimal Cloud allocation requires more bandwidth that optimal MEC allocation. This can only be determined during the Procedure 1 allocation, if at any point the requested rate is greater than leftover vacant bandwidth \( r^n > B \). In this case, Procedure 1 is terminated and the allocation is performed from the start by Procedure 2. During the execution of Procedure 1, tasks are only assigned to the servers. The allocation is only performed at the end of the algorithm.

The steps of the HRA algorithm in the form of pseudo code are summarized in Algorithm 2. Similar to the Optimization algorithm, the first step is to determine \( N \) processed in that allocation period. In the next step requests are sorted in the manner described above. At the start of the Procedure 1, the vacant capacities \( B \) and \( F \) are set to the total available capacities at the time of allocation \( B = R \) and \( F = R \). For each request, it is checked if it can be assigned to the MEC with \( (r_M^*, f_M^*) \). If vacant MEC capacity is insufficient it is checked if request can be assigned to Cloud server with \( (r_C^*, f_C^*) \). It should be noted that only rate condition needs to be checked, because the cloud capacity is considered unlimited. The indicator \( d' \), where \( i \) is the index in the sorted requests list, is set to 0 if request is assigned to MEC or to 1 if the request is assigned to the Cloud. After each assignment, the vacant capacities are updated. Finally, if the last request is successfully assigned, the resources are allocated to each task based on the calculated assignment. If for any request \( i \), the bandwidth is not sufficient, the Procedure 1 is terminated and the algorithm continues with the Procedure 2. Vacant capacities are reset to the initial values \( B = R \), \( F = R \), and the available bandwidth is split between the tasks. Each task is first assigned minimum rate \( r_{\min} \) and the leftover bandwidth is split equally. For each request, \( i \) and assigned bandwidth \( r^i \) algorithm calculates \( f_M^i \) that satisfies delay constraint. Next, it is checked if request \( i \) can be assigned to the MEC. If yes, \( d' \) is set to 0 and vacant MEC capacity is updated. Otherwise, the request is assigned to the Cloud with the rate \( f_C^i \) that satisfies delay constraint for \( r^i \). Finally, the allocation is performed, based on the assignment.

HRA algorithm is very efficient in terms of time and space complexity. The time complexity of the algorithm is linear with the number of requests \( N \). The algorithm updates 4 vector variables \( f_M^*, f_C^*, f_M^* \) and \( f_C^* \) of all size \( N \). Therefore, the space complexity is also linearly dependent on the number of requests.

V. NUMERICAL RESULTS

In this section we analyse the performance of the Heuristic Resource Allocation (HRA) Algorithm. We compare its
performance to the optimal solution as well as to a simple heuristic algorithm. Due to the lack of the appropriate two-tier architecture resource allocation algorithm with strict latency limitations in the literature, we use the following benchmark algorithm. Requests are allocated in the order of arrival with single-task optimal allocation. This ensures that the latency condition is met.

In Fig. 13, the performance of our algorithm is compared to the performance of optimal resource allocation as defined in (3). Since both the optimal allocation and HRA have the same acceptance rate, we compare the total amount of resources the algorithms utilize to serve all computational requests. This is the sum of the transmission rates at the wireless link and computational resources at the MEC and Cloud servers. Due to the large number of input parameters in Fig. 13, we present total resources used in 4 multi-task example scenarios defined in Section V C.

Depending on the case, the results show that HRA performs either equal or slightly worse than optimal allocation while its implementation is significantly simpler. In the cases where the bandwidth is sufficient for the single-task optimal allocation, such as in example 1, our proposed algorithm performs optimally. If the single-task optimal allocation is not possible, our algorithm has sub-optimal performance. In example 2, our HRA algorithm needs around 8.3% more resources to accommodate the same tasks compared to the optimal allocation. Therefore sub-optimal solutions are very close to the optimal while requiring significantly less implementation effort.

In Fig. 14, we compare the total resource consumption of our HRA and optimal algorithm as the number of requests increase from \( N = 2 \) to \( N = 7 \). Every request has the delay threshold of \( D = 5 \)ms for execution of \( J = 100 \) MI and transmission of \( p_1 + p_2 = 20 \) packets. The round trip delay between the BS and the MEC server is \( d^B_{B,M} = 0.5 \)ms, and \( d^M_{M,C} = 2 \)ms between the MEC server and the Cloud center. The available bandwidth capacity is \( W_b = 100 \) packets/second, and vacant MEC computational capacity is \( F = 100 \) MIPS. The capacity parameters stay fixed as the number of requests increase. Our results show that the HRA algorithm performs optimally up to \( N = 5 \) requests. For higher number of requests, HRA algorithm performs very close to optimal.

Next, we compare performance of our solution to the baseline algorithm. The baseline algorithm does not have the same acceptance rate as our algorithm, so we cannot conduct a meaningful comparison with respect to the total resources used. Instead in Fig. 15, the number of accepted requests is presented for the example scenarios, defined in Section V C. HRA always finds a feasible allocation to accommodate all the delay-constrained requests. The baseline algorithm underperforms when the bandwidth is not sufficient for the single-task optimal allocation.

In Fig. 16, we compare the percentage of the accepted resources using the HRA and the Baseline algorithm, as the number of requests increase form \( N = 2 \) to \( N = 7 \). The other parameters are the same as in the setup for Fig. 14. Each request has equal demands and the capacities of the systems are fixed. The baseline algorithm performs well for small number of requests, i.e. when the system capacity is sufficient for the total demand. However, with increased number of requests, the performance of the baseline algorithm rapidly degrades. For \( N = 7 \), request acceptance rate is only around 55%.

In the previous example, the HRA algorithm has an advantage over the Baseline algorithm when the system capacity is insufficient for the single-task optimal allocation. This is because the HRA algorithm utilizes the Procedure 2 in Algorithm 2. However, the HRA algorithm has an advantage over the Baseline algorithm even when using Procedure 1,
due to the sorting of the requests. In the previous example, this is not utilised since all requests are equal. We use the following example to illustrate the benefits of sorting requests in the HRA algorithm. The available bandwidth capacity $W^b = 100$ packets/second and vacant MEC computational capacity $F = 100$ MIPS are utilised to serve $N = 5$ limited latency requests. Computational tasks have the following capacities: $Q^1 = \{D^1 = 4, J^1 = 75, p^1 = 20\}$, $Q^2 = \{D^2 = 4.5, J^2 = 120, p^2 = 15\}$, $Q^3 = \{D^3 = 5, J^3 = 90, p^3 = 25\}$, $Q^4 = \{D^4 = 5, J^4 = 100, p^4 = 20\}$ and $Q^5 = \{D^5 = 6, J^5 = 80, p^5 = 20\}$. Requests are given in a sorted order of arrival. We investigate 5 different orders of arrival: order 1: $\{1, 2, 3, 4, 5\}$, order 2: $\{1, 2, 3, 4, 5\}$, order 3: $\{1, 2, 3, 4, 5\}$, order 4: $\{1, 2, 3, 4, 5\}$, order 5: $\{5, 2, 3, 4, 1\}$. In Fig. 17, the percentage of the accepted requests for HRA and Baseline algorithm is presented for different orders of arrivals. The performance of the Baseline algorithm depends on the order of arrival and for certain orders it performs equally with the HRA, but for others it performs worse. This is a consequence of the single-task allocation using more bandwidth resources when the request is sent to the Cloud service, and therefore it is important to intelligently sort the request to those to be sent to the cloud and those to be sent to the MEC.

VI. DISCUSSION AND CONCLUSION

In this paper we study the problem of two-tier computational offloading of latency limited tasks. We formulate the joint optimization problem to obtain offloading decision, i.e. server selection, and allocation of communication and computation resources necessary to execute the tasks. The objective is to minimize the usage of the resources while maximizing the number of accepted latency-limited task requests. We show that global optimization problem is NP-hard and we propose a computationally efficient heuristic solution. Simulation results show the efficiency of our proposed algorithm, compared to the optimal solution and a benchmark heuristic algorithm. We show for different system parameters that our proposed solution outperforms the benchmark algorithm in terms of the acceptance rate and gives allocations that are equal or close to optimal. Minimizing the usage of resources reduces the cost of computational offloading for infrastructure and service providers. At the same time it improves the scalability of the system, which is especially important in networks with large number of IoT devices. Our algorithm enables execution of computational tasks with strict delay requirements, which is necessary for delay-critical services. Maximizing the number of accepted requests further improves the quality of experience for the end-users and the overall system reliability.

ACKNOWLEDGMENT

The authors would like to thank Tanesh Kumar, Jude Okwuibe, Johirul Islam, and Muneeb Ejaz on a productive discussions over the course of this research.

REFERENCES

[1] P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” IEEE Commun. Surveys Tuts., vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.
[2] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, “Survey on multi-access edge computing for Internet of Things realization,” IEEE Commun. Surveys Tuts., vol. 20, no. 4, pp. 2961–2991, 4th Quart., 2018.
[3] Y. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing a key technology towards 5G,” ETSI, Sophia Antipolis, France, White Paper 11, 2015, pp. 1–16.
[4] T. Taleb, K. Samdanis, B. Mada, H. Flink, S. Dutta, and D. Sabella, “On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration,” IEEE Commun. Surveys Tuts., vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.
[5] E. Harjula, P. Karhula, J. Islam, T. Leppänen, A. Manzoor, M. Liyanage, J. Chauhan, T. Kumar, I. Ahmad, and M. Ylianttila, “Decentralized IoT edge nanoservice architecture for future gadget-free computing,” IEEE Access, vol. 7, pp. 119856–119872, 2019.
[6] X. Jiang, H. Shokri-Ghadikolaei, G. Fodor, E. Modiano, Z. Pang, M. Zorzi, and C. Fischione, “Low-latency networking: Where latency lurks and how to tame it,” Proc. IEEE, vol. 107, no. 2, pp. 280–306, Feb. 2019.
[7] M. Weiner, M. Jorgovanovic, A. Sahai, and B. Nikolic, “Design of a low-latency, high-reliability wireless communication system for control applications,” in Proc. IEEE Int. Conf. Commun. (ICC), Jun. 2014, pp. 3829–3835.
[8] K. Yamamoto, F. Ichihara, K. Hasegawa, M. Tukuda, and I. Omura, “60 GHz wireless signal transmitting gate driver for IGBT,” in Proc. IEEE 27th Int. Symp. Power Semiconductor Devices IC’s (ISPSD), May 2015, pp. 133–136.
[9] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, “Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research,” IEEE Commun. Surveys Tuts., vol. 21, no. 1, pp. 88–145, 1st Quart., 2019.
[10] C. Wang, F. R. Yu, C. Liang, Q. Chen, and L. Tang, “Joint computation offloading and interference management in wireless cellular networks with mobile edge computing,” IEEE Trans. Veh. Technol., vol. 66, no. 8, pp. 7432–7445, Aug. 2017.
X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” IEEE/ACM Trans. Netw., vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, “Multiobjective optimization for computation offloading in fog computing,” IEEE Internet Things J., vol. 5, no. 1, pp. 283–294, Feb. 2018.

Y. Tang and S. He, “Multi-user computation offloading in mobile edge computing: A behavioral perspective,” IEEE Netw., vol. 32, no. 1, pp. 48–53, Jan. 2018.

N. Nouri, P. Ralfeet, and A. Tadaion, “NOMA-based energy-delay tradeoff for mobile edge computing offloading in 5G networks,” in Proc. 9th Int. Symp. Telecommun. (IST), Dec. 2018, pp. 522–527.

X. Sun and N. Ansari, “Latency aware workload offloading in the cloudlet network,” IEEE Commun. Lett., vol. 21, no. 7, pp. 1481–1484, Jul. 2017.

N. Daneshfar, N. Pappas, V. Polishchuk, and V. Angelakis, “Service allocation in a mobile fog infrastructure under availability and QoS constraints,” in Proc. IEEE Global Commun. Conf. (GLOBECOM), Dec. 2016, pp. 1–6.

L. Liu and Q. Fan, “Resource allocation optimization based on mixed integer linear programming in the multi-cloudlet environment,” IEEE Access, vol. 6, pp. 24533–24542, 2018.

N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, “Edge mobile computing: A survey,” IEEE Internet Things J., vol. 5, no. 1, pp. 450–465, Feb. 2018.

S. Sardellitti, G. Scutari, and S. Barbarossa, “Joint optimization of radio and computational resources for multicell mobile-edge computing,” IEEE Trans. Signal Inf. Process. Over Netw., vol. 1, no. 2, pp. 89–103, Jun. 2015.

R. Kaewpuang, D. Niyato, P. Wang, and E. Hossain, “A framework for cooperative resource management in mobile cloud computing,” IEEE J. Sel. Areas Commun., vol. 31, no. 12, pp. 2685–2700, Dec. 2013.

X. Fan, X. He, D. Puthal, S. Chen, C. Xiang, P. Nanda, and X. Rao, “CTOM: Collaborative task offloading mechanism for mobile cloudlet networks,” in Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM), May 2018, pp. 1–6.

X. Wu, J. Yin, X. Guan, G. Bai, and B.-Y. Choi, “A pricing based cost-aware dynamic resource management for cooperative cloudlets in edge computing,” in Proc. 27th Int. Conf. Comput. Commun. Netw. (ICCCN), Jul. 2018, pp. 1–6.

Y. Liu, M. J. Lee, and Y. Zheng, “Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system,” IEEE Trans. Mobile Comput., vol. 15, no. 10, pp. 2398–2410, Oct. 2016.

Q. Li, L. Zhao, J. Gao, H. Liang, L. Zhao, and X. Tang, “SMDP-based coordinated virtual machine allocations in cloud-fog computing systems,” IEEE Internet Things J., vol. 5, no. 3, pp. 1977–1988, Jun. 2018.

J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, “IRAF: A deep reinforcement learning approach for collaborative mobile edge computing IoT networks,” IEEE Internet Things J., vol. 6, no. 4, pp. 7011–7024, Aug. 2019.

T. Zhao, S. Zhou, L. Song, Z. Jiang, X. Gao, and Z. Niu, “Energy-optimal and delay-bounded computation offloading in mobile edge computing with heterogeneous clouds,” China Commun., vol. 17, no. 5, pp. 191–210, May 2020.

J. Zhao, Q. Li, Y. Gong, and K. Zhang, “Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks,” IEEE Trans. Veh. Technol., vol. 68, no. 8, pp. 7944–7956, Aug. 2019.

I. Kovacevic, A. S. Shafigh, S. Glisic, B. Lorenzo, and E. Hossain, “Multi-domain network slicing with latency equalization,” IEEE Trans. Netw. Service Manage., vol. 17, no. 4, pp. 2182–2196, Dec. 2020.

M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, “Applications of second-order cone programming,” Linear Algebra its Appl., vol. 284, nos. 1–3, pp. 193–228, Nov. 1998.

IVANA KOVACEVIC (Member, IEEE) received the bachelor’s degree in electronics and telecommunication engineering from the University of Belgrade, Serbia, in 2012, and the M.Sc. degree in communications engineering from the University of Oulu, Finland, in 2015, where she is currently pursuing the Ph.D. degree in wireless networks. She joined the Centre for Wireless Communications, University of Oulu, in 2015. Her research interests include network slicing, low latency communications, radio resource management, edge computing, network optimization theory, game theory, and machine learning.