Dynamic Task Offloading and Scheduling for Low-Latency IoT Services in Multi-Access Edge Computing

Article *in* IEEE Journal on Selected Areas in Communications · January 2019

DOI: 10.1109/JSAC.2019.2894306

5 authors, including:

Hyame Alameddine
Concordia University Montreal
9 PUBLICATIONS **40** CITATIONS
SEE PROFILE

Sanaa Sharafeddine
Lebanese American University
70 PUBLICATIONS **281** CITATIONS
SEE PROFILE

Samir Sebbah
Concordia University Montreal
29 PUBLICATIONS **165** CITATIONS
SEE PROFILE

Sara Ayoubi
Concordia University Montreal
30 PUBLICATIONS **264** CITATIONS
SEE PROFILE

Dynamic Task Offloading and Scheduling for Low-Latency IoT Services in Multi-Access Edge Computing

Hyame Assem Alameddine[1], Sanaa Sharafeddine[2], Samir Sebbah[1], Sara Ayoubi[3] and Chadi Assi[1]

[1]Concordia University
[2]Lebanese American University
[3]INRIA

***Abstract*—Multi-access Edge Computing (MEC) has recently emerged as a novel paradigm to facilitate access to advanced computing capabilities at the edge of the network, in close proximity to end devices, thereby enabling a rich variety of latency sensitive services demanded by various emerging industry verticals. Internet of Things (IoT) devices, being highly ubiquitous and connected, can offload their computational tasks to be processed by applications hosted on MEC servers due to their limited battery, computing, and storage capacities. Such IoT applications providing services to offloaded tasks of IoT devices are hosted on edge servers with limited computing capabilities. Given the heterogeneity in the requirements of the offloaded tasks (different computing requirements, latency, etc.), and limited MEC capabilities, we jointly decide on the task offloading (tasks to application assignment) and scheduling (order of executing them) which yields a challenging problem of combinatorial nature. Further, we jointly decide on the computing resource allocation for the hosted applications, and we refer to this problem as the Dynamic Task Offloading and Scheduling (DTOS) problem, encompassing the three subproblems mentioned before. We mathematically formulate this problem and owing to its complexity, we design a novel thoughtful decomposition based on the technique of Logic Based Benders Decomposition. This technique solves a relaxed Master, with fewer constraints, and a subproblem, whose resolution allows the generation of cuts which will, iteratively, guide the master to tighten its search space. Ultimately, both the master and the sub-problem will converge to yield the optimal solution. We show that this technique offers several order of magnitude (more than 140 times) improvement in the run time for the studied instances. One other advantage of this method is its capability of providing solutions with performance guarantees. Finally, we use this method to highlight insightful performance trends for different vertical industries as a function of multiple system parameters with focus on delay sensitive use cases.**

***Index Terms*—Multi-access edge computing, Internet of Things, 5G, task offloading, resource allocation, scheduling.**

I. Introduction

The expectations towards a premium Quality of Experience (QoE) are widely increasing with the recent advancement of 5G networks, paving the way towards a broad new set of services such as augmented/virtual reality, traffic safety, image and face recognition, etc. [1], [2]. Enabling proximity services with fast service delivery at anytime in crowded areas, comes hand-in-hand with the goal of 5G systems in providing user-centric QoE that includes but is not limited to ultra-low latency (i.e., $\leq 5ms$), ultra-high reliability (i.e., 99.999%) and a support of 1000 times higher data volumes [1], [3]. The need for the support of higher data volumes comes as a result of the tremendous foreseen increase in the number of wearable Internet of Things (IoT) devices expected to reach 50 billion in 2020 as reported by Cisco [4] along with a subscription uptake forecast for 5G to reach 1 billion by 2023 as predicted by Ericsson [5].

The constrained computation and storage resources of mobile User Equipments (UEs) and IoT devices, in addition to their limited battery life make them not suitable for supporting the processing of resources-hungry mobile and IoT applications providing 5G services such as online gaming and face/speech recognition applications [2], [6], [7]. One possible solution to overcome these limitations is the use of Mobile Cloud Computing (MCC) which allows the UEs to offload their computation-intensive tasks to be processed on applications deployed on a powerful, centralized remote cloud accessible via the Internet or a core network of a mobile operator, thus, expanding the capability of mobile devices [2], [8]. However, such approach incurs high communication delays due to the distant location of the remote cloud from the user which violates the QoE requirements for some real-time delay-sensitive applications. Hence, in order to provide fast service delivery compliant with a real-time context, Mobile Edge Computing (MEC) has been introduced to bring cloud computing resources at the edge of the network, closer to the end-user [1], [9]. MEC was first introduced to accelerate the advancements on edge computing in mobile networks and within the Radio Access Network (RAN). However, the definition of MEC was then slightly modified to "*Edge Computing refers to a broad set of techniques designed to move computing and storage out of the remote cloud (public or private) and closer to the source of data*" [10] to accommodate various set of access technologies (e.g., WiFi, Long-Term Evolution(LTE), etc.) [11]. As such, the European Telecommunications Standards Institute (ETSI) dropped the "Mobile" from MEC and renamed it Multi-access Edge Computing (MEC) [1], [11]. It is worth noting that other edge computing paradigms such as cloudlet and Fog computing exist and differ by the access technology they use and the applications they

This work is supported by NSERC and Concordia University - Canada.

target [12].

MEC consists of edge servers deployed at the edge of the network and implemented either at the cellular Base Stations (BSs) or at the local wireless Access Points (APs) [6]. As the MEC computation resources are more limited than those available in MCC, MEC servers can at anytime offload their demanding tasks to MCC servers through the Internet whenever they are overloaded. Further, multiple MEC servers can collaborate and offload their tasks to each others (e.g., via a backhaul network) to provide better services for the mobile users through balancing their workloads and sharing their resources [8]. For instance, a nearby edge server can choose to offload the task of a connected UE to another edge server, if it does not host the required application to process the task, or if its application is overloaded or can not be allocated enough computing resources to process the task within its delay requirement. Hence, efficiently utilizing MEC resources is necessary to guarantee its foreseen benefits which are intimately associated with solving the following challenges: 1) *The task offloading problem* which consists of determining to which edge server each task should be offloaded. More precisely, it consists of associating each task to an application hosted on an edge server and able to process it; 2) *The application resource allocation problem* which determines the computing resources to be allocated to each application[1] deployed on an edge server in order to process all its assigned tasks within their delay requirements; 3) *The task scheduling problem* which decides on the order in which each task should be processed on the shared application while meeting its deadline.

While some work in the literature focused on determining the edge server to which each task should be offloaded and the computing resources it needs to be allocated [6], [13], [14], [15], [16], others addressed the joint problem of task offloading and scheduling either through stochastic optimization [9], [17] or using algorithmic solutions [8], [18], [19]. The methods explored in existing works are approximate solutions which do not explore the benefits that can be brought by enabling dynamic modifications of the computing resources allocated to the shared IoT applications and their impact on the scheduling of offloaded tasks.

In this work, we follow a more holistic approach for task offloading with joint resource allocation and scheduling with special focus on delay-sensitive IoT services [20]. This is motivated by recent 5G standardization activities on ultra-low latency uses cases, in addition to the emerging trend of zero touch networks empowered by virtualization technologies that enable dynamic scaling of resources. Hence, our contributions can be summarized as follows:

- We mathematically define and formulate the *Dynamic Task Offloading and Scheduling problem (DTOS)* as a Mixed Integer Program (MIP) *(DTOS-MIP)*.
- Given its complexity, we explore the *DTOS-LBBD*, a Logic Based Benders Decomposition (LBBD) approach to efficiently solve the DTOS problem to optimality. The DTOS-LBBD decomposes the DTOS problem into a master problem that performs the task offloading and the resource allocation; and multiple sub-problems, each addresses the scheduling of tasks offloaded to a single IoT application.
- Extensive numerical evaluations are carried out to examine the efficiency of the DTOS-LBBD compared to the DTOS-MIP in terms of runtime. In addition, valuable performance trends are explored to highlight the impact of task offloading and scheduling in meeting the diverse QoE requirements aligned with 5G vertical industries.

The remainder of the paper is organized as follows. Section II presents the literature review. Section III discusses the system model and motivates the problem. Section IV defines and mathematically formulates the DTOS problem. Section V presents and explains the various aspects of the DTOS-LBBD approach. Our numerical evaluation is depicted in Section VI. We conclude in Section VII.

II. Literature review

A. Joint Task Offloading and Resource Allocation

Sun et al. [14] solved the latency-aware workload offloading (LEAD) problem where they mathematically formulated the task offloading problem under the objective of minimizing the average response time for mobile users and presented an algorithmic approach to solve it efficiently. The limited resource pool in MEC in comparison to a centralized cloud computing motivated many work in the literature to jointly address the resource allocation and task offloading problem in the quest to efficiently utilize these resources [11].

The authors of [6] mathematically formulated the joint problem of task offloading and resource allocation in MEC where they do not only account for the computing resource allocation but also for the transmission power allocation of mobile users. Given the NP-hardness of the problem, they decompose it into a task offloading problem which they solve using a heuristic approach and a resource allocation problem which they solve using convex and quasi convex optimization techniques. Unlike [6], the work of [13] focused on minimizing the cost of the online resource allocation in MEC under unpredictable resources prices and user mobility. The authors of [13] provide an online optimization-based algorithm to solve the resource allocation problem of mobile users at each time slot by considering adapting their allocated resources based on the optimal solution obtained at the previous one.

The work in [16], [15] accounted for mobile users requesting a determined type of IoT applications. With the objective of minimizing the average response time in terms of network and computing delays, the authors of [16] formulated and addressed the problem of placing IoT applications of different types on existing MEC while deciding on their computing resources and determining the tasks that will be offloaded to each of them. In their work, they specified a maximum allowable computing delay for each application. In contrast, Jia et al. [15] considered a Network Function Virtualization (NFV) based MEC, where they solved the task offloading

[1]An IoT application can be hosted on a Virtual Machine(VM) or a container/docker hosted at the MEC. The amount of resources of the VM/-docker/container is to be determined.

problem for a set of mobile users requesting a specific type of Virtual Network Function (VNF) (a software implementation of a network function) within specific latency requirements. The work in [7] aimed at evaluating the impact of edge computing and its enabling technologies on the response time. Thus, the authors of [7] consider the special use case of a mobile gaming 3-D application and evaluate the response delay incurred by offloading the tasks to be processed on edge servers deployed at three different locations (local deployment, special-purpose cloud infrastructure and commercial public cloud). Their experimental evaluation shows that the location of edge servers and the virtualization technology used (i.e., container, Virtual Machine (VM), bar metal) highly impacts the latency experienced. Other works on the task offloading and resource allocation problem have been reviewed in [12], [21].

B. Task Scheduling

An efficient resource utilization entails a smart orchestration of resources sharing which can not be accomplished without a proficient scheduling of their utilization. Hence, to maximize the revenue of the infrastructure owner, Katsalis et al. [17] devised a Lyapunov optimization framework to address the VMs placement and scheduling problem while accounting for the Service Level Agreement (SLA) for time-critical services. Their scheduling approach considers the scheduling of the number and the type (small, medium, large) of VMs to deploy at each time slot for each mobile service operator based on the variability of its workload. Similarly, the authors of [9] employed a Lyapunov function to decide on the offloading schedules of task while stochastically maximizing the network utility under partial out-of-date network states information without any consideration for the tasks delay requirements. The authors of [18] jointly optimized the task offloading and scheduling problems along with the transmit power allocation problem. They were interested in minimizing the weighted sum of execution delay and service energy consumption. They decomposed the problem into a task offloading and scheduling problem which they solve using the Johnson's algorithm with the objective of minimizing the makespan of all the jobs and a transmit power allocation problem which they address using convex optimization. The work in [18] considers that all the jobs are sharing a single-CPU core at the edge server while overlooking the deadline requirements of the tasks.

The task assignment and scheduling problem was investigated by Wang et al. [19] as well, who presented an algorithmic solution to solve the problem while accounting for the deadline requirement of the tasks and for the scheduling of their transmission and computation. The authors of [8] solved the task offloading problem while accounting for the possibility of dispatching jobs to a remote cloud as well as to an MEC. In addition, they presented a preemptive scheduling for the offloaded tasks with the objective of minimizing their weighted response time using an online algorithm.

C. Novelty of our Work in Comparison to the Literature

While only few works in the literature accounted for the deadline of the offloaded tasks and their demands on being processed by specific type of IoT applications, we target in this work, the QoE requirements of the rising 5G services of different business verticals by presenting a complete offloading scheme that accounts for a joint resource provisioning of IoT applications as well as a fine grained task scheduling to meet the delay sensitive requirements of these 5G services. We show that such computing resource provisioning directly affects the scheduling decisions and impacts the number of tasks that can be admitted to the network. Further, another major contribution of this work is the solution methodology and the achieved scalability. Although the DTOS problem is very complex, to the best of our knowledge, we are the first to present a LBBD framework for the this problem that is able to achieve several order of magnitude of faster run times while providing the optimal solution.

III. System Model

The tremendous move towards smart cities is gaining momentum with the development of 5G. Smart cities will enable several services such as smart traffic management, security, energy efficiency and smart health care which make use of multiple IoT devices and applications. We consider a smart city Wide Area Network (WAN) as depicted in Fig.(1), composed of a set S of cellular base stations which can be represented by either a macro cell (eNB) or a small cell (SCeNB). For simplicity and without loss of generality, we will represent by *eNB* any type of base station. In order to enable flexible routing and communications among eNBs, we consider that the core cellular network is enabled with Software Defined Network (SDN) technology. SDN is a new paradigm that simplifies network management by logically centralizing the control logic in one centralized entity called SDN controller. Instructed by network applications about the desired network policy (e.g., a routing application decides on the path of a flow); the SDN controller installs the appropriate forwarding rules in all forwarding devices' routing tables such as those in openFlow switches [22]. Thus, we consider an SDN-based cellular network enabled with SDN controllers and openFlow switches [16]. The SDN controllers benefit from a global view of the network and can be used to provide some monitoring based information such as the latency experienced by a flow between two eNBs [23]. A subset of eNBs in S are mounted with MEC servers to provide computation offloading services to the IoT users (i.e., User Equipments (UEs) such as tablets and wearable devices) (Fig.(1)). The MEC-enabled system that we consider operates in a time-slotted structure, where we denote by $\delta \in \Delta$ a time slot.

Let M be a set of deployed MEC servers; each MEC server $m \in M$ consists of a pool of physical servers with an aggregated computing capacity c_m specified in terms of cycles/second or MHz. The MEC servers are hosting a set A of IoT applications of multiple types (e.g., face recognition, video encoding, etc.) designated to process the offloaded tasks of UEs. Each application $a \in A$ is a software which can be deployed on top of a VM or a container hosted on a MEC server $m \in M$. It is of specific type that we depict by $t_a \in T$ where T is a set of IoT application types. Like any other software, an application $a \in A$ requires some minimum system specifications to be able to operate efficiently. For simplicity

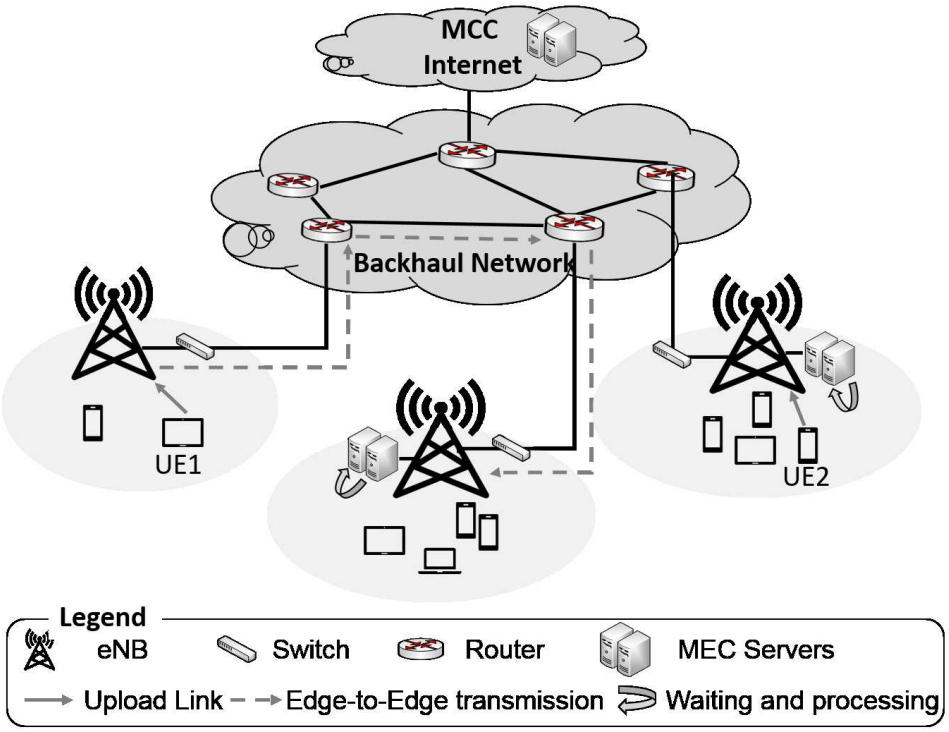


Fig. 1: System Model.

and without loss of generality, we represent the minimum system requirements of an application a by the minimum processing capacity p^a_{min}, it requires to operate. However, each application can be provisioned with some computing resources p_a, that exceed its minimum requirement p^a_{min}, in order to maximize the workload it can process within a time limit. The processing capacity p_a of an application is represented in terms of cycles/second and is related to the virtual CPU (vCPU) resources (number of cores) assigned to the VM/container on top of which the application is running [24]. Further, we assume that these applications can be shared by many UEs but can process the task of one UE at a time.

A. UEs Computation Tasks

We consider a set U of UEs requesting to offload their delay-sensitive tasks to be processed by an IoT application $a \in A$ of a suitable type deployed on an edge server $m \in M$. In this work, we account for a quasi-static scenario where the set U of UEs remains unchanged during the offloading period[2], however, it may change across different periods. We consider that each UE $u \in U$ has one computation task at a time. Thus, in the following we use task and UE interchangeably. We represent each task by a tuple $< t_u, \mu_u, \theta_u >$ where $t_u \in T$ depicts the type of the IoT application required to process the task of UE $u \in U$. μ_u represents the workload (cycles) required to accomplish the processing of the task of UE u and can be obtained by profiling the task execution [25]. θ_u denotes the latency requirements (i.e., deadline) in terms of time slots of the task of UE u. Note that, if the latter was not processed within its deadline, it will be rejected from the network.

[2]We leave the study of the dynamic scenario where mobile UEs arrive and depart dynamically during an offloading period for future work.

B. Experienced Delays

Processing the offloaded tasks with respect to their latency requirements entails deciding on the MEC server to which each of the tasks should be offloaded, determining the computation resources to allocate to the IoT applications that will process the tasks, in addition to specifying the order in which the offloaded tasks should be processed by each of the applications. Solving the three aforementioned challenges highly impacts the admission of the tasks to the network as they directly affect some of the delays they experience. In the following, we summarize the delays that an offloaded task experiences.

1) Upload delay d^u_{up}: The task uploading delay corresponds to the time required to transmit the task from the UE u to the serving eNB. We assume that the serving eNB $s \in S$ of each UE u is the base station with the highest received signal quality. For simplicity and without loss of generality, we assume that d^u_{up} is predefined and can be calculated based on the Signal to Interference plus Noise Ratio (SINR) as explained in [6].

2) Edge-to-edge delay d^u_{ee}: Once the task of UE u is uploaded, it should be processed by an IoT application $a \in A$ of a suitable type, deployed on the MEC $m \in M$ to which the task was offloaded. It is of the best interest of u to have its task processed by the MEC attached to its serving eNB to avoid any additional network delays. However, the serving eNB may not be enabled with MEC capabilities (UE1 in Fig.(1)), or the MEC server attached to it may not be able to process the task of u within its deadline θ_u; that is, 1) the MEC server

m is not hosting an application instance a of the same type of that required by u ($t_a! = t_u$) or 2) the hosted application instance a of the same type requested by u does not have enough processing capacity p_a to meet the task's deadline, or 3) a is overloaded and hence, the task of UE u will have to experience long waiting delay in its buffer before being processed as other tasks were scheduled before it since a can process the task of one UE at a time. Thus, in any of these situations, the task of UE u can be offloaded to another MEC server m' that is able to process it with respect to its QoE requirements. In this case, the serving eNB needs to transmit the task to another eNB $s \in S$ where m' is hosted. Hence, we denote by d^u_{ee}, the delay incurred for transmitting the task of a UE u from its serving eNB to the eNB connected to the MEC server where the task of u will be processed. As our SDN-based cellular core network can be used to establish a routing path between two eNBs [14], the edge-to-edge delay d^u_{ee} can be measured by the SDN controllers enabled with monitoring tools such as SLAM [23]. Thus, we define a matrix $\mathcal{H}$ with elements h^m_u to represent the value of d^u_{ee} for each UE $u \in U$ to each MEC server $m \in M$. Note that $d^u_{ee} = h^m_u = 0$, if the MEC server m is attached to the serving eNB of u.

3) Waiting delay d^u_{wait}: When the task of UE u reaches the MEC server m hosting the IoT application a that can process it, it may experience some waiting delays, that we denote by d^u_{wait}, in the buffer of a. Such delay depends on the scheduling order and the size of tasks assigned to a.

4) Processing delay d^u_{proc}: Once the task of u starts processing on its assigned application a, it will experience some processing delay, that we depict by d^u_{proc}. d^u_{proc} is the time taken by a to execute the task of UE u and is inversely proportional to the computing resources allocated to a as specified in Eq.(1) where μ_u and p_a are as defined above.

$$d^u_{proc} = \frac{\mu_u}{p_a} \tag{1}$$

5) Download delay d^u_{down}: Once the execution of the task of a UE u by IoT application a is finalized, the output should be transmitted back to u. As the size of the output is usually much smaller than the initial size of the task, we assume that the download delay incurred by transferring the output to u is negligible [6]. Thus, we consider that $d^u_{down} = 0$.

In the following, we address the joint problem of task offloading, application resource allocation, and task scheduling and refer to it as the *Dynamic Task Offloading and Scheduling (DTOS)* problem.

IV. Dynamic Task Offloading and Scheduling - A mixed integer program (DTOS-MIP)

We define and mathematically formulate the Dynamic Task Offloading and Scheduling (DTOS) problem as a Mixed Integer Program (MIP).

A. Problem Definition

Let $G(N, E)$ be a physical network consisting of a set of nodes $N = R \cup M \cup S$ where R is a set of physical equipment (e.g., switches, routers, etc.) and M is a set of MEC servers attached to a set S of eNB; E is a set of links connecting them. Let A be the set of IoT applications of different types deployed on the MEC servers $m \in M$, and let U be the set of UEs requesting to offload and process their latency-sensitive tasks on these applications. The DTOS problem can be formally defined as follows:

Definition 1. *Given a physical network $G(N; E)$, a set U of UEs, each UE requesting to offload and process a generated task on an IoT application of the same type deployed on one of the MEC servers $m \in M$; determine the optimal assignment of the tasks generated by UEs to the set of applications $a \in A$, provision computing resources for each application a and schedule the processing of tasks assigned to each of them in order to maximize the number of admitted tasks with respect to their latency requirements.*

B. Problem Formulation

Table I delineates the parameters used in the formulation of the DTOS-MIP problem presented below. We define the

Network Inputs	
$G(N, E)$	Physical network of N nodes where $N = R \cup M \cup S$ and E links connecting them.
S	Set of eNBs.
M	Set of MECs.
R	Set of physical equipment.
A	Set of IoT applications to be deployed on $m \in M$.
T	Set of IoT application types.
P	Set of processing capacities which can be assigned to an IoT application $a \in A$.
$c_m \in \mathbb{N}^+$	Processing capacity of an MEC server $m \in M$.
$x^a_m \in \{0, 1\}$	Specifies that an MEC server $m \in M$ is hosting the IoT application $a \in A$ (1), or not (0).
$t_a \in \mathbb{N}^+$	Type of IoT application $a \in A$ ($t_a \in T$).
$p^a_{min} \in \mathbb{N}^+$	Minimum processing capacity required by the IoT application $a \in A$.
User Equipments Inputs	
U	Set of UEs.
$t_u \in \mathbb{N}^+$	Type of Iot application requested to process the task of UE $u \in U$ ($t_u \in T$).
$\theta_u \in \mathbb{N}^+$	Deadline of the task offloaded by UE $u \in U$.
$\mu_u \in \mathbb{N}^+$	Number of cycles required to process the task of UE $u \in U$.
$d^u_{up} \in \mathbb{N}^+$	Upload delay of the task of UE $u \in U$.
$h^m_u \in \mathbb{N}^+$	Edge-to-edge transmission delay of the task of a UE $u \in U$ to a MEC server $m \in M$.
Other Inputs	
Δ	Set of time slots (time line).
H	Big integer number.

TABLE I: Parameters of the DTOS-MIP.

variable $y^{a\delta}_u$ to determine that the IoT application $a \in A$ started processing the task of UE $u \in U$ at time slot $\delta \in \Delta$.

$$y^{a\delta}_u = \begin{cases} 1 \text{ if task of UE } u \in U \text{ started its processing on IoT} \\ \text{application } a \in A \text{ at time slot } \delta \in \Delta, \\ 0 \text{ otherwise.} \end{cases}$$

Our objective is to maximize the number of admitted tasks (Eq.(2)). A task of a UE $u \in U$ is admitted if it can be processed by an IoT application $a \in A$ within its specified deadline θ_u.

$$Maximize \sum_{u \in U} \sum_{a \in A} \sum_{\delta \in \Delta} y^{a\delta}_u \tag{2}$$

In order to meet our objective, several constraints that we elucidate in the following, have to be respected. Towards defining these constraints, we declare:
$p_a \in \mathbb{R}^+$ as a decision variable that determines the processing capacity allocated to an IoT application $a \in A$.
We introduce the variable $n_a \in \{0,1\}$ to depict that an IoT application $a \in A$ is used, that is, it is processing at least one task (1), or not (0).

$$n_a = \begin{cases} 1 \text{ if IoT application } a \in A \text{ is used,} \\ 0 \text{ otherwise.} \end{cases}$$

Further, we declare $s^a_{uu'} \in \{0,1\}$ as a decision variable to indicate that task of UE u started processing on IoT application a before the task of UE u'.

$$s^a_{uu'} = \begin{cases} 1 \text{ if task of UE } u \text{ started processing on application} \\ a \in A \text{ before the task of UE } u', \\ 0 \text{ otherwise.} \end{cases}$$

In order to maximize the number of admitted tasks, we need first to decide on the computing resources to allocate to the deployed IoT applications. Hence, we define Eq.(3) and Eq.(4) to specify that an IoT application a is used if at least one task is scheduled to be processed on it, and to ensure that it is not used otherwise.

$$n_a \leq \sum_{u \in U} \sum_{\delta \in \Delta} y^{a\delta}_u \quad \forall a \in A \tag{3}$$

$$H n_a \geq \sum_{u \in U} \sum_{\delta \in \Delta} y^{a\delta}_u \quad \forall a \in A \tag{4}$$

Eq.(5) guarantees that a used IoT application $a \in A$ is at least allocated the minimum computing resources p^a_{min} it requires to operate.

$$p_a \geq n_a p^a_{min} \quad \forall a \in A \tag{5}$$

Eq.(6) guarantees that the maximum computing capacity that can be assigned to an IoT application $a \in A$ can not exceed the capacity of the MEC server $m \in M$ hosting it.

$$p_a \leq n_a \sum_{m \in M} x^a_m c_m \quad \forall a \in A \tag{6}$$

Note that Eq.(5) and Eq.(6) ensure that an application $a \in A$ will not be allocated any computing resources if it is not used. Eq.(7) guarantees that the capacity of an MEC server $m \in M$ is not violated.

$$\sum_{a \in A} x^a_m p_a \leq c_m \quad \forall m \in M \tag{7}$$

A valid task offloading suggests that the task of a UE u can not be scheduled on more than one application $a \in A$ (Eq.(8)).

$$\sum_{a \in A} \sum_{\delta \in \Delta} y^{a\delta}_u \leq 1 \quad \forall u \in U \tag{8}$$

Further, the task of a UE $u \in U$, can not be scheduled by an IoT application $a \in A$ which is of different type $t_a \in T$ than the requested one $t_u \in T$ (Eq.(9)).

$$\sum_{u \in U} \sum_{a \in A:(t_a != t_u)} \sum_{\delta \in \Delta} y^{a\delta}_u \leq 0 \tag{9}$$

In addition, Eq.(10) guarantees that an IoT application $a \in A$ can at most process one task in a time slot.

$$\sum_{u \in U} y^{a\delta}_u \leq 1 \quad \substack{\forall \delta \in \Delta \\ \forall a \in A} \tag{10}$$

As we assume a non-preemptive scheduling, we present Eq.(11) and Eq.(12) to ensure that an IoT application $a \in A$ processes a UE task $u \in U$ completely before starting a new task. Thus, an IoT application a can not start processing the task of UE u' before finishing the processing of the task of a UE u; that is only if u is scheduled before u' on a (Eq.(11)). Similarly, an IoT application a can not start processing the task of a UE u before finishing the processing of the task of u'; that is only if u' is scheduled before u on a (Eq.(12)).

$$\sum_{\delta \in \Delta} y^{a\delta}_{u'} \delta \geq \sum_{\delta \in \Delta} y^{a\delta}_u \delta + d^u_{proc} - H(1 - s^a_{uu'}) \quad \substack{\forall a \in A:(t_u = t_{u'} = t_a) \\ \forall u,u' \in U:(u != u')} \tag{11}$$

$$\sum_{\delta \in \Delta} y^{a\delta}_u \delta \geq \sum_{\delta \in \Delta} y^{a\delta}_{u'} \delta + d^{u'}_{proc} - H(1 - s^a_{u'u}) \quad \substack{\forall a \in A:(t_u = t_{u'} = t_a) \\ \forall u,u' \in U:(u != u')} \tag{12}$$

d^u_{proc} in Eq.(12) and Eq.(11) determines the processing delay experienced by task of UE u on the IoT application processing it. It is calculated as in Eq.(13).

$$d^u_{proc} = \sum_{a \in A} \sum_{\delta \in \Delta} y^{a\delta}_u \frac{\mu_u}{p_a} \quad \forall u \in U \tag{13}$$

Eq.(14) represents the precedence constraint of the schedule of the tasks of UEs u and u' on IoT application $a \in A$.

$$s^a_{uu'} + s^a_{u'u} = \sum_{\delta \in \Delta} y^{a\delta}_u \sum_{\delta \in \Delta} y^{a\delta}_{u'} \quad \substack{\forall a \in A:(t_u = t_{u'} = t_a) \\ \forall u,u' \in U:(u != u')} \tag{14}$$

An IoT application a can not start processing the task of a UE u, unless the task is uploaded and transmitted to the application (Eq.(15)).

$$\sum_{a \in A} \sum_{\delta \in \Delta} y^{a\delta}_u d^u_{up} + d^u_{ee} \leq \sum_{a \in A} \sum_{\delta \in \Delta} y^{a\delta}_u \delta \quad \forall u \in U \tag{15}$$

where d^u_{ee} captures the edge-to-edge delay (Section III) and is determined as specified in Eq.(16).

$$d^u_{ee} = \sum_{m \in M} \sum_{a \in A} \sum_{\delta \in \Delta} y^{a\delta}_u x^a_m h^m_u \quad \forall u \in U \tag{16}$$

Finally, since we are addressing tasks with stringent deadlines, we need to ensure that the total delay experienced by a task of UE $u \in U$ should not exceed its deadline as specified in Eq.(17) where d^u_{proc} is as defined in Eq.(13).

$$\sum_{a \in A} \sum_{\delta \in \Delta} y^{a\delta}_u \delta + d^u_{proc} \leq \theta_u \quad \forall u \in U \tag{17}$$

Eqs. (11), (12), and (17) are non linear due to the non linearity of d^u_{proc} (Eq.(13)). Such non linearity is related to the division by the decision variable p_a which is multiplied by another decision variable $y^{a\delta}_u$. Hence, in order to linearize it, we reduce the search space by allowing p_a to take at most one specific value of a predefined set P instead of all the values in $\mathbb{R}^+$. This is determined by Eq.(18).

$$\sum_{p \in P} z^p_a \leq 1 \quad \forall a \in A \tag{18}$$

where z^p_a is a new decision variable defined as follows:

$$z^p_a = \begin{cases} 1 \text{ if } a \text{ is allocated the processing capacity } p \in P, \\ 0 \text{ otherwise.} \end{cases}$$

Thus, Eq.(13) can then be rewritten as in Eq.(19).

$$d^u_{proc} = \sum_{a \in A} \sum_{\delta \in \Delta} y^{a\delta}_u \sum_{p \in P} z^p_a \frac{\mu_u}{p} \quad \forall u \in U \tag{19}$$

Similarly, p_a can be replaced by $\sum_{p \in P} z^p_a p$ in constraints (5), (6) and (7). Finally, Eqs.(11), (12), (14), (17) and (19) are non linear and can be easily linearized, but we omit the linearization details due to space limitation.

C. DTOS Complexity

DTOS is a MIP (DTOS-MIP) which is complex to solve. It is NP-hard given that it is a combination of three NP-hard problems which are the task offloading problem, the application resource allocation problem and the the non-preemptive task scheduling problem. In fact, the task offloading problem can be been proven as NP-hard via a reduction from the generalized assignment problem [26] while the application resource allocation problem can be proven as NP-hard via a reduction from the two-dimensional bin-packing problem [27] where the MEC servers are the bins and the IoT applications are the objects to pack. Similarly, the task scheduling problem can be proven as NP-hard via a reduction from the job-shop scheduling problem [28].

V. DTOS-LBBD: A Logic Based Benders Decomposition

Given the complexity of the DTOS problem, we devise in the following a Logic-Based Benders Decomposition (LBBD) technique to solve it (DTOS-LBBD).

A. LBBD in a Nutshell

LBBD [29], is a row generation technique which follows the "no-good learning" strategy. It consists of decomposing the problem into a Master Problem (MP) representing a relaxation of the original model and one or more Sub-Problems (SPs). The SP, also known as inference dual, is an optimization over the secondary variables while fixing the primary variables to values computed based on the solution of the MP. By assigning the primary variables some trial values and solving the SP, the LBBD learns about the quality of other trial solutions, which are then used to reduce the number of solutions that need to be enumerated in order to find the optimal one. More precisely, the solution of the SP is used to derive Benders' cuts which are added to the MP in order to cut infeasible solutions from its solution space. An infeasible solution or a non globally feasible solution can be defined as follows:

Definition 2. *An infeasible solution or a non globally feasible solution is a solution provided by the MP that is not feasible to the SP (i.e., a solution that is unlikely to satisfy the SP constraints).*

Benders' cuts are derived from the inference dual which can be defined as the problem of inferring the tightest possible bound on the optimal value of the main problem [29]. LBBD consists of iteratively solving the MP and the SPs, deriving and adding Benders' cuts from the SPs to the MP until the MP and the SPs solutions converge.

B. DTOS Decomposition Strategy

The efficiency of LBBD relies on the decomposition approach and the strength of the defined Benders' cuts. Unlike the classical Benders approach where Benders' cuts can be easily defined based on Lagrangean multipliers obtained from the solution of the SP dual, no standard scheme for generating Benders' cuts exists for LBBD [30]. As classical Benders is inappropriate for the DTOS problem given that it requires that the SP be a continuous linear or non linear program [30], designing an efficient LBBD with strong Benders' cuts for the DTOS problem is challenging.

Hence, our DTOS-LBBD consists of dividing the DTOS problem into a MP which solves the task assignment and the application resource allocation problems, and multiple SPs to resolve the task scheduling problem. Jointly solving the task assignment and the application resource allocation problems has a positive impact on the efficiency of our DTOS-LBBD: 1) It overlooks the granularity of scheduling the tasks with respect to their latency requirements. Instead, it considers assigning the tasks of UEs to IoT applications of suitable types while allocating each of these applications the minimum processing capacity required to meet the deadline of each of the assigned tasks. This, indeed, provides an upper bound on the number of tasks that can be scheduled as it represents a relaxation of the original problem. 2) Since application schedules are independent from each other, our DTOS-LBBD design allows us to benefit from a distributed scheduling scheme by devising a scheduling problem for each used IoT application. Defining an SP per IoT application may not be possible if the application resource provisioning is to be solved as part of the LBBD SP due to the MEC servers capacity constraint which controls the processing capacity allocated to its hosted applications. Finally, a distributed scheduling enables the parallel execution of the SPs, and hence, reduces the overall computation time of the DTOS-LBBD.

As depicted in Fig.2, the DTOS-LBBD starts by solving the MP. The MP solution yields an assignment of a subset of tasks of UEs to the hosted IoT applications and a processing capacity allocated to each of these applications. For each used application $a \in A$, an SP is defined and fed by the set of tasks of UEs assigned by the MP to a and by the processing capacity p_a allocated to a. Let ψ_{MP_a} be the total number of tasks of UEs assigned to a by the MP. Further, let ψ_{SP_a} denote the maximum number of tasks of UEs that can be scheduled within their delay requirements on a by the SP. For every used application $a \in A$, if $(\psi_{MP_a} > \psi_{SP_a})$, a Benders' cut is derived and added to the MP to guide it towards determining a better value for p_a and hence performing task assignment that are likely to be all scheduled by the SP. The MP problem is solved again after adding the Benders' cuts inferred from all the SPs. This process is repeated until $\sum_{a \in A} \psi_{MP_a} = \sum_{a \in A} \psi_{SP_a}$ when the optimal solution is reached.

C. The Master Problem (MP)

The MP specific parameters are detailed in Table II while the remaining ones are as specified in Table I. We define the

Mater Problem Inputs	
P_u^a	Set of processing capacities which each of them, if assigned to application $a \in A$, enables the task of UE u to meet its deadline ($P_u^a \subset P$).
$\sigma_u^a \in \mathbb{N}^+$	Arrival time of task of UE $u \in U$ to application $a \in A$.
$\sigma_{min}^a \in \mathbb{N}^+$	Minimum arrival time to application $a \in A$ of all tasks of UE $u \in U$ that can be processed on it.
$\theta_{max}^a \in \mathbb{N}^+$	Maximum deadline of all the tasks of UEs $u \in U$ that can be processed on $a \in A$.

TABLE II: Parameters of the MP.

decision variable $q_u^a \in \{0,1\}$ to determine that the task of UE $u \in U$ is mapped to IoT application $a \in A$.

$$q_u^a = \begin{cases} 1 \text{ if task of UE } u \in U \text{ is mapped to IoT application } a, \\ 0 \text{ otherwise.} \end{cases}$$

The objective of the MP is to maximize the number of admitted tasks (Eq.(20)).

$$\psi_{MP} = Maximize \sum_{u \in U} \sum_{a \in A} q_u^a \tag{20}$$

The MP is subject to several constraints; we start by defining a decision variable $n_a \in \{0,1\}$ to specify that an application $a \in A$ is used, that is, if a task of at least one UE is assigned to it.

$$n_a = \begin{cases} 1 \text{ if IoT application } a \in A \text{ is used,} \\ 0 \text{ otherwise.} \end{cases}$$

We define $p_a \in \mathbb{R}^+$ as a decision variable that specifies the processing capacity allocated to an application $a \in A$.

MP basic constraints

The set of constraints includes Eq.(21) which depicts that the task of UE $u \in U$ can be processed by at most one IoT application $a \in A$.

$$\sum_{a \in A} q_u^a \leq 1 \quad \forall u \in U \tag{21}$$

Further, Eq.(22) is formulated to prevent the tasks of UEs to be processed on IoT applications of different type.

$$\sum_{u \in U} \sum_{a \in A: t_a != t_u} q_u^a = 0 \tag{22}$$

Eq.(23) and Eq.(24) specify that an IoT application $a \in A$ is used if at least one task is offloaded to be processed on it.

$$n_a \leq \sum_{u \in U} q_u^a \quad \forall a \in A \tag{23}$$

$$H n_a \geq \sum_{u \in U} q_u^a \quad \forall a \in A \tag{24}$$

Eq.(25) guarantees that an IoT application $a \in A$, if used, should at least, be allocated its minimum required processing capacity p_{min}^a.

$$p_a \geq n_a p_{min}^a \quad \forall a \in A \tag{25}$$

Eq.(26) guarantees that the maximum computing resources that can be allocated to an IoT application $a \in A$ can not exceed those of the MEC server hosting it.

$$p_a \leq n_a \sum_{m \in M} x_m^a c_m \quad \forall a \in A \tag{26}$$

Eq.(25) and Eq.(26), guarantee that no computing resources can be assigned to an unused IoT application $a \in A$. We define Eq.(27) to ensure that the capacity of each MEC server $m \in M$ is respected.

$$\sum_{a \in A} x_m^a p_a \leq c_m \quad \forall m \in M \tag{27}$$

Strengthening the MP formulation

While the above MP formulation provides an upper bound on the optimal solution of the DTOS problem, our experiments have shown that it is not sufficient to efficiently solve big test instances. Therefore, we strengthen the MP formulation by adding valid inequalities to further tighten the solution space. Thus, we first add Eq.(28) to determine a lower bound on the processing resources to be allocated to each application $a \in A$ based on its assigned tasks. The minimum processing resources needed to process all the assigned tasks to an application a can be determined based on Eq.(1) by considering the maximum processing time available to them on a, and which can be calculated by deducting the earliest arrival time σ_{min}^a to a (i.e., $\sigma_{min}^a = min \quad \sigma_u^a \quad \forall u \in U : (t_u = t_a)$ where σ_u^a accounts for the upload and edge-to-edge delays of task of UE u to a) from the maximum deadline θ_{max}^a of all tasks of UEs requiring the same type of a.

$$p_a \geq \frac{\sum_{u \in U: (t_u = t_a)} \mu_u q_u^a}{\theta_{max}^a - \sigma_{min}^a} \quad \forall a \in A \tag{28}$$

Similarly, p_a can be lower bounded by the computing resources that, if allocated to a, allows the task of UE u to meet its deadline on a. Thus, we determine such computing resources set P_u^a by a pre-processing scheme through first applying Eq.(29) to calculate the minimum processing resources (p_{min}^{ua}) required on a to meet the deadline of the task of UE u. P_u^a can then be determined by adding all the processing capacities $p \in P$ that exceeds p_{min}^{ua} and do not surpass the capacity of the MEC server hosting a ($P_u^a = \{p \in P : p_{min}^{ua} \leq p \leq \sum_{m \in M} x_m^a c_m\}$).

$$p_{min}^{ua} = \frac{\mu_u}{\theta_u - \sigma_u^a} \quad \substack{\forall u \in U \\ \forall a \in A: (t_u = t_a)} \tag{29}$$

Hence, we define $\beta_{ua}^j \in \{0,1\}$ as a new decision variable to determine the processing capacity $j \in P_u^a$ which is selected to process the task of UE $u \in U$ on application $a \in A$.

$$\beta_{ua}^j = \begin{cases} 1 \text{ if } j \in P_u^a \text{ is used for processing the task of UE } u \text{ on } a, \\ 0 \text{ otherwise.} \end{cases}$$

We add the inequality depicted in Eq.(30) as a constraint in the MP to specify that p_a should be greater than or equal to the maximum processing resources chosen to process any of the tasks of UEs $u \in U$ assigned to it.

$$p_a \geq \sum_{j \in P_u^a} j \beta_{ua}^j \quad \substack{\forall u \in U \\ \forall a \in A} \tag{30}$$

Eq.(31) is added to guarantee that one processing capacity is chosen for the task of UE $u \in U$ on application $a \in A$, if and only if, it is mapped on a.

$$\sum_{j \in P_u^a} \beta_{ua}^j = q_u^a \quad \substack{\forall u \in U \\ \forall a \in A} \tag{31}$$

As some tasks may experience high edge-to-edge delays if they were assigned to an application a of the same type hosted on a certain MEC m, the minimum processing capacity they require to meet their deadline on a may be very high and surpasses the capacity c_m of MEC m that can be provisioned to a (i.e, $P_u^a = \emptyset$). Thus, the assignment of such tasks to these applications will always lead to their rejection. Hence, we determine Eq.(32) to prune such assignments.

$$\sum_{u \in U: (P_u^a = \emptyset)} \sum_{a \in A: t_u = t_a} q_u^a = 0 \tag{32}$$

Note that as in DTOS-MIP, we replace p_a by $\sum_{p \in P} z_a^p p$ in Eqs. (25), (26), (27), (28) and (30) where $z_a^p \in \{0,1\}$ is a decision variable as defined in Section IV. We add Eq.(18) to guarantee that one processing capacity is allocated to an application $a \in A$.

D. The Sub-Problem (SP)

Table III depicts the parameters of the SP model. The remaining parameters are as specified in Table I. The SP for-

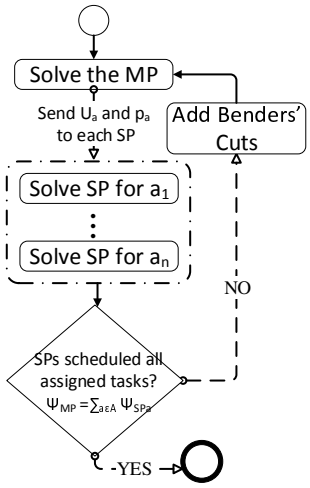


Fig. 2: DTOS-LBBD flowchart.

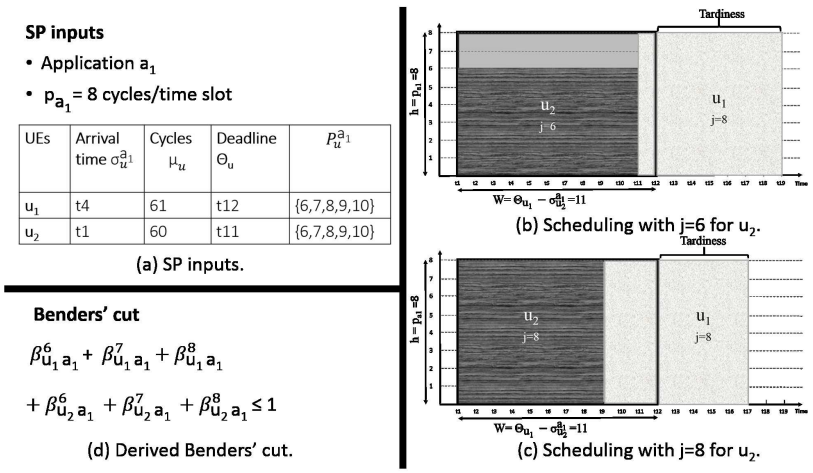


Fig. 3: Benders' cut insights.

mulation is presented in the following. We define $\alpha_u \in \{0,1\}$

Sub-problem Inputs	
a	Used application $a \in A$ for which the SP is defined.
p_a	Processing capacity of application $a \in A$.
U_a	Subset of UEs $u \in U$ which were assigned to application a based on the solution provided by the MP.
$\sigma_u^a \in \mathbb{N}^+$	Arrival time of task of UE $u \in U$ to application $a \in A$.

TABLE III: Parameters of the SP.

as a decision variable which determines whether the task of UE $u \in U_a$ is admitted on application a; that is it was able to be scheduled within its latency requirements on a.

$$\alpha_u = \begin{cases} 1 \text{ task of UE } u \in U_a \text{ is admitted on } a, \\ 0 \text{ otherwise.} \end{cases}$$

We define the decision variable $y_u \in \mathbb{N}^+$ to determine the time slot at which the task of UE $u \in U_a$ starts processing on application a. Further, we declare $s_{uu'} \in \{0,1\}$ as a decision variable to indicate whether the task of UE $u \in U_a$ started processing on application a before the task of a UE $u' \in U_a$.

$$s_{uu'} = \begin{cases} 1 \text{ task of UE } u \in U_a \text{ started processing on} \\ \text{application } a \text{ before the task of UE } u' \in U_a, \\ 0 \text{ otherwise.} \end{cases}$$

The objective of the SP is to maximize the number of admitted tasks (Eq.(33)).

$$\psi_{SP_a} = Maximize \sum_{u \in U_a} \alpha_u \tag{33}$$

The SP is subject to several constraints. Eq.(34) is used to guarantee that the tasks of UE $u \in U_a$ can not start processing on a before its arrival time, if it is admitted.

$$y_u \geq \sigma_u^a \alpha_u \quad \forall u \in U_a \tag{34}$$

Further, the application a should guarantee the consecutive processing of the task of UE $u \in U_a$ during all its required processing time. Hence, Eq.(35) and Eq.(36) ensure that no two tasks can be scheduled on a at the same time.

$$y_u \geq y_{u'} + d^{u'}_{proc} \alpha_{u'} - H(1 - s_{u'u}) \quad \forall u, u' \in U_a : (u! = u') \tag{35}$$

$$y_{u'} \geq y_u + d^{u}_{proc} \alpha_u - H(1 - s_{uu'}) \quad \forall u, u' \in U_a : (u! = u') \tag{36}$$

where d^u_{proc} and $d^{u'}_{proc}$ determine the processing delays of u and u' (Eq.(1)). Eq.(37) represents the precedence constraint of the schedule of the tasks of UEs u and u' on a.

$$s_{uu'} + s_{u'u} = \alpha_u \alpha_{u'} \quad \forall u, u' \in U_a : (u! = u') \tag{37}$$

Finally, Eq.(38) ensures that the total delay experienced by a task of UE $u \in U_a$ does not exceed its latency requirement.

$$y_u + d^u_{proc} \alpha_u \leq \theta_u \quad \forall u \in U_a \tag{38}$$

Note that, Eq.(37) is non linear and can be easily linearized, however, we remove the linearization details due to space limitation.

E. Benders' Cut

When an SP fails to schedule all the tasks of UEs assigned to application a by the MP, a Benders' cut has to be generated and added to the MP to prune such solution and similar non feasible ones. In fact, the failure of the SP to schedule all of the assigned tasks is a result of an allocation of the MP of low computing resources p_a to application a. Hence, a Benders' cut can be added to guide the MP to either increase the value of p_a while keeping the same assignment of tasks, or to assign fewer tasks on the application to match those that were able to be admitted by the SP. While such cut is valid, we believe it is not strong enough as it only prunes the solutions sent by the MP without considering any other similar infeasible ones.

In order to define a stronger Benders' cut, we try to identify the solutions that are likely to be provided by the MP and will be infeasible for the SP. Thus, we graphically depict an application a as a bin of height $h = p_a$ and of width $w = \theta^a_{max} - \sigma^a_{min}$ which indicates the time horizon during which the tasks of UEs $u \in U_a$ have to be scheduled and processed on a (Fig.(3)(b)). Each task $u \in U_a$ can be seen as a rectangle of height $h_u = j$ where j is the processing capacity assigned to it by the MP ($j \in P^a_u : \beta^j_{ua} = 1$) and width $w_u = d^u_{proc}$ representing the processing time of u on a when assigned the processing capacity j. The scheduling problem can then be abstracted to a bin-packing problem where a is the bin and the tasks are the objects to place in a. The geometrical size of the task $u \in U_a$ can increase by its height if we increase j

(i.e., more processing capacity) or by its width if we decrease j (i.e., extend its completion time) (Eq.(1)).

To elaborate, we consider the example shown in Fig.(3) where we assume two tasks u_1 and u_2, to be scheduled on application a_1 (Fig.(3)(a)). a_1 can be presented as a bin, and u_1 and u_2 are the objects to be placed in it (Fig.(3)(b) and Fig.(3)(c)). In Fig.(3)(b), we assign the task u_2 computing resources $j = 6$ $cycles/time\ slot$ which yields a processing delay of 10 time slots to finish at t_{11}. Increasing the computing resources to $j = 8$ $cylces/time\ slot$ for the processing of task u_2 decreases its processing time to 8 time slots to finish at t_9 (Fig.(3)(c)). However, in both cases it was not possible to admit u_1 on a_1 and satisfy its deadline.

Hence, with a processing capacity $p_{a_1} = 8$ $cylces/time\ slot$ assigned to application a_1, at most one of both tasks u_1 or u_2 can be admitted on a_1 when assigned any computing resources $j \leq p_{a_1}$. More precisely, varying the processing capacity j assigned to u_1 or u_2 will result in the same infeasible solution. Thus, we conclude that, if the set of tasks $u \in U_a$ were not able to be scheduled by the SP with $p_a \in P$, then for sure they will not be admitted with any other value $(p'_a < p_a) \in P$. Therefore, we define a cut (Eq.(39)) to prune such infeasible solutions where p_a is the processing capacity allocated to application a by the MP at the previous iteration, $\tilde{U}_a$ is the set of tasks of UEs that were admitted by the SP on a and $\hat{U}_a$ is the set of UEs whose tasks were rejected by the SP. Note that Eq.(39) guides the MP towards assigning to application a at most the same number of tasks $|\tilde{U}_a|$ that were admitted by the SP from the set of tasks in $(\tilde{U}_a \cup u'_{\in \hat{U}_a})$. Such guidance is only applicable in the case where the tasks in $(\tilde{U}_a \cup u'_{\in \hat{U}_a})$ were assigned computing resources $j \leq p_a$. The cut (Eq.(39)) is added for every task rejected by the SP. Fig.(3)(d) depicts the cut that needs to be added for the example in Fig.(3).

$$\underbrace{\sum_{u \in \tilde{U}_a} \sum_{j \in P_u^a : (j \leq p_a)} \beta_{ua}^j}_{Admitted\ tasks\ by\ SP} + \underbrace{\sum_{j \in P_{u'}^a : (j \leq p_a)} \beta_{u'a}^j}_{One\ rejected\ task\ by\ SP} \quad \begin{matrix} \forall u' \in \hat{U}_a \\ \forall a \in A \end{matrix}$$
$$\leq \underbrace{|\tilde{U}_a|}_{number\ of\ admitted\ tasks\ by\ SP} \tag{39}$$

To guarantee that the DTOS-LBBD converges to an optimal solution, we need to prove that Eq.(39) is a valid Benders' cut [31]. A Benders' cut is valid if it satisfies the following two conditions [31]:

Condition 1. *The cut must exclude the current MP solution if it is not globally feasible.*

Condition 2. *The cut must not remove any global feasible solutions composed of any combination of tasks that were selected by the MP at a previous iteration and requiring a processing capacity $j \leq p_a$.*

Chu and Xia [31] show that Condition 1 guarantees finite convergence if the MP variables have finite domains, and that Condition 2 guarantees optimality since the cuts never cut feasible solutions.

Theorem 1. *Benders'cut in Eq.(39) is valid.*

Proof. To prove the validity of our proposed cut, we need to show that Condition 1 and Condition 2 are satisfied.

We first prove that Condition 1 is satisfied. To show that Eq.(39) cuts off infeasible solutions provided by the MP, we will show that Eq.(39) will not be satisfied if the same set of tasks were admitted again by the MP on application a. Thus, we let $U_a^{(i)}$ be the set tasks which were admitted by the MP on application a at iteration i, hence resulting in a MP solution which is globally infeasible (as found by the SP). Further, let $\tilde{U}_a^{(i)}$ be the set tasks which were admitted by the SP at iteration i ($\tilde{U}_a^{(i)} \subset U_a^{(i)}$). This will result in the cut depicted in Eq.(40)

$$\sum_{u \in \tilde{U}_a^{(i)}} \sum_{j \in P_u^a : (j \leq p_a^{(i)})} \beta_{ua}^j + \sum_{j \in P_{u'}^a : (j \leq p_a^{(i)})} \beta_{u'a}^j \leq |\tilde{U}_a^{(i)}| \quad \begin{matrix} \forall u' \in \hat{U}_a^{(i)} \\ \forall a \in A \end{matrix} \tag{40}$$

If at a subsequent iteration $k > i$, the same set of tasks U_a is admitted by the MP, the left hand side in Eq.(40) will be equal to $|\tilde{U}_a^{(i)}| + 1$ as shown in Eq.(41) where $\beta_{ua}^{j(k)}$ depicts the MP solution at iteration k.

$$\underbrace{\sum_{u \in \tilde{U}_a^{(i)}} \sum_{j \in P_u^a : (j \leq p_a^{(i)})} \beta_{ua}^{j(k)}}_{|\tilde{U}_a^{(i)}|} + \underbrace{\sum_{j \in P_{u'}^a : (j \leq p_a^{(i)})} \beta_{u'a}^{j(k)}}_{1} = |\tilde{U}_a^{(i)}| + 1 \quad \begin{matrix} \forall u' \in \hat{U}_a^{(i)} \\ \forall a \in A \end{matrix} \tag{41}$$

The equality in Eq.(41) results from the following. Based on Eq.(31), a constraint of the MP that should be valid for its provided solution ($\beta_{ua}^{j(k)}$), we derive that $\sum_{j \in P_u^a} \beta_{ua}^{j(k)} = 1$. In addition, by accounting for the validity of Eq.(30), we obtain $\sum_{j \in P_u^a : (j \leq p_a^{(i)})} \beta_{ua}^{j(k)} = 1$. Hence, $\sum_{u \in \tilde{U}_a^{(i)}} \sum_{j \in P_u^a : (j \leq p_a^{(i)})} \beta_{ua}^{j(k)} = |\tilde{U}_a^{(i)}|$. Similarly, $\sum_{j \in P_{u'}^a : (j \leq p_a^{(i)})} \beta_{u'a}^{j(k)} = 1$. This explains the equality depicted in Eq.(41) which indeed shows that the MP solution violates the cut presented in Eq.(40). This proves that Condition 1 is satisfied.

Next, we prove that Condition 2 is satisfied. As we need to show that the cut (Eq.(39)) does not cut any feasible solution, we will provide a proof by contradiction where we consider a globally feasible solution W removed by the cut, and we show that such solution can not be feasible; where the contradiction resides. Hence, we first consider the legitimate infeasible solution I provided at iteration i and which resulted in the cut shown in Eq.(42), where not all the tasks assigned by the MP were admitted by the SP with a determined processing capacity $p_a^{(i)}$.

$$\sum_{u \in \tilde{U}_a^{(i)}} \sum_{j \in P_u^a : (j \leq p_a^{(i)})} \beta_{ua}^j + \sum_{j \in P_{u'}^a : (j \leq p_a^{(i)})} \beta_{u'a}^j \leq |\tilde{U}_a^{(i)}| \quad \begin{matrix} \forall u' \in \hat{U}_a^{(i)} \\ \forall a \in A \end{matrix} \tag{42}$$

The cut in (Eq.(42)) is designed to remove any infeasible solution composed of a subset of tasks from those in $U_a^{(i)}$ for any processing capacity less or equal than $p_a^{(i)}$ assigned to application a, and should not remove any feasible solutions. To prove this by contradiction, we consider a globally feasible solution W found at iteration $w > i$ that was removed by the cut (Eq.(42)). That is, in W, the tasks in $\tilde{U}_a^{(i)}$ admitted in I in addition to one or more tasks in $\hat{U}_a^{(i)}$ (that were rejected in

I) are admitted in W with a processing capacity $p_a^{(w)} \leq p_a^{(i)}$. Therefore, the opposite of the cut in Eq.(42) which is presented by Eq.(43) is valid for W.

$$\sum_{u \in \tilde{U}_a^{(i)}} \sum_{j \in P_u^a : (j \leq p_a^{(i)})} \beta_{ua}^{j(w)} + \sum_{j \in P_{u'}^a : (j \leq p_a^{(i)})} \beta_{u'a}^{j(w)} > |\tilde{U}_a^{(i)}| \quad (43)$$

As the tasks in W are assigned a processing capacity $p_a^{(w)} \leq p_a^{(i)}$, their processing time on application a will increase in comparison to that observed with $p_a^{(i)}$, and hence the total schedule length of all tasks will be greater or equal to that obtained with $p_a^{(i)}$. As such solution W is feasible, any other solution where the tasks experience less processing delay than that observed with $p_a^{(w)}$ should also be feasible (i.e., tasks meet their deadlines). For the tasks to experience less processing delays, they need to be assigned a processing capacity higher than $p_a^{(w)}$ which is the case of solution I which is infeasible. Hence, W can not be feasible which completes the proof. □

VI. Performance Evaluation

We carry out an extensive empirical study to evaluate the performance of our DTOS-LBBD approach against the DTOS-MIP. Further, we explore the engineering impact of the DTOS problem under varying system parameters and QoE requirements. We highlight the influence of the different problems solved (i.e., task offloading, application resource allocation and task scheduling) on serving multiple vertical industries while analyzing the effectiveness of our proposed DTOS-LBBD framework.

Industry Vertical	Allowable latency (ms)	Applied latency (θ_u) (ms)
Tactile Internet	1 - 10	7
Factory Automation	0.25 - 10	10
Smart Grid	3 - 20	20
Intelligent transportation Systems (ITS)	10 - 100	50
Tele Surgery	≤ 250	110

TABLE IV: Latency requirements of different industry verticals [32], [33].

A. Experimental Setup

In our numerical study, we consider networks of different sizes with varying number of MEC servers, each having a capacity of $c_m = 20Ghz$ [6]. We account for $|T| = 5$ different types of varying number of IoT applications that belong to the same industry vertical (unless stated otherwise). Each IoT application requires minimum computing resources (p_{min}^a) randomly generated between $[2-5]$ Ghz. The IoT applications are randomly placed on the MEC servers. We assume multiple UEs offloading tasks belonging to different industry verticals and hence, are of varying QoE requirements. Thus, we depict in Table IV the different industry verticals accounted for in our tests, and present the range of their latency requirements in addition to the ones used in our tests. We consider that the number of cycles (μ_u) demanded by UEs are randomly generated between $[20 - 100]$ $cycles$. The upload and edge-to-edge delays of the offloaded tasks are randomly generated between $[1 - 2]$ ms and $[1 - 3]$ ms [34] respectively. All our numerical evaluations are averaged over 5 sets. They are conducted using Cplex version 12.4 to solve the MIPs on an Intel core i7-4790 CPU at 3.60 GHZ with 16 GB RAM.

B. DTOS-MIP vs. DTOS-LBBD

We start by evaluating the performance of DTOS-LBBD against the DTOS-MIP in terms of execution time as we vary the number of UEs' offloaded tasks. Increasing the number of offloaded tasks makes the problem harder to solve given the limited computing resources. Hence, we also look at the the impact of such increase on the admission rate. Thus, we consider a network composed of $|M| = 3$ MEC servers and $|A| = 15$ IoT applications of $|T| = 5$ different types representing multiple industry verticals. The deadlines of the offloaded tasks are randomly generated between $[5 - 20]$ ms. Our results are presented in Table V.

	Execution Time (ms)		Admission Rate (%)	
Nb. of UEs ($\|U\|$)	*DTOS-MIP*	*DTOS-LBBD*	*DTOS-MIP*	*DTOS-LBBD*
5	922	56	92	92
10	2359	116.4	84	84
15	15512.6	1214	76	76
20	251218.4	10077.4	67	67
25	3014109.8	21602.6	62.4	62.4

TABLE V: DTOS-MIP vs. DTOS-LBBD.

1- Admission Rate: LBBD is an exact method which is able to provide the optimal solution as shown in Table V, where the admission rates of the DTOS-MIP and DTOS-LBBD are equal. The same table depicts that as the number of UEs increases the admission rate decreases. Such decrease is expected as more tasks are contending for the same amount of computing resources, hence, some of them will be suffering from high waiting delays on some IoT applications, waiting for them to be freed. This will negatively impact their latency requirements which will lead to miss their deadlines and get rejected from the network.

2- Execution Time: We evaluate the scalability of the DTOS-LBBD against the DTOS-MIP. Our results shown in Table V clearly depict that the DTOS-LBBD is much more scalable than the DTOS-MIP. In fact, it is able to provide the optimal solution on an average of 95% faster than the DTOS-MIP. This is because the LBBD learns from the quality of the solution generated at each iteration to cut off similar infeasible solutions from the solution space. This will restrict the search space as the number of iterations increases and hence, will help reaching the optimal solution faster than the DTOS-MIP. In addition, the decomposition of the problem into multiple sub-problems helps in reducing the execution time of DTOS-LBBD especially that multiple scheduling SPs are run in parallel using threads.

C. Evaluation of DTOS-LBBD

We evaluate the performance of DTOS-LBBD under different system parameters while studying the engineering impact of the DTOS problem.

1- DTOS-LBBD convergence: In order to evaluate the performance of DTOS-LBBD, we account for a single test instance and we plot in Fig.(4), the number of admitted tasks at each iteration as determined by the MP and the SPs. We consider a network of $|M| = 10$ MEC servers hosting $|A| = 15$ IoT applications. We account for $|U| = 30$ UEs' tasks belonging to a factory automation industry vertical ($\theta_u = 10ms$). Fig.(4) depicts that the objective of the MP

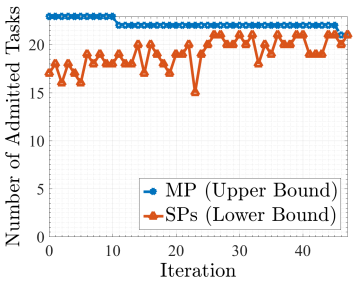


Fig. 4: DTOS-LBBD convergence.

	DTOS-LBBD Execution Time (ms)		
Nb. of UEs ($\|U\|$)	*Optimal Solution*	*Optimality Gap*< 10%	10% <*Optimality Gap*< 20%
20	15474.8	2816.6	634.2
30	295859	30607	8085
40	1473334.6	516176.8	27890
50	1760259	419640.3333	38563.33333

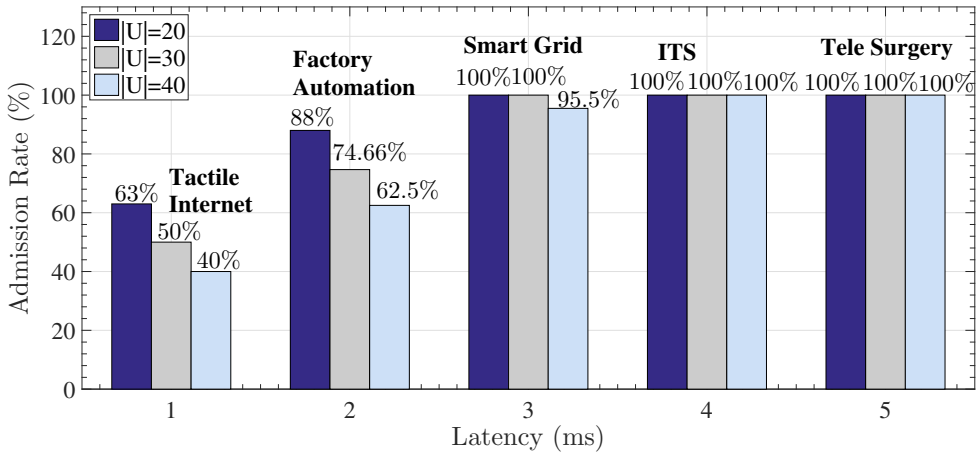
TABLE VI: DTOS-LBBD execution time (ms).

represents an upper bound on the optimal objective value while the number of tasks admitted by the SPs represents a lower bound. As the number of iterations increases, the objective value of the MP decreases given that more Benders' cuts are added to it. In contrast, the number of tasks admitted by the SPs varies between the iterations depending on the requirements of the tasks (i.e., number of cycles, arrival time) sent by the MP at each of them. However, it is important to note that the optimal objective value always lies between the maximum lower bound and the minimum upper bound attained so far. Further, the variance of the gap existing between the upper and lower bound provides the option to terminate the DTOS-LBBD at anytime based on the desired solution quality and the runtime. For instance, one may terminate the DTOS-LBBD at iteration 14 with a gap of 9% between the upper and lower bound, scarifying little in the quality of the solution while gaining about 75.4% in terms of runtime. If a better solution quality is desired, one can stop the DTOS-LBBD at iteration 26 where the gap reaches 4.5%; however, the gain in terms of computation time is about 53%.

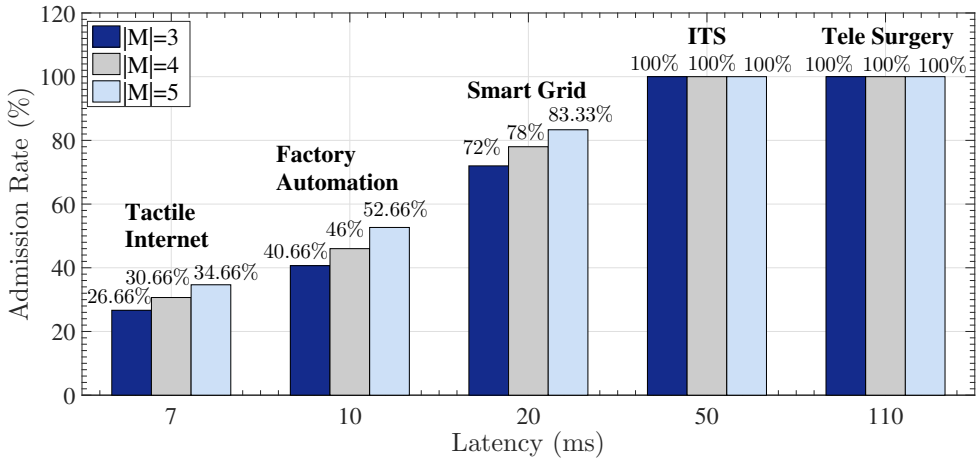
2- Trade-off between optimality gap and runtime: To further emphasize the fact that the LBBD approach represents an anytime algorithm that can be stopped at any iteration while providing a feasible solution, we show in Table VI an averaged runtime of the DTOS-LBBD using the same network settings mentioned in the previous paragraph. The results reported in Table VI depict the runtime of the DTOS-LBBD at the optimal solution and at the first occurrence of an optimality gap which is either less than 10% or between 10% and 20%. It is clear that for a determined number of UEs, the runtime increases with the decrease of the optimality gap. In fact, the runtime of DTOS-LBBD increases with the increase of the number of iterations. In this case, more Benders' cuts are added to the MP tightening its solution space, hence, better locating the optimal solution which is likely to decrease the gap between the upper bound provided by the MP and the lower bound given by the SPs. Thus, stopping the DTOS-LBBD at a certain tolerable gap can lead to high gains in terms of computation time. For instance, when $|U| = 30$, 97.26% of gain in runtime is depicted when the gap is between 10% and 20%, while 89.65% is obtained with a gap less than 10%. Finally, it is worth noting that as the number of UEs increases, the runtime of the DTOS-LBBD increases as the size of the problem grows and hence, the problem becomes harder to solve.

3-Impact of varying the number of UEs: We vary the number of UEs and evaluate its impact for different industry verticals. Thus, we consider a network of $|U| = 10$ MEC servers hosting $|A| = 15$ IoT applications. Our results presented in Fig.(5(a)) show that as the number of UEs increases the admission rate decreases for each vertical industry as more tasks are contending the same computing resources (IoT applications) which become overloaded and hence, fail to meet the delay requirements of all the UEs requesting their service. In fact, some tasks will suffer from extra waiting delays which will lead them to miss their deadlines and thus, get rejected from the network. However, such waiting delays can be tolerated if the latency requirements increased. For instance, one can note the tactile Internet vertical where the number of UEs increased from $|U| = 20$ to $|U| = 40$ while the admission rate decreased by 23% as the limited computing resources failed to cope with such increase. In contrast, for $|U| = 20$, the admission rate increased to 100% for less latency sensitive tasks such as those belonging to Tele Surgery industry. Further, as Intelligent Transportation Systems (ITS) and Tele surgery vertical industries possesses relatively high delay requirement, the admission rate of UEs requiring such types of services was not affected by the increase of the number of tasks and was kept constant to 100%.

4- Impact of varying the number of MEC servers: We study in Fig.(5(b)) the impact of the increase of the computing resources for different vertical industries on the admission rate. Hence, we consider a network of varying number of MEC servers hosting $|A| = 15$ IoT applications. We account for $|U| = 30$ UEs offloading tasks of varying latency requirements. Fig.(5(b)) shows that adding more MEC servers in the network increases the amount of computing resources available. This allows the hosted IoT applications to be provisioned more processing capacity, which will reduce the processing time of the assigned tasks. Thus, as tasks will be processed faster by the applications, others waiting for the same resource to be freed will experience less waiting delays, and hence their chances in meeting their deadlines and be admitted to the network will increase. In addition, one can note that for a fixed number of MEC servers, the admission rate increases with the increase of the latency requirements; as less-sensitive tasks can tolerate more waiting delay on the shared IoT applications. For instance, with $|M| = 3$ MEC servers, the admission rate increased by 73.34% as the deadline of the tasks increased from $7ms$ for tactile Internet to $110ms$ for Tele surgery. Further, the 100% admission rate depicted for ITS and Tele surgery vertical industries for the varying number of MEC servers depicts that $|M| = 3$ MEC servers



(a) Admission rate for varying number of UEs.



(b) Admission rate for varying number of MEC servers.

Fig. 5: Admission rate of DTOS-LBBD for different verticals.

were enough to admit all the assigned tasks given their relaxed latency demands.

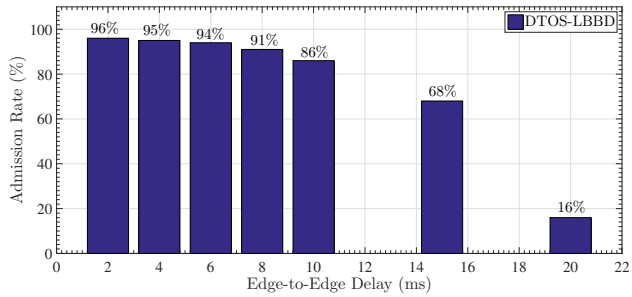


Fig. 6: Admission rate over varying edge-to-edge delay.

5- Impact of varying the edge-to-edge delay: To explore the impact of the edge-to-edge delay on the admission rate, we consider a network of $|M| = 5$ MEC servers hosting $|A| = 15$ IoT applications. We account for $|U| = 25$ UEs belonging to a smart grid industry vertical ($\theta_u = 20ms$) and we fix the edge-to-edge delays for all the tasks to a defined value. Our results presented in Fig.(6) show that the edge-to-edge delay increase becomes prohibitive in allowing the admission of the tasks. In fact, it is of the best interest of each task to be processed on an MEC server attached to its serving eNB in order to overcome the edge-to-edge delay. However, as the number of IoT applications is fixed in the network, some MEC servers may not be hosting certain types, further, some of their deployed applications may be overloaded. This will force the tasks served by eNB attached to those MEC servers to travel through the network to be processed on an IoT application hosted on another MEC server, which will make them suffer from high edge-to-edge delay. With their latency-sensitive requirements, the mentioned tasks will be left with very little processing time which the IoT application on which they are assigned might fail to meet, hence, leading to their rejection from the network.

VII. Conclusion

In this paper, we motivated and studied the DTOS problem which jointly addresses the task offloading, application resource allocation in addition to the task scheduling problems

in a MEC network. We alleviate virtualization technologies capabilities in being able to automatically modify their allocated computing resources, by being the first to study the task scheduling problem under undetermined computing resources allocation. Given the complexity of DTOS, we presented a novel decomposition strategy implementing the LBBD technique. Our novel DTOS-LBBD method decomposes the problem into a MP which solves the task offloading and application resource allocation problems; and multiple SPs, each addressing the scheduling of tasks on a single used IoT application. DTOS-LBBD is an exact method characterized by an anytime algorithm providing the opportunity to be terminated at any iteration, hence, realizing the trade-off between the solution quality and the computation time. Through extensive simulations, we show that the DTOS-LBBD can achieve more than 140 order of magnitude improvement in terms of runtime compared to the DTOS-MIP and can serve as a benchmark algorithm to compare against other methods. Further, we explored the interleaving dependence and implications of the aforementioned DTOS sub-problems for different vertical industries with variable latency requirements. While mainly targeting low-latency services, we evaluated the trade-offs existing between the number of MEC servers, the number of tasks and the latency requirements.

REFERENCES

[1] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[2] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *arXiv preprint arXiv:1702.05309*, 2017.

[3] 5G Infrastructure PPP Association et al. 5g vision-the 5g infrastructure public private partnership: the next generation of communication networks and services. *White Paper, February*, 2015.

[4] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1(2011):1–11, 2011.

[5] Ericsson. New capabilities with distributed cloud.

[6] Tuyen X Tran and Dario Pompili. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *arXiv preprint arXiv:1705.00704*, 2017.

[7] Gopika Premsankar, Mario Di Francesco, and Tarik Taleb. Edge computing for the internet of things: a case study. *IEEE Internet of Things Journal*, 5(2):1275–1284, 2018.

[8] Haisheng Tan, Zhenhua Han, Xiang-Yang Li, and Francis CM Lau. Online job dispatching and scheduling in edge-clouds. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2017.

[9] Xinchen Lyu, Wei Ni, Hui Tian, Ren Ping Liu, Xin Wang, Georgios B Giannakis, and Arogyaswami Paulraj. Optimal schedule of mobile edge computing for internet of things using partial information. *IEEE Journal on Selected Areas in Communications*, 35(11):2606–2615, 2017.

[10] Alex Reznik, Rohit Arora, Mark Cannon, Luca Cominardi, Walter Featherstone, Rui Frazao, Fabio Giust, Sami Kekki, Alice Li, Dario Sabella, et al. Developing software for multi-access edge computing. *ETSI, White Paper*, (20), 2017.

[11] Hiroyuki Tanaka, Masahiro Yoshida, Koya Mori, and Noriyuki Takahashi. Multi-access edge computing: A survey. *Journal of Information Processing*, 26:87–97, 2018.

[12] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H Glitho, Monique J Morrow, and Paul A Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 20(1):416–464, 2017.

[13] Lin Wang, Lei Jiao, Jun Li, and Max Mühlhäuser. Online resource allocation for arbitrary user mobility in distributed edge clouds. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 1281–1290. IEEE, 2017.

[14] Xiang Sun and Nirwan Ansari. Latency aware workload offloading in the cloudlet network. *IEEE Communications Letters*, 21(7):1481–1484, 2017.

[15] Mike Jia, Weifa Liang, and Zichuan Xu. Qos-aware task offloading in distributed cloudlets with virtual network function services. In *Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems*, pages 109–116. ACM, 2017.

[16] Qiang Fan and Nirwan Ansari. Application aware workload allocation for edge computing-based iot. *IEEE Internet of Things Journal*, 5(3):2146–2153, 2018.

[17] Kostas Katsalis, Thanasis G Papaioannou, Navid Nikaein, and Leandros Tassiulas. Sla-driven vm scheduling in mobile edge computing. In *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*, pages 750–757. IEEE, 2016.

[18] Yuyi Mao, Jun Zhang, and Khaled B Letaief. Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems. In *Wireless Communications and Networking Conference (WCNC), 2017 IEEE*, pages 1–6. IEEE, 2017.

[19] Lin Wang, Lei Jiao, Dzmitry Kliazovich, and Pascal Bouvry. Reconciling task assignment and scheduling in mobile edge clouds. In *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*, pages 1–6. IEEE, 2016.

[20] Ivan Farris, Tarik Taleb, Hannu Flinck, and Antonio Iera. Providing ultra-short latency to user-centric 5g applications at the mobile network edge. *Transactions on Emerging Telecommunications Technologies*, 29(4):e3169, 2018.

[21] Pawani Porambage, Jude Okwuibe, Madhusanka Liyanage, Mika Ylianttila, and Tarik Taleb. Survey on multi-access edge computing for internet of things realization. *arXiv preprint arXiv:1805.06695*, 2018.

[22] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[23] Curtis Yu, Cristian Lumezanu, Abhishek Sharma, Qiang Xu, Guofei Jiang, and Harsha V Madhyastha. Software-defined latency monitoring in data center networks. In *International Conference on Passive and Active Network Measurement*, pages 360–372. Springer, 2015.

[24] Azure windows vm sizes - compute optimized.

[25] Lei Yang, Jiannong Cao, Hui Cheng, and Yusheng Ji. Multi-user computation partitioning for latency sensitive mobile cloud applications. *IEEE Transactions on Computers*, 64(8):2253–2266, 2015.

[26] Mutsunori Yagiura and Toshihide Ibaraki. The generalized assignment problem and its generalizations.

[27] David Pisinger and Mikkel Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS Journal on Computing*, 19(1):36–51, 2007.

[28] Yu N Sotskov and Natalia V Shakhlevich. Np-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59(3):237–266, 1995.

[29] John N Hooker and Greger Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

[30] John N Hooker. Planning and scheduling by logic-based benders decomposition. *Operations Research*, 55(3):588–602, 2007.

[31] Yingyi Chu and Quanshi Xia. Generating benders cuts for a general class of integer programming problems. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 127–141. Springer, 2004.

[32] Maria A Lema, Andres Laya, Toktam Mahmoodi, Maria Cuevas, Joachim Sachs, Jan Markendahl, and Mischa Dohler. Business case and technology analysis for 5g low latency applications. *IEEE Access*, 5:5917–5935, 2017.

[33] Philipp Schulz, Maximilian Matthe, Henrik Klessig, Meryem Simsek, Gerhard Fettweis, Junaid Ansari, Shehzad Ali Ashraf, Bjoern Almeroth, Jens Voigt, Ines Riedel, et al. Latency critical iot applications in 5g: Perspective on the design of radio interface and network architecture. *IEEE Communications Magazine*, 55(2):70–78, 2017.

[34] Ruozhou Yu, Guoliang Xue, and Xiang Zhang. Application provisioning in fog computing-enabled internet-of-things: A network perspective. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 783–791. IEEE, 2018.