Distribution of Latency-Constrained Tasks in Multi-Access Edge Computing Networks

Guilherme Iecker Ricardo[1], Amal Benhamiche[1], Nancy Perrot[1], and Yannick Carlinet[1]

Orange Labs, Châtillon, France
Emails: {firstname.lastname}@orange.com

Abstract*—Multi-Access Edge Computing (MEC) paradigm has been widely studied as a potential solution to cope with the challenges emerging from new generations of mobile networks. By processing applications' data closer to the users, service providers are able to offload origin servers and their underlying network infrastructure, which consequently reduces users' experienced latency. In this paper, we consider internet-based applications with strict latency tolerance which are primarily enabled by the MEC architecture. Moreover, nodes at the edge may host application-related tasks as well as assist in their provision. We introduce the Task Distribution (TD) problem, where the objective is to maximize the overall Quality of Service (QoS) while ensuring that tasks' latency requirements are satisfied. The TD problem is modeled as an Integer Programming problem, taking into account three components: (i) tasks' priority assignment, (ii) placement and (iii) routing through the MEC network. We propose to approach the TD problem through an approximation scheme, called Shortest-Flow Approximation (SFA), which considers pre-determined network flows selected according to a Betweenness-based criterion. In our experiments, we first extract useful insights on the TD problem that help characterize its solution. Then, we evaluate SFA's performance empirically across different experimental settings, showing numerical results confirming its proximity to the optimal solution.

***Index Terms*—Edge Computing, Resource Allocation, Quality of Service, Operations Research, Combinatorial Optimization**

I. Introduction

The unprecedented increasing trend in terms of mobile connectivity and the evolution of data consumption profiles have imposed a series of challenges to the next generation of mobile networks [1]. For example, they should be able to accommodate an immense and heterogeneous number of connected devices, ranging from personal smartphones to Internet of Things (IoT) equipment (smart appliances, sensors, etc.) [2]. Not only must the infrastructure be able to handle intense traffic and data processing, it must also guarantee that specific requirements, e.g., in terms of latency and bandwidth, are met in order to enable disruptive technologies and innovative applications.

It has been widely discussed in the literature that shifting from classic Cloud Computing to the Multi-Access (Mobile) Edge Computing (MEC) paradigm may significantly contribute to address such challenges [3]. If we consider the MEC reference architecture [4], roughly speaking, the idea is to move data processing and storage from remote cloud servers to network nodes that are located at the "edge", closer to the application's users. In Figure 1, we illustrate a MEC architecture separated in three layers: (i) Devices, (ii) Far-Edge, and (iii) Near-Edge layers. Users in the Devices layer may consume application data and services directly provided by Far- and Near-Edge MEC hosts, without accessing the origin cloud servers through the core network. The distribution of computation across the MEC network offers many advantages, e.g., the possibility to offload both origin servers and the underlying network infrastructure as well as to provide data and services with smaller access overhead and end-to-end delay. However, enabling MEC systems in the architecture of new generation of mobile networks is only the first step. In order to be able to explore its full potential, MEC-powered mobile networks must be carefully designed so that computational and network resources are efficiently managed to support services at satisfying quality levels and meet other specific requirements.

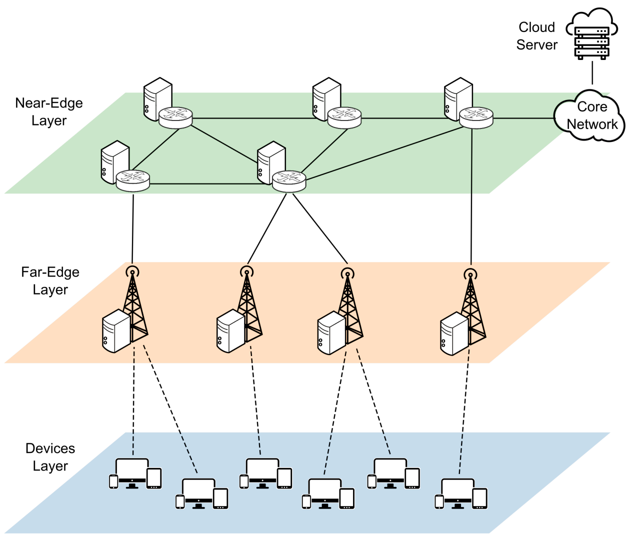

Fig. 1. Three-layer MEC architecture: Devices, Far-Edge, and Near-Edge layers. Far-Edge MEC hosts are further from the cloud servers and provide interface with the end-users. Near-Edge MEC hosts are closer to the cloud servers and provide access to the cloud servers through the core network.

The authors are with the Department of Mathematical Models for Optimization and Performance Evaluation, Orange Innovation, Châtillon, 92012 France. Corresponding author's information: http://www.guilhermeir.github.io

A. Use Case: Smart Warehouses

In the context of Industry 4.0, Smart Warehouse applications bring together all the aforementioned challenges in an effort to promote digitization and automation of industrial processes. They have been considered a key use case in most next generation architectures envisioned designs, e.g., the EU-funded DEDICAT 6G project [5]. In this work, we are particularly interested in real-time human-machine interaction services with strict latency requirements, e.g., automatically guided vehicules, timely computer-aided industrial operations (e.g., assisted with virtual and augmented reality technologies), etc. A MEC-powered network design is a key enabler of such services through back-end computation offload and opportunistic networking.

B. Related Work

MEC architectures and applications are comprehensively summarized in [6], [7]. Resource allocation on MEC systems has been addressed in numerous variants, e.g., joint minimization of task completion time and energy cost [8], cross-layer multiple resource allocation [9], auction-based blockchain applications [10], and QoS for vehicular systems [11]. Some industrial solutions include, for example, efficient routing for energy-balanced consumption in heterogeneous Industrial IoT (IIoT) [12] and energy-efficient scheduling guaranteeing delay requirements [13]. Authors in [14] propose a full model for joint optimization of service placement and routing capabilities targeting low latency applications. Finally, in [15], the authors propose an optimization model and greedy algorithm to perform resource allocation in MEC network for the edge-gaming application, considering energy saving and high intensity computation. We remark that our proposed model share structural similarities with problems of different natures. For example, in [16], the authors study the implementation cost minimization problem in OFDM two-layer networks, in which an optimization model imposes capacity as well as routing constraints.

C. Paper Contributions and Organization

We outline our main contributions and discuss the paper organization as follows:

- In Section II, we introduce the considered network structure and operation as well as the underlying assumptions.
- In Section III, we propose an optimization formulation for the Task Distribution Problem. We discuss the problem's complexity and some of its properties.
- We discuss, in Section IV, how we can use the concept of flows to re-write the Task Distribution problem. We propose to approximate the problem by reducing its size, considering only the shortest flows.
- We present our experimental setup in Section V. We try to elucidate the impact of the number of shortest flows and the filtering strategy on the quality of the approximation. We further discuss the computational advantages of applying such a technique.

II. System Model and Assumptions

Consider a MEC architecture introduced in Section I. The *Devices* layer comprises a set $\mathcal{U}$ of user equipment (UEs), each of which is associated with a single cellular Base Station (BS). Each BS is equipped with a MEC host for application-oriented data storage, processing, and routing, which we, henceforth, refer to as Points of Connection (PoCs) or "far-edge" hosts. For simplicity, we assume that PoCs are able to perfectly handle all transmissions from and to their associated UEs, so that UE-BS communication will be transparent to our model. In the transport network, BSs' traffic often converges at sink nodes/gateways powered with more abundant computational resources, which we refer to as "near-edge" MEC hosts. The *Edge* layer is composed of interconnected (e.g., through Mp3 interfaces) far- and near-edge MEC hosts, which we also refer to as the Edge network. We represent the set comprising all MEC hosts by $\mathcal{H}$.

In our model, we do not make any distinction between applications and services and refer to them simply as *tasks*. The set of all considered tasks is denoted by $\mathcal{T}$. As in some related work (e.g., [17]), we assume that each MEC host has all tasks implemented a priori. We define the Multi-Access Edge Orchestrator (MEO) as a centralized intelligence aware of the entire network's infrastructure and available resources. The system operates periodically such that each period is split into two phases: observation and management. In the observation phase, UEs exchange their requested tasks' data and potentially place requests for more tasks, which will be served only in the observation phase of the next operation period. During the whole observation phase, resources are dedicated to their assigned requests. In the management phase, the MEO knows the set of all placed requests $\mathcal{R} \subseteq \mathcal{U} \times \mathcal{T}$ and decides how to manage the network's available resources in order to accommodate all requests.

In the management phase, depending on the availability of network and computational resources, any MEC host may be assigned by the MEO to handle a request and to provide the related task's data to its UE, becoming the request's *provider*. If the closest host to the UE, i.e., its associated PoC, is not available, then another MEC host may become the provider. In this case, the MEO must also provide a *route*, i.e., a sequence of links, connecting the request's provider and its UE. Additionally, the MEO may also assign different priority levels, from a pre-determined set $\mathcal{P}$, in order to force a higher transmission rate in exchange of more computational resources utilization. The resources related to the described operation are summarized as follows:

- When a request $r \in \mathcal{R}$ is assigned to a host $h \in \mathcal{H}$ at a priority $p \in \mathcal{P}$, it consumes a fraction D_p^r of host h's total available computational resources, C_h.
- The *throughput* T_p^r, i.e., the average achieved data transmission rate, to serve request r's data at priority p must be ensured by the provider host and each other host participating in the request's routing.

In the design of MEC systems that are able to meet strict latency requirements, we have to consider the tradeoff between throughput and network latency. We assume error-free channels in the network, so the (average) throughput equals the data rate. Achieving higher throughput may provide UEs with smoother, uninterrupted experience [18], so we use it as a measure of Quality of Service (QoS). Moreover, we consider the network latency as the end-to-end delay, i.e., the time to transmit application packets from the UE to the host providing the requested task (or vice-versa) [19]. We assume that the end-to-end delay consists uniquely of queuing delay, which is fundamentally impacted by the total throughput[1]. If, on one hand, we want to provide better QoS, on the other hand, it increases the network latency (queuing delay), becoming the bottleneck for timely applications.

In summary, the MEO, aware of the request set $\mathcal{R}$, must determine a *network setup*. We focus this paper on techniques to find such a network setup, which consists of:

(i) Priority assignment: Each request $r \in \mathcal{R}$ will be assigned a priority $p \in \mathcal{P}$. Higher priorities are associated to higher throughput and higher computation. We assume that the lowest priority is related to the minimum throughput to provide a task correctly.

(ii) Request allocation: Each request $r \in \mathcal{R}$ will be provided by an edge host $h \in \mathcal{H}$. Hosts may have different amounts of available resources, e.g., near-edge hosts usually enjoy more computation power than far-edge hosts.

(iii) Request routing: For each request $r \in \mathcal{R}$, if it is provided by a host other than its UE's PoC, then the MEO must find a route through which task-related data will flow between UE and the actual provider.

We illustrate the operation of the considered system in Figure 2. In this scheme, a set of requests $\mathcal{R} = \{A, B, \ldots, G\}$ is placed by the UEs of PoCs 1 and 2. The MEO collects the requests and try to assign them to the MEC hosts in order to provide UEs with the highest throughput possible. Consider that Far-Edge hosts can only provide tasks at the lowest priority. Then, the MEO will try to distribute the tasks among the Near-Edge hosts, i.e., hosts 3, 4, and 5. In this example, due to computational limitations, Near-Edge hosts can only provide up to 2 tasks at intermediate priority and 1 task at high priority. Similarly, latency requirements are only satisfied if arcs carry data traffic of up to 2 tasks at intermediate priority and 1 task at high priority. A possible assignment is shown in the figure, where tasks are placed on each MEC host and each color is associated to a different priority level. The colored arrows indicate the tasks' data route, flowing from its provider to the PoC.

[1]As introduced in [19], the end-to-end delay has four components: transmission, propagation, processing, and queuing delays. We justify this assumption by considering a roughly homogeneous network (e.g., packets with the same size, links with equal capacity, etc.), so, in order to simplify our analysis, we can disregard sources of delay other than the queuing delay.

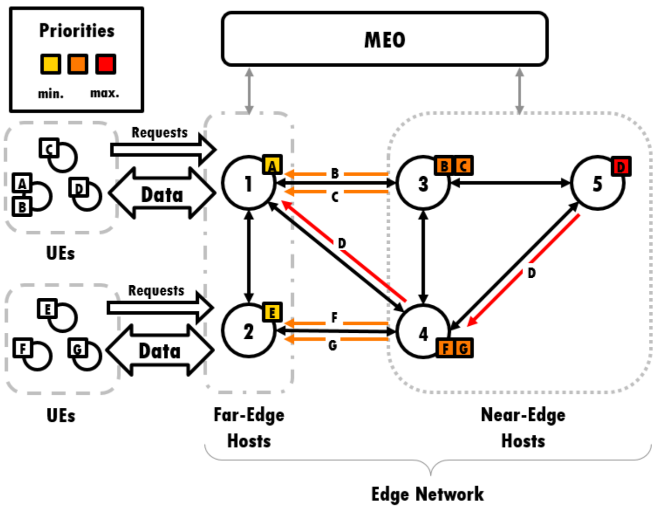

Fig. 2. Operation Example: UEs place requests for tasks to their neighboring PoCs, which are, in turn, forwarded to the MEO. To each request, the MEO assigns a provider host, a priority level, and a low-latency route between provider and PoC through the Edge network.

III. The Task Distribution Problem

Considering the system model and assumptions discussed in Section II, we introduce next the Task Distribution (TD) Problem and model it as an Integer Programming (IP) optimization problem.

A. Mathematical Formulation

First, we represent the Edge network as a graph $G = (\mathcal{H}, \mathcal{A})$, where the hosts compose the set of vertices and we consider a set of arcs $\mathcal{A}$. If there is a communication link between two network hosts, we consider one arc in $\mathcal{A}$ in each direction. Note that arcs in G do not indicate transmission direction, they rather assist in the route creation process. Then, the route for serving request $r = (u, t) \in \mathcal{R}$ may be represented as a path on graph G from the UE to its provider host. Since we assumed that UE-BS is transparent to our model, it suffices to build a route from the UE's PoC to the request's provider. Therefore, from now on, we use index u to indicate the UE's PoC, i.e., $u \in \mathcal{H}$ is a (far-edge) MEC host.

In what follows, we introduce the elements of our optimization formulation.

1) Decision Variables: To each problem component described at the end of Section II, we associate a binary variable:

(i) We model the priority assignment with variable

$$\mathbf{x} \in \mathbb{B}^{\mathcal{R} \times \mathcal{P}}, \tag{1}$$

indicating whether priority $p \in \mathcal{P}$ is assigned to request $r \in \mathcal{R}$ (i.e., $x_p^r = 1$) or not (i.e., $x_p^r = 0$).

(ii) We model the request allocation with variable

$$\mathbf{y} \in \mathbb{B}^{\mathcal{R} \times \mathcal{H}}, \tag{2}$$

indicating whether request $r \in \mathcal{R}$ is provided by node $h \in \mathcal{H}$ (i.e., $y_h^r = 1$) or not (i.e., $y_h^r = 0$).

(iii) We model the request routing with variable

$$\mathbf{z} \in \mathbb{B}^{\mathcal{R}\times\mathcal{A}}, \tag{3}$$

indicating whether request $r \in \mathcal{R}$ data is forwarded through arc $a \in \mathcal{A}$ (i.e., $z_a^r = 1$) or not (i.e., $z_a^r = 0$).

Structural Constraints: First, to every request $r \in \mathcal{R}$, we must assign exactly one priority in $\mathcal{P}$, i.e.,

$$\sum_{p\in\mathcal{P}} x_p^r = 1, \forall r \in \mathcal{R}. \tag{4}$$

Similarly, every request $r \in \mathcal{R}$ should be provided by one single node, i.e.,

$$\sum_{h\in\mathcal{H}} y_h^r = 1, \forall r \in \mathcal{R}. \tag{5}$$

Now, we define the constraints that will enforce the creation of flows (or paths) over the underlying graph $G = (\mathcal{H}, \mathcal{A})$ through which every request $r \in \mathcal{R}$ will be routed. This can be achieved by imposing the following set of *flow conservation* constraints

$$\sum_{h'\in\delta(h)} \left[z^r_{(h,h')} - z^r_{(h',h)}\right] = \mathbb{1}_{h=u(r)} - y_h^r, \quad \substack{\forall r\in\mathcal{R},\\ \forall h\in\mathcal{H}} \tag{6}$$

where $u(r)$ is the PoC of the UE which placed request r and we define $\delta(h) \subseteq \mathcal{H}$ as the set of *neighbors* of host h, i.e., host h's communicating hosts, and $\mathbb{1}_e$ indicate whether event e occurs (i.e., $\mathbb{1}_e = 1$) or not (i.e., $\mathbb{1}_e = 0$).

Computational Capacity Constraints: In what follows, we need to guarantee that a target setup can be implemented considering the available computational resources. We ensure that the *computational* utilization of each host meets its RAM/CPU capacity with

$$\sum_{r\in\mathcal{R}}\sum_{p\in\mathcal{P}} D_p^r x_p^r y_h^r \le C_h, \forall h \in \mathcal{H}, \tag{7}$$

where $D_p^r \in \mathbb{R}_+$ is the computational demand to provide the task associated to request r at priority p and $C_h \in \mathbb{R}_+$ is the total computational resources available at node h.

Latency Requirements Constraints: We model the latency requirements by imposing the following constraints

$$\sum_{a\in\mathcal{A}}\left[\alpha_a\left(\sum_{r'\in\mathcal{R}}\sum_{p\in\mathcal{P}} T_p^{r'} x_p^{r'} z_a^{r'}\right) + \beta_a\right] z_a^r \le L^r, \forall r \in \mathcal{R}, \tag{8}$$

where $T_p^r \in \mathbb{R}_+$ is the achieved throughput for request r at priority p and request r has tolerance $L^r \in \mathbb{R}_+$. We define the arc's *traffic load* as the total throughput achieved by all requests being routed through a. As in some related work (e.g., [20]), we approximate the network latency (i.e., the queuing delay) of a request r by a linear function of the traffic load on the links routing it, such that α_a and β_a are the line coefficients for arc a.[2] In summary, latency is impacted by two factors:

[2]The system is stable if the total throughput over every link is bounded by the link's capacity [19]. We assume that all links have enough capacity to accommodate the maximum priority throughput.

1) The number of links forwarding the request's data.
2) The total traffic load at those links, i.e., total throughput of all requests traversing each link.

Objective Function: Our goal is to maximize the QoS, which is simply the total throughput achieved in a given setup to serve all considered requests, i.e.,

$$\text{QoS}(\mathbf{x}) \triangleq \sum_{r\in\mathcal{R}}\sum_{p\in\mathcal{P}} T_p^r x_p^r. \tag{9}$$

Definition 1. *The Task Distribution (TD) Problem aims at finding a* valid *network setup, i.e., an assignment of* $\mathbf{x}, \mathbf{y}, \mathbf{z}$ *that satisfies constraints* (1)-(8)*, that maximizes the objective function* (9)*. In other words, the TD Problem is to solve the following optimization problem:*

Problem 1 (Task Distribution (TD) Problem)**.**

$$\begin{array}{lll}(\text{TD}) & \underset{\mathbf{x},\mathbf{y},\mathbf{z}}{\text{maximize}} & (9)\\ & \text{subject to} & (1)-(8)\end{array}$$

Proposition 1. *Problem 1 is $\mathcal{NP}$-Hard.*

Proof. Consider the following special instance of Problem 1: Each request has sufficiently large latency tolerance so that all requests may share its route's arcs at the highest achievable throughput, i.e.,

$$L^r \ge \sum_{a\in\mathcal{A}} \alpha_a |\mathcal{R}| T_{\max} + \beta_a, \forall r \in \mathcal{R},$$

where $T_{\max} \ge T_p^r, \forall r \in \mathcal{R}, \forall p \in \mathcal{P}$. Notice that latency constraints can be relaxed and the MEO is free to choose whatever route it judges to be convenient for each request. The MEO still needs to assign each request to a host satisfying their computational capacity. Therefore, in this setup, the TD Problem can be translated to the classic Multiple Knapsack (MK) Problem, which is proven to be $\mathcal{NP}$-Hard [21]. By reducing the MK Problem to this instance of the TD Problem, we show that, even in such a simple case, the TD Problem is $\mathcal{NP}$-Hard. Therefore, the TD Problem is $\mathcal{NP}$-Hard, so general instances cannot be solved in polynomial time, unless $\mathcal{P} = \mathcal{NP}$. □

B. Linearization

Even though Problem 1 is $\mathcal{NP}-Hard$, it can be linearized in order to be addressed via traditional IP solving techniques. This can be achieved by introducing auxiliary variables capturing the product of Problem 1's main variables.

Computational Capacity Constraints: We consider a first set of auxiliary variables

$$\mathbf{u} \in \mathbb{B}^{\mathcal{R}\times\mathcal{H}\times\mathcal{P}} \tag{10}$$

such that computational capacity constraints (7) are written as

$$\sum_{r\in\mathcal{R}}\sum_{p\in\mathcal{P}} D_p^r u_{h,p}^r \le C_h, \forall h \in \mathcal{H}, \tag{11}$$

where $u^r_{h,p} \triangleq x^r_p y^r_h, \forall r \in \mathcal{R}, \forall h \in \mathcal{H}, \forall p \in \mathcal{P}$. We ensure that auxiliary variables $\mathbf{u}$ are coherently associated to the main variables $\mathbf{x}, \mathbf{y}$ by enforcing the following set of constraints

$$\begin{aligned} &u^r_{h,p} \leq y^r_h, && \forall r \in \mathcal{R}, \forall h \in \mathcal{H}, \forall p \in \mathcal{P} \\ &u^r_{h,p} \leq x^r_p, && \forall r \in \mathcal{R}, \forall h \in \mathcal{H}, \forall p \in \mathcal{P} \\ &u^r_{h,p} \geq y^r_h + x^r_p - 1, && \forall r \in \mathcal{R}, \forall h \in \mathcal{H}, \forall p \in \mathcal{P}. \end{aligned} \tag{12}$$

Latency Requirements Constraints: We consider a second set of auxiliary variables

$$\mathbf{v} \in \mathbb{B}^{\mathcal{R} \times \mathcal{R} \times \mathcal{A} \times \mathcal{P}}, \tag{13}$$

such that latency requirement constraints are written as

$$\sum_{a \in \mathcal{A}} \alpha_a \left(\sum_{r' \in \mathcal{R}} \sum_{p \in \mathcal{P}} T^{r'}_p v^{r,r'}_{a,p} \right) + \beta_a z^r_a \leq L^r, \forall r \in \mathcal{R}, \tag{14}$$

where $v^{r,r'}_{a,p} \triangleq x^{r'}_p z^{r'}_a z^r_a, \forall r, r' \in \mathcal{R}, \forall a \in \mathcal{A}, \forall p \in \mathcal{P}$. We ensure that auxiliary variables $\mathbf{v}$ are coherently associated to the main variables $\mathbf{x}, \mathbf{z}$ by enforcing the following set of constraints

$$\begin{aligned} &v^{r,r'}_{a,p} \leq x^{r'}_p, && \forall r, r' \in \mathcal{R}, \forall a \in \mathcal{A}, \forall p \in \mathcal{P} \\ &v^{r,r'}_{a,p} \leq z^{r'}_a, && \forall r, r' \in \mathcal{R}, \forall a \in \mathcal{A}, \forall p \in \mathcal{P} \\ &v^{r,r'}_{a,p} \leq z^r_a, && \forall r, r' \in \mathcal{R}, \forall a \in \mathcal{A}, \forall p \in \mathcal{P} \\ &v^{r,r'}_{a,p} \geq x^{r'}_p + z^{r'}_a + z^r_a - 2, && \forall r, r' \in \mathcal{R}, \forall a \in \mathcal{A}, \forall p \in \mathcal{P} \end{aligned} \tag{15}$$

The final formulation resulting from the linearization of Problem 1 is provided in Problem 2.

Problem 2 (Linear TD (LinTD) Problem)**.**

$$\text{(LinTD)} \quad \underset{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{u},\mathbf{v}}{\text{maximize}} \quad (9) \qquad \text{subject to} \quad (1) - (6), (10) - (15)$$

Remark 1. We note that we can provide an upper bound to Problem 1 if we solve the continuous relaxation of its linear version, i.e., relaxing the integrality of variables $\mathbf{x}, \mathbf{y}, \mathbf{z}$. Then, Problem 1 can be translated to a Linear Programming (LP) problem, which can be efficiently solved in practice. Nevertheless, it is important to emphasize that Problem 2's size grows quickly, as each new auxiliary variable introduces at least three new constraints.

IV. Approximating the TD Problem

In this section, we discuss an approximation of Problem 1 based on *Decomposition* [22]. In the Decomposition technique, we tackle the original problem using a two-stage approach. In the first stage, we solve an auxiliary problem which is simpler and captures only part of the original problem's constraints. The solution of the auxiliary problem is somehow used as an input to the master problem at the second stage. With the relaxed constraints related to the auxiliary problem, the master problem will find the best solution that suits its input.

In order to discuss how we can apply the decomposition technique to solve Problem 1, we need to introduce some additional notation. First, we define a *flow* $f_{s,d} = \{(s,h), \ldots, (h', d)\}$ simply as a path on G, i.e., a sequence of arcs from a source node $s \in \mathcal{H}$ to a destination node $d \in \mathcal{H}$. Since multiple flows may exist connecting a pair of nodes s, d on G, we denote by $F_{s,d} > 0$ the total number of (s,d) flows. We denote the set of all flows in a graph G as

$$\mathcal{F} \triangleq \{f_{s,d}(k) : k = 1, \ldots, F_{s,d}, \forall s, d \in \mathcal{H}\}, \tag{16}$$

where k is a unique index for each (s,d) flow. If $s = d$, then we consider that $F_{s,d} = 1$ and $f_{s,d}(1) = \emptyset$.

TABLE I
TD PROBLEM – NOTATION SUMMARY

Symbol	Description
$\mathcal{U}$	Set of User Equipments (UEs)
$\mathcal{H}$	Set of MEC (Far- and Near-Edge) hosts
$\mathcal{R}$	Set of considered requests
$\mathcal{P}$	Set of priorities
$\mathcal{T}$	Set of tasks
x^r_p	Variable indicating if prio. p is assigned to req. r
y^r_h	Variable indicating if host h provides req. r
z^r_a	Variable indicating if arc a routes req. r
$\lambda^r_{f,p}$	Variable indicating if flow f is assigned to req. r at prio. p
D^r_p	Comp. demand to provide r with priority p
C_h	Comp. resources available at h
T^r_p	Throughput of of request r at priority p
α_a	Multiplicative line coefficient of delay in arc a
β_a	Additive line coefficient of delay in arc a
L^r	Tolerated latency for request r

A. The Flow Formulation – Master Problem

Given that the set of flows $\mathcal{F}$ was obtained in advance, we propose to reformulate Problem 1 in terms of flows. Roughly speaking, the MEO's resource management can be reduced to assigning flows to requests. Then, we integrate all problem's components into the following decision variables

$$\boldsymbol{\lambda} \in \mathbb{B}^{\mathcal{R} \times \mathcal{F} \times \mathcal{P}}, \tag{17}$$

indicating whether request r's data is routed using flow $f \in \mathcal{F}$ at priority p (i.e., $\lambda^r_{f,p} = 1$) or not (i.e., $\lambda^r_{f,p} = 0$). Note that, when selecting a flow for a request, we are implicitly selecting (i) the provider host and (ii) the sequence of arcs in G through which request's data is sent.

We enforce that a single flow, whose source node is the request's PoC, is assigned to each request at exactly one priority by imposing constraints

$$\sum_{f \in \mathcal{F}} \sum_{p \in \mathcal{P}} \mathbb{1}_{s(f)=u(r)} \lambda^r_{f,p} = 1, \forall r \in \mathcal{R}, \tag{18}$$

where $s(f)$ denotes the source node of flow f. The computational capacity constraints can be translated to

$$\sum_{r \in \mathcal{R}} \sum_{p \in \mathcal{P}} \mathbb{1}_{d(f)=h} D^r_p \lambda^r_{f,p} \leq C_h, \forall h \in \mathcal{H}, \tag{19}$$

where $d(f)$ denotes the destination node of flow f. Similarly, the latency requirements constraints are re-defined as follows

$$\sum_{\substack{f \in \mathcal{F} \\ p \in \mathcal{P}}} \lambda^r_{f,p} \sum_{a \in f} \left[\alpha_a \left(\sum_{\substack{r' \in \mathcal{R}, f' \in \mathcal{F} \\ p' \in \mathcal{P}}} \mathbb{1}_{a \in f'} T^{r'}_{p'} \lambda^{r'}_{f',p'} \right) + \beta_a \right] \underset{\forall r \in \mathcal{R}}{\leq L^r,} \tag{20}$$

Finally, the objective function is

$$\text{QoS}_{\mathcal{F}}(\boldsymbol{\lambda}) \triangleq \sum_{r\in\mathcal{R}} \sum_{f\in\mathcal{F}} \sum_{p\in\mathcal{P}} T_p^r \lambda_{f,p}^r. \tag{21}$$

We formulate the TD Problem in terms of flows as follows:

Problem 3 (Flow-Based TD (FTD) Problem)**.**

$$\begin{array}{lll} \text{(FTD)} & \underset{\boldsymbol{\lambda}}{\text{maximize}} & (21) \\ & \text{subject to} & (17) - (20) \end{array}$$

Proposition 2. *If $\mathcal{F}$ contains all possible flows on graph G, then problems 1 and 3 are equivalent, i.e., if $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ and $\boldsymbol{\lambda}^*$ are their respective optimal solutions, then*

$$\text{QoS}(\mathbf{x}^*) = \text{QoS}_{\mathcal{F}}(\boldsymbol{\lambda}^*).$$

The intuition behind Proposition 2 is that every flow in $\mathcal{F}$ satisfies the flow conservation constraints (6). Therefore, by building a solution for Problem 3 we are implicitly finding a valid assignment for variables $\mathbf{z}$ from the original formulation.

Proposition 3. *Problem 3 is $\mathcal{NP}$-Hard.*

The proof of Proposition 3 follows the same lines as the proof for Proposition 1. We consider a setup for Problem 3 with sufficiently large latency requirements, and note that the MK Problem can be reduced to such a particular instance. Therefore, Problem 3 is $\mathcal{NP}$-Hard.

Remark 2. Even though Problem 3 can be linearized (similarly to what was done for Problem 1), the size of set $\mathcal{F}$ grows rapidly with the size of the network G, which may easily lead to a computationally intractable problem size. However, we remark that the structure of Problem 3 suits the Column Generation framework, which may be able to find good approximations in practice [22].

We discuss next how to further exploit the flow formulation to approximate the TD Problem.

B. Shortest-Flow Approximation

The main idea of the Shortest-Flow Approximation (SFA) is to consider that the optimal solution of the TD Problem is primarily built upon shortest flows. This is reasonable because the number of links routing a task's data is the first factor that impacts the latency (recall the discussion on Latency Constraints in Section III).

For a given network G, we define the set of shortest flows from source node s to destination node d as follows

$$\mathcal{SF}_{s,d} \triangleq \left\{ f_{s,d} \in \mathcal{F} : \forall f'_{s,d} \in \mathcal{F}, |f_{s,d}| \leq |f'_{s,d}| \right\}, \tag{22}$$

where $|f|$ is the size of set f, i.e., the number of arcs in flow f. Additionally, we consider that the set of shortest flows can be limited to a maximum number $K \geq 1$ of equivalent flows between every pair of source-destination nodes. We define the set of K-shortest flows between nodes s, d as $\mathcal{SF}_{s,d}^K$. Notice that, because a pair of source-destination nodes may have a number of shortest flows smaller than K, then $|\mathcal{SF}_{s,d}^K| = \min(K, |\mathcal{SF}_{s,d}|)$. Finally, we denote the set of the K-shortest flows for all pairs of nodes in G as $\mathcal{SF}^K = \{\mathcal{SF}_{s,d}^K : \forall(s,d) \in \mathcal{H}\}$. In SFA, we build and solve Problem 3 using the set $\mathcal{SF}^K$ of K-shortest flows instead of the entire set of flows $\mathcal{F}$.

Remark 3. There are two immediate consequences of applying SFA: (i) Set $\mathcal{SF}^K$ can be efficiently obtained, e.g., via Dijkstra algorithm [23] and (ii) we consider a significantly smaller problem. However, by limiting set $\mathcal{F}$, we provide the solver with less flows to integrate solution candidates, which often results in sub-optimal solutions. In summary, if $\boldsymbol{\lambda}^*$ and $\boldsymbol{\lambda}^*_{\text{SFA}}$ are the optimal solutions of Problem 3 considering the entire set of flows $\mathcal{F}$ and considering set $\mathcal{SF}^K$ of the K-shortest flows, respectively, then,

$$\text{QoS}_{\mathcal{F}}(\boldsymbol{\lambda}^*) \geq \text{QoS}_{\mathcal{SF}_K}(\boldsymbol{\lambda}^*_{\text{SFA}}).$$

Henceforth, we refer to SFA limited by the K-shortest flows as K-SFA. The K-SFA's ability to achieve solutions close to the optimal (considering all flows) depends primarily on two factors, which we discuss next.

1) Selecting K: The larger is K, the better may be the approximation. For large enough values of K, the closer set $\mathcal{SF}^K$ gets to $\mathcal{SF}$, which provides the maximum number of shortest-flow candidates and, in turn, the best approximate solution. However, it is still not guaranteed that the real optimal solution can be achieved, given that optimal flows are not necessarily the shortest ones (i.e., optimal flows may reside within $\mathcal{F} \setminus \mathcal{SF}$). Moreover, the optimization problem's size grows rapidly with K. Therefore, we must find the sweet spot in this tradeoff for K, in order to obtain satisfactory approximations from computationally treatable optimization problems.

2) Selecting the K-shortest flows: In the case where $|\mathcal{SF}_{s,d}| > K$, the strategy used to choose the K-shortest flows among this subset to constitute set $\mathcal{SF}^K$ may also impact the quality of K-SFA's solution. If, in a given set $\mathcal{SF}^K$, flows tend to overshare the same arc, then sub-optimal assignments will integrate the solution in order not to violate the latency requirement constraints. The idea is that we must choose the K-shortest flows that provide the largest number of feasible assignment candidates possible, in order to reduce the optimality gap. Inspired by [24], we capture the notion of potential traffic load of a given arc by using its edge-betweenness centrality. In short, the edge-betweenness of a link a is the fraction of shortest flows traversing it (see [25] for a more detailed discussion). We propose to rank flow f's "idleness" level according to the following metric

$$B(f) \triangleq \sum_{a\in f} \left(1 - \frac{1}{|\mathcal{SF}|} \sum_{f\in\mathcal{SF}} \mathbb{1}_{a\in f'} \right). \tag{23}$$

As the total traffic load traversing each arc is the second factor impacting latency, choosing the K-shortest most-idle flows provides a space of solution candidates with higher degree of freedom, potentially reducing the optimality gap.

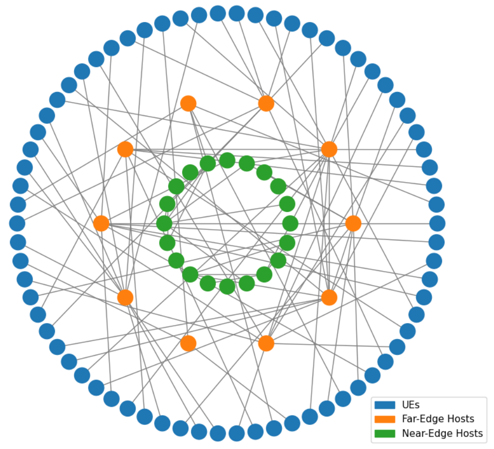

Fig. 3. Experimental network topology consisting of 100 nodes: 70 UEs, 10 Far-edge nodes (PoCs) and 20 Near-edge hosts.

TABLE II
EXPERIMENTAL PARAMETERS

Computational Capacity	Far-Edge Hosts		Near-Edge Hosts
	$C_h = 32.0$ GB		$C_h = 64.0$ GB
Latency Tolerance	$L^r \sim$ UNIFORM$[50.0, 150.0]$ ms		
Priorities Attributes	$p = 1$	$p = 2$	$p = 3$
	$T_p = 10.0$ Mbps	$T_p = 20.0$ Mbps	$T_p = 30.0$ Mbps
	$D_p = 1.0$ GB	$D_p = 2.0$ GB	$D_p = 4.0$ GB
Arcs' Latency Parameters	$\alpha_a \sim$ UNIFORM$[0.0, 5.0]$	$\beta_a \sim$ UNIFORM$[0.0, 200.0]$	

V. EXPERIMENTAL RESULTS

In this section, we study the performance of K-SFA for specific values of K and how the selection strategy based on edge-betweenness may affect the quality of the approximation. In our experiments, we consider the Berlin topology: a cellular network consisting of 10 PoCs located according to the positions of T-mobile BSs in Berlin extracted from [26]. Moreover, we consider that the PoCs communicate to a randomly generated (connected) network of 20 near-edge hosts. There is a set of 70 UEs, each of which randomly connected to a PoC. We plot in Figure 3 the network topology used in our experiments.

We consider that values of computational utilization and throughput only depends on the priority, i.e., $D_p^r = D_p, \forall r \in \mathcal{R}, \forall p \in \mathcal{P}$ and $T_p^r = T_p, \forall r \in \mathcal{R}, \forall p \in \mathcal{P}$. There are 3 priorities, such that, $\forall r \in \mathcal{R}$, computational demands are $D_1^r = 1.0$ GB, $D_2^r = 2.0$ GB, and $D_3^r = 4.0$ GB, and the throughput levels are $T_1^r = 10.0$ Mbps, $T_d^r = 20.0$ Mbps, and $T_3^r = 30.0$ Mbps. Far- and Near-edge hosts have $C_h = 32.0$ GB and $C_h = 64.0$ GB of RAM, respectively. Requests' UEs are chosen uniformly at random and their related latency is also uniformly selected from the interval $[50.0, 150.0]$ ms. All these values are consistent with the literature (e.g., [15]). For each arc in the network, the latency model parameters are randomly chosen from a fixed interval. Specifically, $\forall a \in \mathcal{A}$, α_a is chosen from interval $[0.0, 5.0]$ and β_a is selected within $[0.0, 200.0]$. We summarize the parameters in Table II.

Given the described experimental setup, we generate the request set such that PoCs can provide all their UEs' requests at the lowest priority possible. This guarantees that the optimization problem always have a feasible solution, which we refer to as the *trivial* solution. We propose to compare the approximation of K-SFA with different values of K and with different selection strategies. We also define the lower bound (LB) as the trivial solution where all requests are deployed at the UEs' PoCs at the lowest priority. We compare K-SFA and LB with Problem 2's upper bound (UB), i.e., the solution of its continuous relaxation.

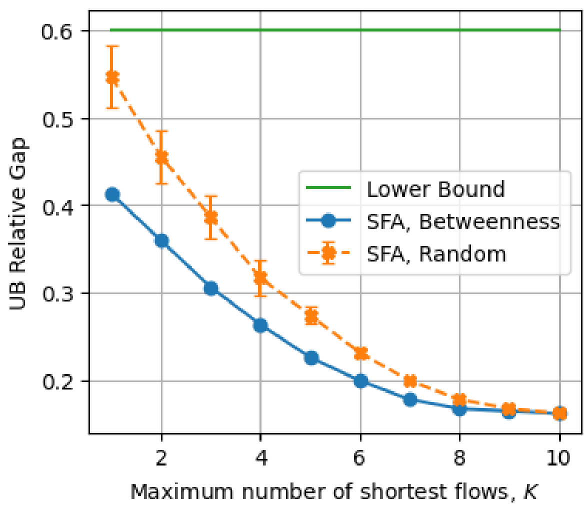

Fig. 4. Upper bound gaps relative to (i) lower bound, (ii) K-SFA (Betweenness), (iii) K-SFA (Random), versus the maximum number of flows K.

In our first set of experiments, we propose to observe the impact of K and the different shortest flows selection strategies for a set of 150 requests. For each $K \in \{1, \dots, 10\}$, we solve[3] an instance of Problem 3 with a set of K-shortest flows filtered (i) uniformly at *random*, $\mathcal{SF}_R^K$, and (ii) using *betweenness* strategy, $\mathcal{SF}_B^K$. For each K, we generate 30 problem instances with randomly chosen $\mathcal{F}_{K,R}$ and average their relative UB gaps. Finally, we calculate the relative UB gap for the betweenness strategy. We show the first experimental results in Figure 4.

We first note that the performance gap between random and betweenness (of around 27%) has its largest value when $K = 1$. At this point, both approximations have large UB gaps and as K increases, (i) K-SFA tends to provide better results (i.e., closer to the UB), in general, until it reaches a convergence point at $K = 8$, and (ii) the performance gap between random and betweenness decreases. Notice that K-SFA (regardless of the selection strategy) may achieve results up to 10% far from the problem's UB for when $K = 10$. At this point, the selection strategy has little impact because, for $K = 10$ in this network, we are considering almost all shortest flows. This hypothesis is also supported by the fact that the

[3]The optimization model was developed with PyOMO [27] and solved using IBM CPLEX solver [28].

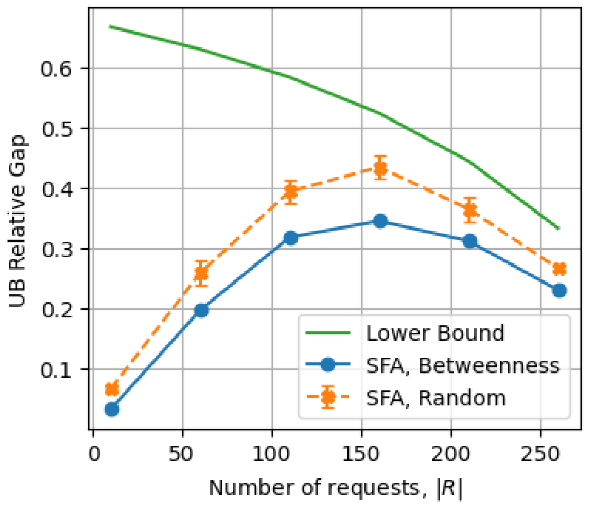

Fig. 5. Upper bound gaps relative to (i) lower bound, (ii) K-SFA (Betweenness), (iii) K-SFA (Random), versus the total number of requests $|\mathcal{R}|$.

standard deviation for the random selection is close to zero for $K > 5$. This facts altogether corroborate our conclusion that the approximation provided by K-SFA is at most 10% from the optimal solution in this setup. This means that the optimal solution is composed of some non-shortest flows.

We show on Table III the completion time, in seconds, for different stages of our solutions. In general, the auxiliary problem's solution time is considerably smaller than the the master problem's solution time. As we briefly mentioned in Section IV, the shortest paths can be efficiently obtained. The largest overhead is due to betweenness calculation time, which is still a negligible increase in comparison with the master problem solution time. Experiments were run on a computer with an Intel(R) Xeon(R) CPU E5-2650 v2 running at 2.60 GHz using 256 GB of RAM and Linux Debian 6.3.0 operating system.

In our second set of experiments, we fix $K = 2$ for K-SFA and solve the problem for requests sets of different sizes. Once again, we plot K-SFA's (both Betweenness and Random) and LB's gap relative to the problem's UB. We observe that, for $|\mathcal{R}| = 10$, the network can accommodate almost all the request at the highest priority and K-SFA was capable of approximating this solution. As we increase the number of requests, the larger is the gap with the UB. This trend can be explained by the fact that having to handle more requests require more complex solutions not only based on shortest paths. In other words, having less busy shortest flows improves the result (Betweenness is still performing better than Random), but it gets a less significant advantage, given that non-shortest flows are more present in the solution as the problem grows in complexity (with more requests to handle). Interestingly, after reaching its peak at $|\mathcal{R}| = 160$, the curves tend to decrease and to get close to each other. We assign this trend to the increasing excess of requests (i.e., those which

TABLE III
RUN TIME FOR DIFFERENT SOLUTION APPROACHES.

	Original Formulation	K-Shortest Flow Approximation				
		K=2	K=4	K=6	K=8	K=10
Auxiliary Problem Time	-	912	1122	1507	2355	2880
Main/Master Problem Time	-	18352	38355	67355	80561	110803
Total Solution Time	7458	19264	39477	68862	82916	113683

cannot be accommodated in the network at all), which end up having trivial assignments (PoC with smallest priority). This excess tends to dominate the portion of requests that are optimally assigned as $|\mathcal{R}|$ grows large, which, in turn, makes the relative UB gap reduce.

VI. Conclusions and Future Work

The TD Problem captures fundamentally the primary requirements of latency-sensitive systems based on MEC architectures. In this paper, we propose a linear integer programming model for the TD Problem. We model requests' latency as the classic queuing delay and approximate it as a linear function of the total achieved throughput in the participating communication links. We approach the problem using the K-SFA and we introduce a shortest flow selection strategy based on edge-betweenness centrality. In our experiments, we could observe our technique's performance and conclude that K-SFA can achieve satisfactory results even for relatively small values of K. The results presented in this paper will pave the way to more sophisticated modeling and more efficient approximations. Furthermore, the framework can be expanded to consider a mathematical modeling for the robust optimization of the expected QoS under random requests and accounting for uncertainties.

Acknowledgement

This work was supported by the European Union H2020 Project DEDICAT 6G under grant no. 101016499.

References

[1] C. De Alwis, A. Kalla, Q.-V. Pham, P. Kumar, K. Dev, W.-J. Hwang, and M. Liyanage, "Survey on 6g frontiers: Trends, applications, requirements, technologies and future research," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 836–886, 2021.
[2] CISCO, "Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021 white paper," CISCO, Tech. Rep., Feb 2017.
[3] Y. Zhang, "Mobile edge computing for beyond 5g and 6g," in *Mobile Edge Computing*. Springer, 2022, pp. 37–45.
[4] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, "Mobile-edge computing architecture: The role of mec in the internet of things," *IEEE Consumer Electronics Magazine*, vol. 5, no. 4, pp. 84–91, 2016.
[5] V. Stavroulaki, E. C. Strinati, F. Carrez, Y. Carlinet, M. Maman, D. Draskovic, D. Ribar, A. Lallet, K. Mößner, M. Tosic, M. Uitto, S. A. Hadiwardoyo x, J. Marquez-Barja, E. Garrido, M. Stamatelatos, K. Sarayeddine, P. Sánchez Vivas, A. Mämmelä, and P. Demestichas, "Dedicat 6g - dynamic coverage extension and distributed intelligence for human centric applications with assured security, privacy and trust: from 5g to 6g," in *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, 2021, pp. 556–561.

[6] I. Sittón-Candanedo, R. S. Alonso, J. M. Corchado, S. Rodríguez-González, and R. Casado-Vara, "A review of edge computing reference architectures and a new global edge proposal," *Future Generation Computer Systems*, vol. 99, pp. 278–294, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X1930264X

[7] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, vol. 98, pp. 289–330, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762118306349

[8] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2019.

[9] P. Wang, C. Yao, Z. Zheng, G. Sun, and L. Song, "Joint task assignment, transmission, and computing resource allocation in multilayer mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2872–2884, 2019.

[10] Y. Jiao, P. Wang, D. Niyato, and Z. Xiong, "Social welfare maximization auction in edge computing resource allocation for mobile blockchain," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[11] Y.-J. Ku, P.-H. Chiang, and S. Dey, "Quality of service optimization for vehicular edge computing with solar-powered road side units," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, 2018, pp. 1–10.

[12] P. G. V. Naranjo, M. Shojafar, A. Abraham, and E. Baccarelli, "A new stable election-based routing algorithm to preserve aliveness and energy in fog-supported wireless sensor networks," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2016, pp. 002 413–002 418.

[13] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE internet of things journal*, vol. 3, no. 6, pp. 1171–1181, 2016.

[14] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Service placement and request routing in mec networks with storage, computation, and communication constraints," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1047–1060, 2020.

[15] F. Spinelli, A. Bazco-Nogueras, and V. Mancuso, "Edge gaming: A greening perspective," *Computer Communications*, vol. 192, pp. 89–105, 2022.

[16] A. Benhamiche, A. Mahjoub, N. Perrot, and E. Uchoa, "Capacitated multi-layer network design with unsplittable demands: Polyhedra and branch-and-cut," *Discrete Optimization*, vol. 35, p. 100555, 2020.

[17] M. Charikar, Y. Naamad, J. Rexford, and X. K. Zou, "Multi-commodity flow with in-network processing," in *Algorithmic Aspects of Cloud Computing*, Y. Disser and V. S. Verykios, Eds. Cham: Springer International Publishing, 2019, pp. 73–101.

[18] M. Taha, L. Garcia, J. M. Jimenez, and J. Lloret, "Sdn-based throughput allocation in wireless networks for heterogeneous adaptive video streaming applications," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2017, pp. 963–968.

[19] D. Bertsekas and R. Gallager, *Data networks*. Athena Scientific, 2021.

[20] P. Bonami, D. Mazauric, and Y. Vaxes, "Maximum flow under proportional delay constraint," *Theoretical Computer Science*, vol. 689, pp. 58–66, 2017.

[21] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[22] J. Desrosiers and M. E. Lübbecke, "A primer in column generation," in *Column generation*. Springer, 2005, pp. 1–32.

[23] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[24] Y. Zhuo, Y. Liang, Y. Huang, Y. Cao, J. Nie, and Y. Qi, "A routing strategy based on the betweenness centrality for multi-layers complex networks," in *2021 IEEE 9th International Conference on Information, Communication and Networks (ICICN)*, 2021, pp. 384–388.

[25] U. Brandes, "On variants of shortest-path betweenness centrality and their generic computation," *Social Networks*, vol. 30, no. 2, pp. 136–145, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378873307000731

[26] "Openmobilenetwork." [Online]. Available: openmobilenetwork.org/

[27] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson, and D. L. Woodruff, *Pyomo–optimization modeling in python*, 3rd ed. Springer Science & Business Media, 2021, vol. 67.

[28] I. I. Cplex, "V12. 1: User's manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.