



**Università degli Studi Mediterranea di Reggio Calabria**  
Archivio Istituzionale dei prodotti della ricerca

In-network Placement of Reusable Computing Tasks in an SDN-based Network Edge

This is the peer reviewed version of the following article:

*Original*

In-network Placement of Reusable Computing Tasks in an SDN-based Network Edge / Amadeo, Marica; Campolo, Claudia; Lia, Gianmarco; Molinaro, Antonella; Ruggeri, Giuseppe. - In: IEEE TRANSACTIONS ON MOBILE COMPUTING. - ISSN 1536-1233. - 23:2(2024), pp. 1456-1471. [10.1109/TMC.2023.3237765]

*Availability:*

This version is available at: <https://hdl.handle.net/20.500.12318/132532> since: 2024-09-12T12:54:05Z

*Published*

DOI: <http://doi.org/10.1109/TMC.2023.3237765>

The final published version is available online at: <https://ieeexplore.ieee.org/document/10018567>

*Terms of use:*

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website

*Publisher copyright*

This item was downloaded from IRIS Università Mediterranea di Reggio Calabria (<https://iris.unirc.it/>) When citing, please refer to the published version.

(Article begins on next page)

10 May 2026

# In-network Placement of Reusable Computing Tasks in an SDN-based Network Edge

Marica Amadeo, Claudia Campolo, *Senior Member, IEEE*, Gianmarco Lia, Antonella Molinaro, *Senior Member, IEEE*, and Giuseppe Ruggeri, *Member, IEEE*

**Abstract**—Edge computing is aimed to support compute-intensive data-hungry interactive applications which can hardly run on resource-constrained consumer devices and may suffer from running in the cloud due to the long data transfer delay. The edge network nodes' heterogeneous and limited (compared to the cloud) capabilities make the computing task placement a challenge. In this paper, we propose a novel *in-network* task placement strategy aimed at minimizing the edge network resources usage. The proposal specifically accounts for *time-limited reusable* computing tasks, i.e., tasks whose output can be cached to serve requests from different consumers for a certain time. Caching such results, during their time validity, achieves the twofold benefit of reducing the service provisioning time and improving the edge resource utilization, by avoiding redundant computations and data exchange. The devised strategy is implemented as a network application of a Software-defined Networking Controller in charge of overseeing the edge domain. We formulate the optimal task placement through an integer linear programming problem, and we define an efficient heuristic algorithm that well approximates the solution achieved through a standard optimal solver. Achieved results show that the proposal successfully meets the targeted objectives in a wide variety of simulated scenarios, by outperforming benchmark solutions.

**Index Terms**—Edge computing, Software-defined Networking, Compute reuse, Task placement

## 1 INTRODUCTION

Task offloading to the edge is gaining momentum as a solution aimed at satisfying the demands of compute-intensive and latency-sensitive applications, such as Augmented Reality (AR), on-line games, cognitive assistance, autonomous driving [1]. In the more general case, offloading a computing task to the edge requires (i) an edge node, with available processing resources, to be selected as task *executor*, (ii) application data to be transferred from the source(s) to the identified edge node as an input for the required computation, (iii) a software program, i.e., the computing function available at the executor to perform the given data processing. For instance, for an AR application, a

picture should be provided as input and processed through a program implementing an object detection task.

The limited edge nodes' capabilities compared to the cloud, coupled with the peculiarities of the aforementioned applications entail a *smart* and *judicious orchestration* to decide *where* placing the execution of computing tasks. This is much more challenging when edge nodes are not limited to one or few purpose-built edge servers, rather they encompass multiple network nodes with largely heterogeneous capabilities, such as the ones composing a campus network, the points of presence of a Telco provider, or the backhaul segment of a mobile network. Such a trend is fueled by recent initiatives pushing *in-network computing*, like the IETF Computing in the Network (COIN) Research Group [2], and by recent research works [3], [4].

Multiple task placement solutions have been proposed so far in the literature targeting objectives like the reduction of energy consumption and computation latency, while meeting delay constraints, and saving network bandwidth [5], [6], [7]. In addition, recently, caching the program and/or the output (i.e., the result) of the computing task at the edge has been recognized as an effective solution to further reduce the delay experienced by the user, the energy consumption, and the edge resources (i.e., bandwidth, processing) utilization [5], [8], [9].

While the majority of the cited works have focused on caching the program, which is typically reusable [9], [10], less attention has been devoted on caching the computation result. However, as evaluated in [11], this operation, referred to as *compute reuse*, dramatically reduces the amount of repeated data transmissions and edge computation and may result in up to 50x lower task completion times than the case with no reuse. A variety of applications can benefit from either partial or full compute reuse at the edge. AR applications for users attending the same event (e.g., music concert) or visiting the same place can exploit (part of) the same computation result. For instance, several visitors in a museum providing pictures of the same painting, from different angles or with different shades, may request the same computational task (e.g., the history of that painting) [12]. Just to provide a figure, the Louvre museum attracts about 15K visitors per day, containing more than 380K objects and displaying 35K works of art. This substantiates the chance for massive amounts of compute reuse [13]. Mobile gaming applications like a chess game, for the limited set of First-

- M. Amadeo, C. Campolo, G. Lia, A. Molinaro, G. Ruggeri are with the Department of Information Engineering, Infrastructure and Sustainable Energy, University of Reggio Calabria, Italy, and with the Italian National Inter-University Consortium for Telecommunications (CNIT).  
E-mail: name.surname@unirc.it.
- Antonella Molinaro is also with the Laboratory of Signals and Systems, CentraleSupélec-CNRS-University Paris Sud, University Paris Saclay Gif-sur-Yvette, France.

Move Configuration (FMC), can reuse the responses (i.e., countermoves) previously computed in front of the same configurations and moves of the players [5], [11].

In many cases, compute reuse is also characterized by a *temporal validity*: results are useful only for a limited time interval and then, they can be evicted from the cache. For instance, event-based processing [14] can be highly popular in the sense that it can provide results that are highly requested (and therefore, reusable) by distinct consumers but only for a limited time after which the output of the computation gets meaningless, because the spatio-temporal conditions in which input data are retrieved change. Multiple vehicles at an intersection may request the obstacles detected in the area and can be served by the same executed object detection task, but only if approaching the intersection in very close time instants. In the following, we will refer to as *time-limited reusable computing tasks* those tasks whose output is reusable for a certain time.

The main goal of this paper is to devise a novel placement strategy of *time-limited reusable computing tasks* in a network edge domain, aimed at *minimizing the edge resources usage*, by both reducing the amount of intra-domain traffic and better exploiting the edge nodes' computing resources. In our proposal, without loss of generality, we refer to a Software-defined Networking (SDN)-based approach to enforce the conceived centralized task placement strategy. Such a choice is aligned with the recent literature targeting a *unified network-edge service provisioning* [15], [4] and ensures the joint optimization of network and computing edge resources. This work goes beyond the state-of-the-art and, notably, it provides the following main contributions laying the foundations of upcoming sixth generation (6G) networks:

- We consider an SDN-based network edge domain and formulate an optimization problem for the in-network placement of time-limited reusable computing tasks through an Integer Linear Programming (ILP) problem. The objective is to identify the candidate task executors among the SDN edge network nodes, in order to reduce the amount of intra-domain traffic while minimizing the edge resource usage and improving the Quality of Service (QoS) experienced by end-users, also through the reuse of the output of the tasks.
- We design, implement and validate a new heuristic algorithm which provides an approximate solution to the formulated problem, but in a significantly shorter time compared to the one achieved through a standard optimization solver. The formulated placement strategy is also proven to be solvable through a near-optimal well-established heuristic algorithm in the literature. Thus, the study overall represents a valuable contribution for the design of efficient and cost-effective edge frameworks.
- We evaluate the proposed placement strategy, when compared to benchmark solutions, under different load settings in terms of valuable metrics, among others, the amount of exchanged data for the task execution, the task computation delay, while assessing the impact of the compute reuse under different

popularity patterns and output time validity settings.

The remainder of this paper is organized as follows. Section 2 provides background material on reusable computing tasks and SDN-based solutions for task allocation at the edge. The system model is described in Section 3. The optimization problem is formulated in Section 4, whereas the proposed heuristics is reported in Section 5. Section 6 summarizes the main results of the conducted evaluation study, before concluding in Section 7.

## 2 BACKGROUND AND MOTIVATIONS

### 2.1 Reusable computing tasks

**State-of-the-art.** Related work on edge computing typically assumes that computing tasks requested from different clients are distinct from each other and they need to be executed independently [16]. To improve the performance, however, the research community has started to consider the possibility of caching, and therefore reusing, the program, the input data and/or the computation result.

The benefits of caching the program (sometimes referred to as service caching [10], [17], [18]) are largely recognized in the recent literature [9], [10]. Moreover, other works have considered the possibility of caching both the input data and the program, e.g., in order to reduce the service latency and the energy consumption of mobile devices [19].

At the same time, however, there are multiple cases where computation results are fully or partially reusable. Caching outputs of computing tasks in order to reduce redundant computations is referred to as *computation content caching* in [9] and as *compute reuse* in [12], [20]. For instance, a processed gaming scene may be requested (almost) synchronously by individual players in case of a mobile online game [21]. Of course, caching the result and reusing it would reduce both the computation load on the edge nodes and the service provisioning latency experienced by the players. Also, distinct clients in close proximity may issue requests exhibiting spatio-temporal locality, e.g., in AR applications, visitors/passengers in the same area within a museum/airport, may request, nearly simultaneously, the same processed AR output or part of it [5]. In such cases, compute reuse is viable because of input similarity, which stems from the same contextual information being captured, such as landmarks, road signs [22].

Compute reuse may be particularly helpful for edge nodes with limited capabilities, which may save precious resources otherwise wasted to redundantly execute particularly compute-intensive tasks, like Deep Neural Network (DNN) inference [23]. In this context, the work in [21] considers a multi-user single-edge server environment and presents an energy-efficient task offloading scheme with compute reuse subject to deadline constraints. Conversely, the work in [11] presents an empirical approach to motivate the need for compute reuse across clients and stakeholders. By implementing three common edge computing applications, namely matrix multiplication, face detection and chess, the authors quantify the gain of edge computing systems with compute reuse, versus those that do not apply this feature. They find that, thanks to the reuse, task completion times and CPU usage are significantly reduced.

From a networking perspective, in presence of a distributed edge computing infrastructure, a big challenge of compute reuse is to recognize distinct requests for the same task and forward them to the same edge node that cached the result. This also implies the definition of specific mechanisms that uniquely name and describe the computing tasks [3]. As recognized in [8], these aspects can be addressed by Information Centric Networking (ICN) paradigms, which directly name contents and computations at the network layer and implement routing-by-name mechanisms, instead of traditional IP-based forwarding. In [12], the authors present ICedge, a first network-based framework that leverages ICN to offer name-based compute reuse abstractions in a fully distributed manner. The focus is on the definition of standard naming conventions and name-based forwarding rules, without considering the optimal orchestration of tasks.

**Contribution.** Without loss of generality, it is worth observing that the output of a reusable computing task can have a *limited time validity*, ranging from hundreds of milliseconds, up to minutes or even days, after which it becomes useless. This is a common feature for tasks that take as input time-varying data, like environmental parameters or context-aware information tracked either by a Google Lens app of a tourist walking around a city or a surveillance camera in a building. In the case of a multi-player mobile gaming application, different players may need to visualize the same scene rendered by the server. The output can be computed once and cached to serve requests coming close to each other in time. The output becomes useless after the maximum tolerable latency for a player (ranging from 100ms up to 1s, according to the type of interactive game) expires [24]. The number of players that can reuse the same output varies according to the type of game and the maximum number of players that can be hosted on the same server, from tens to hundreds, up to thousands for massive multi-player games [25].

On the other hand, in case of a chess game, responses to an already computed FMC can be reused for a significantly longer time for different players [11]. Similarly, the request for the history of a famous landmark issued by a Google Lens app may satisfy multiple consumers over a long time period [22].

The time validity can vary over time for the same kind of task. For example, in peak periods, the time validity of the output of a road safety-related task is short, whereas in off-peak periods, since the environment changes slowly, the time validity may be longer.

However, to the best of our knowledge and mainly due to the infancy of the topic, the impact of the time validity of the computation result on the achieved performance has not been yet investigated, which motivates our work to properly account for it in the edge placement decision of reusable tasks.

## 2.2 SDN-driven Edge Computing

**Basics.** SDN revolutionizes the networking realm by decoupling the control plane from the data plane and moving it to a logically centralized entity, the Controller. On top of it, network applications (e.g., routing, load balancing) are abstracted from the underlying network nodes, which become

simple forwarding elements. Thanks to the network-wide view kept at the Controller about link status and network nodes under its control, the implementation of sophisticated mechanisms for traffic control and resource management can be facilitated. This is possible through the injection of proper rules in the flow table of nodes overseen by the Controller.

**State-of-the-art.** Recently, SDN has been also considered as key enabler for task orchestration in edge computing scenarios [15].

A task offloading strategy in a software-defined ultra-dense cellular network is presented in [26], where base stations are augmented with edge computing capabilities. By considering the network status from a global perspective, an SDN Controller instructs the mobile devices about whether offloading a task or executing it locally and, in the former case, to which base station, so to minimize the task completion time. With similar purposes, an SDN edge framework for task allocation is proposed in [27] that targets vehicular networks. There, the SDN Controller tracks the status (i.e., how much memory and CPU is available) of distributed edge servers and collects the task information from vehicles. According to this information, the Controller instructs the vehicles to compute tasks locally or offload them to a nearby edge server. In the same scenario, the work in [28] leverages a vehicle mobility analyzer at the Controller to predict the communication time between the vehicle and the nearby edge servers. A greedy algorithm is then proposed to offload the task to the *best edge server*, i.e., the one that maximizes the success probability of total task execution within a completion time limit.

In [29], the focus is on computing tasks requiring multiple input data from heterogeneous end devices in an edge mesh scenario, i.e., a wirelessly-connected collaborative edge network. The authors consider traffic congestion and network bandwidth consumption when transmitting the input data and study the task allocation problem to jointly schedule tasks and network flows with the objective of minimizing the service completion time. A multistage greedy adjustment algorithm, implemented at the SDN Controller, is proposed to support the placement of tasks according to the bandwidth of the flows.

In [30] an SDN-based framework is proposed to optimally place edge clouds on access points and to schedule computing tasks by ensuring the minimum total energy consumption without violating the tasks delay constraints.

In [7], we formulated a preliminary optimization problem for the SDN-driven placement of delay-constrained computing tasks with the aim of minimizing the data exchanged in the edge domain.

**Contribution.** Despite the differences in terms of targeted optimization objectives, the scanned works share with the proposal the idea of leveraging SDN for the centralized decision of task allocation to distributed edge nodes. However, unlike our work, they do not consider the possibility of compute reuse.

As theoretically argued in [3], SDN could oversee the forwarding of requests toward edge nodes that can reuse the results of previously executed tasks. To the best of our knowledge, a preliminary work considering this aspect in a software-defined edge infrastructure is in [4]. There,

the integration between the ICN and SDN paradigms is proposed to support the compute reuse. SDN is in charge of managing routing and task allocation, with the target of ensuring the overall shortest service provisioning time, while the native in-network caching capability of ICN is used for easily caching computation results. The possible limited time validity of the tasks is however not considered.

In this paper, we abstract from the specific networking solution (IP-based or ICN-based) of edge nodes and, instead, we focus on the definition of the optimal in-network placement of time-limited reusable computing tasks in a distributed SDN-based edge infrastructure, with the target of minimizing the network resource usage, while satisfying the QoS of end-users. Our focus is to achieve an improved communication and computing resources utilization of edge nodes thanks to: (i) the reduction of the amount of data traversing the edge domain, (ii) the reuse of computation results of tasks requested from different consumers during their time validity.

### 3 SYSTEM MODEL

#### 3.1 Reference scenario and main assumptions

As a reference scenario for our study, we consider an edge network domain supervised by an SDN Controller, as illustrated in Fig. 1. The domain is composed of a set,  $N$ , of wired inter-connected edge nodes, which are SDN-enabled and directly interact with the Controller. Such nodes may encompass access points, base stations, as well as edge routers, e.g., as it is common in the backhaul network of a mobile network operator. Each node  $i$  (with  $i \in N$ ) is equipped with computing capabilities  $\mu_i$ , expressed in terms of CPU cycles per second, and storage capabilities  $S_i$ , expressed in terms of kB.

Edge nodes can act as task executors. In addition, computing tasks can be executed in the remote cloud whenever their placement at the edge is not possible, due to the limited computing resources of the edge nodes. For the sake of simplicity, the cloud is represented as a single network node  $d$  with large computing resources.

Some of the SDN nodes act as ingress nodes and provide access to the domain for a set of (i) data providers, which generate the input data for the computing tasks, and (ii) consumers, which request the computing tasks. These latter can also act as providers when requesting a computation over a self-generated content. An egress node connects the edge domain to the remote cloud through a core network segment.

Routing functions as well as computing task placement are orchestrated by the SDN Controller. In particular, the placement of computing tasks is implemented by the SDN Controller as a new *network application*, thanks to its native programmability. The proposal can benefit from the built-in SDN routines for domain-wide view of the resources. Indeed, the status of resources can be tracked by the Controller according to the approach in [4], by exploiting OpenFlow (OF) messages extension.

Although we consider an SDN-capable Controller, it is worth remarking that the conceived task placement strategy can be deployed regardless SDN, and can be applicable in general centrally-managed architectures. For instance, it can

be deployed either on top of a multi-access edge computing (MEC) orchestrator, within the ETSI MEC architecture or on a purpose-built proxy server. The latter one can then interact with a separated SDN Controller to get network-related parameters or enforce additional workarounds (native in SDN) to get routing information, so that the targeted objectives of minimizing network resource usage and meeting task delay constraints are achieved.

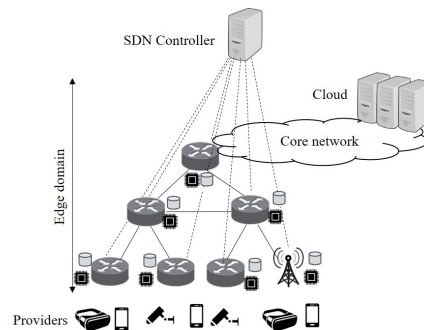


Fig. 1. Reference scenario.

A computing task,  $j$  (with  $j \in M$ , being  $M$  the set of tasks), can be described by the following tuple:  $\{s_j, D_j^{max}, l_j, T_j^{max}\}$ . Here,  $s_j$  is the size of the input data for computation;  $D_j^{max}$  is the maximum delay constraint for the computation of task  $j$ ;  $l_j$  is the amount of the computing resources, in terms of CPU cycles, required to accomplish task  $j$ ; and  $T_j^{max}$  is the time validity of the output of computing task  $j$  (i.e., at the expiration of  $T_j^{max}$  the output of the computing task  $j$  is considered meaningless and cannot serve further requests, hence it is removed from the cache). Since for a given type of computing task, the time validity may vary according to the spatio-temporal context,  $T_j^{max}$  applies to a specific instance of a computing task and it is not tied to the type of computing task. In particular, without loss of generality, in our design,  $T_j^{max}$  reflects the validity of the input data feeding the computing task, which is defined directly by data producers. For example, in a smart building scenario, the environmental parameters periodically collected by the sensors can be processed and then used by different consumers, e.g., users applications, Heating, Ventilation and Air Conditioning (HAVC) systems, energy management systems, etc. Environmental parameters, e.g., temperature, humidity, have a specific time validity typically set by the producer, e.g., some minutes. Therefore the output of a processing task taking as input those parameters will be valid for the same time frame. Then, new instances of the same data will be produced and a new computation needs to be performed. If the task takes as input data with different validity times, the lowest value will be considered as reference validity time for the task output by the executor.

For ease of reference, the key notations used in the paper are summarized in Table 1.

The following main assumptions hold in this study:

- The accomplishment of a computing task requires

data to be given as input to a computing program<sup>1</sup> which, after the execution, provides an output.

- Computing tasks are *atomic* in that they cannot be split in multiple sub-tasks, so each task is allocated to a single edge node [32].
- The input data for each computing task may need to be retrieved by *multiple providers*. For instance, a large set of data should be collected from multiple sources to train a DNN. In the case of a face detection service, several security cameras in a smart city area can send a video snapshot to the task executor [11]. In the case of an AR application, users attending the same event may provide helpful inputs to execute the computation, while also requesting the offloading.
- The time needed to transmit the result of the computation from the edge to the consumer can be neglected. Indeed, for many applications, e.g., object/face recognition, tracking, the size of the output of the computation is much smaller than the input data [33], [34], [35]. For instance, the object detection inference result over a few MB-large picture can be in the order of a few hundreds bytes [36].
- The output of a computing task is assumed to have a limited time validity, after being executed. Requests, coming after the time validity expires, entail the task to be executed again over new input data.
- Computing tasks are fully *reusable* during their time validity and therefore, once executed, they can *serve requests from different consumers*. The arrival task request rate, which determines the popularity of a task, can only affect the number of times the computation output is reused within its time validity, but it does not affect the time validity itself.

Task requests from consumers are received by the ingress nodes and forwarded to the Controller, which sets the routing path towards the edge node(s) selected for the execution, if available. An edge node can either reply with a cached output of the computing task, if still valid, or perform the computation from scratch and send the newly resulting output. How to perform a lookup operation upon the issued request depends on the networking paradigm and other specific implementation details of the edge domain. Here, without loss of generality, we consider the use of an application-level name-based representation of the computing tasks [4], [37], [38].

When a computation cannot be performed at the edge, the ingress node is instructed to forward the request to the cloud.

The centralized approach allowing a more effective task placement unavoidably incurs an additional delay to the task accomplishment needed for the task allocation request to reach the Controller. However, solutions for the most appropriate Controller placement can be devised to keep extremely low the ingress node-to-controller delay. Common values in the literature are in the order of 10 ms [39].

1. Being the focus of our study on compute reuse, we assume that the programs needed for the execution of the task are available at each edge node [31]. They can be proactively downloaded from a storage service available in the domain.

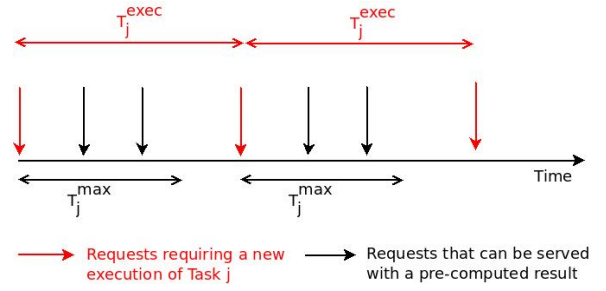


Fig. 2. Time period,  $T_j^{exec}$ , between successive executions of task  $j$  with validity time  $T_j^{max}$ .

### 3.2 Time validity of the output of the computing task

We assume that the arrival rate of requests for a computing task  $j$  in the edge domain follows a Poisson distribution with parameter  $\lambda_j$ . This is a quite common assumption in the related literature [6], [40], [41], as justified by the statistics of Google data centers, which show that the arrival intervals between tasks are exponentially distributed [42], [43]. Moreover, Poisson distribution represents a reasonable approximation when the requests are not synchronized and independent of each other, as it is the case for the considered scenario. It is worth remarking that the conceived framework still holds provided that the proper mean delays are considered for a different arrival rate (e.g., generic).

Every time an edge node is selected as executor for a computing task  $j$ , it stores the output of the computation until the time validity  $T_j^{max}$  has not expired. If, during such a time, a request for the same task is received, there is no need to execute it again (and to retrieve new input data for it). Instead, if a request from a consumer for the same task arrives after  $T_j^{max}$ , the task needs to be executed again and the new input data retrieved from the source(s). The probability that a request can be satisfied by a pre-computed task  $j$  is called *validity probability* of task  $j$  and denoted as  $p_j^v$ .

Hence, if  $\lambda_j$  is the request arrival rate for task  $j$ , this latter will be actually executed with a rate  $\lambda_j^{exec}$ , given by the following equation:

$$\lambda_j^{exec} = (1 - p_j^v) \cdot \lambda_j. \quad (1)$$

We define  $T_j^{exec} = \frac{1}{\lambda_j^{exec}}$  as the average time interval between two successive executions of task  $j$ , as graphically sketched in Fig. 2.

Under the hypothesis at hand,  $T_j^{exec}$  is larger than  $T_j^{max}$ , therefore, similarly to [44], the validity probability  $p_j^v$  can be expressed as the ratio between the time validity of the output of task  $j$  over  $T_j^{exec}$ :

$$p_j^v = \frac{T_j^{max}}{T_j^{exec}} = T_j^{max} \cdot \lambda_j^{exec}. \quad (2)$$

By combining Eq. (1) and Eq. (2), the probability  $p_j^v$  can be derived as follows:

$$p_j^v = \frac{T_j^{max} \cdot \lambda_j}{1 + T_j^{max} \cdot \lambda_j}. \quad (3)$$

### 3.3 Computation delay

Typically, in related literature [30], [33], the computation delay is equal to the time needed for executing the task and it is derived as the ratio of the computing resources necessary to a given task and the computing capability of the candidate node. However, in the most general case, the computation delay of task  $j$  at edge node  $i$  includes two contributions, i.e., the *task execution delay* at node  $i$  and the *task queuing delay*, which is the time experienced at the same node before the task can be executed, in the case other tasks are already running and nodes resources are fully utilized. Tasks are executed at each edge node one after another [45], [46].

In this work, we consider both latency contributions.

As discussed in the previous section, each task  $j$  will be actually executed with an average rate  $\lambda_j^{exec}$ . If we denote  $X_{ij}$  as the binary variable:

$$X_{ij} = \begin{cases} 1, & \text{if the task } j \text{ is executed by the node } i, \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

the overall execution rate at a node  $i$ , which is exponentially distributed, can be defined as:

$$\sum_{j \in M} X_{ij} \lambda_j^{exec}, \quad \forall i \in N \cup d. \quad (5)$$

We assume that the amount of the computing resources per task,  $l_j$ , is exponentially distributed with average value  $\bar{l}$ . Therefore, each edge node can form an M/M/1 queuing model [47] to process its corresponding computing tasks [6]. The corresponding service time follows an exponential distribution, with parameter  $\frac{\bar{l}}{\mu_i}$ .

Hence, we derive the average computation delay for a generic task at node  $i$  as the average system delay of the queue (i.e., including the queuing time and the service time corresponding respectively, to the task queuing delay and the task execution delay):

$$\bar{D}_i = \frac{1}{\frac{\mu_i}{\bar{l}} - \sum_{j \in M} X_{ij} \lambda_j^{exec}}, \quad \forall i \in N \cup d. \quad (6)$$

To keep the queue stable, the average arrival rate at node  $i$ ,  $\sum_{j \in M} X_{ij} \lambda_j^{exec}$ , should be smaller than the average service rate (i.e.,  $\frac{\mu_i}{\bar{l}} - \sum_{j \in M} X_{ij} \lambda_j^{exec} > 0$ ).

## 4 OPTIMIZATION PROBLEM

Given the system model, the main goals of this paper are (i) to minimize the network resource usage for data exchange, by also taking advantage of the reuse of computation results to additionally improve the utilization of the in-network computing resources of edge nodes, and also (ii) to meet the task delay requirements, in order to satisfy the QoS of the end-users. The first objective, which well reflects the operator's perspective, is included in the devised objective function. The possibility of compute reuse, however, is limited by the time validity of the output of the computing task, as discussed in Section 3.2. In particular, the objective function accounts for heterogeneous tasks in terms of time validity of the output, through the parameter  $T_j^{max}$ , which affects the validity probability  $p_j^v$ .

TABLE 1  
Summary of the main notations.

Symbol	Description
$N$	set of edge nodes
$d$	cloud node
$\tilde{N}$	set of candidate executors $N \cup d$
$i$	generic edge node
$\mu_i$	CPU capabilities of node $i$ (CPU cycles/s)
$S_i$	caching capabilities of node $i$
$M$	set of computing tasks
$j$	generic computing task
$D_j^{max}$	maximum delay for task $j$ to be computed
$T_j^{max}$	time validity of the output of task $j$
$K_j$	number of input contents for computing task $j$
$s_j$	size of the overall input data for computing task $j$
$s_{k_j}$	size of each input content $c_{k_j}$ required for computing task $j$
$l_j$	CPU requirement for task $j$
$X_{ij}$	binary decision variable: 1 if task $j$ is executed by node $i$ , 0 otherwise
$\lambda_j$	arrival rate of requests for task $j$
$\lambda_j^{exec}$	actual execution rate of requests for task $j$
$T_j^{exec}$	average time interval between two successive executions of task $j$
$\Omega_{i,k_j}^{min}$	number of hops separating the potential executor $i$ and the closest providers for each input content $c_{k_j}$ required for task $j$
$p_j^v$	probability that the output of a computing task $j$ is still available and valid in the edge domain

Furthermore, the network operator is concerned in guaranteeing a delay to the user that is below a given target that the user can tolerate ( $D_j^{max}$ ). This is tracked through a specific constraint in the formulated optimization problem, as clarified in the following.

Intuitively, to minimize the network resource usage and save network bandwidth, the most appropriate place for executing an offloaded task is the edge node in the closest proximity to the source(s) of the input data. Notwithstanding, the computing resources of edge nodes are limited: by offloading all computing tasks to the closest edge node(s), the performance may degrade if the processing workload is too heavy, both due to the waiting time before serving the computation request and to the service execution time itself. Furthermore, inefficiencies can be experienced if such a placement decision is taken while being oblivious of the chance of reusing computing tasks. Hence, the formulated optimization problem specifically accounts for the time validity of the output of a computing task.

As per our assumption, the input data for each computing task may need to be retrieved by multiple content sources. Hence, given a computing task  $j$ , we assume that the overall input data (of size  $s_j$ ) is the collection of distinct input contents of size  $s_{k_j}$  that are streamed to the executor from a set of different  $K_j$  sources. Therefore, it results  $s_j = \sum_{k_j=1}^{K_j} s_{k_j}$ .

Multiple paths may exist between a given content source and a candidate executor. Without loss of generality, in our study, we assume that the *shortest path*, i.e., the one incurring the *minimum number of hops*, is selected by the Controller as the route over which input data are transferred to the executor. The shortest path is considered for each input content for a given task.

We denote by  $\Omega_{i,k_j}^{min}$  the minimum number of hops be-

tween the candidate executor,  $i$ , and each content source  $k_j$  to be traversed by the input data required for computing task  $j$ .

We express the network resources usage relevant to the execution of task  $j$  as the sum of the product of each input<sup>2</sup> content size ( $s_{k_j}$ ) and the number of hops between the source of the input content and the candidate executor (either an edge node or the cloud, hence belonging to the set  $N \cup d$ , henceforth denoted as  $\tilde{N}$ ), timed by the execution rate  $\lambda_j^{exec}$ . The latter parameter, as expressed in Eq. 1, depends on the validity of probability of task  $j$ , and therefore, indirectly captures the limited time validity of a cached output.

Hence, stated in mathematical terms, recalling that we denoted as  $X_{ij}$  the optimization variable, which is equal to 1 if the task  $j$  is executed by node  $i$  and 0 otherwise, the optimization problem can be formulated as follows:

$$\min \sum_{i \in \tilde{N}} \sum_{j \in M} X_{ij} \lambda_j^{exec} \sum_{k_j=1}^{K_j} s_{k_j} \Omega_{i,k_j}^{min} \quad (7)$$

s.t.

$$\sum_{i \in \tilde{N}} X_{ij} = 1, \quad \forall j \in M; \quad (8)$$

$$X_{ij} \overline{D}_i \leq X_{ij} D_j^{max}, \quad \forall i \in \tilde{N}; \quad \forall j \in M; \quad (9)$$

$$X_{ij} \in \{0, 1\}, \quad \forall i \in \tilde{N}, \quad \forall j \in M. \quad (10)$$

To better evidence all the factors influencing the problem at hand, we can rewrite Eq. (7) by expressing  $\lambda_{exec}$  through Eq. (1), thus obtaining:

$$\min \sum_{i \in \tilde{N}} \sum_{j \in M} X_{ij} (1 - p_j^v) \cdot \lambda_j \sum_{k_j=1}^{K_j} s_{k_j} \Omega_{i,k_j}^{min}. \quad (11)$$

Constraint (8) ensures that tasks are treated as atomic. In other words, each task is assigned to only one node, executed by it in whole and not further split in sub-tasks. Constraint (9) imposes that every node must process each task  $j$  within its maximum tolerable computation delay  $D_j^{max}$ . Whenever the delay constraint cannot be ensured by edge nodes, because of their limited resources, computing tasks will be offloaded to the cloud, which can leverage higher, virtually unlimited computation capabilities and, hence, always satisfy constraint (9). We note that this constraint, due to Eq. (6), is described by nonlinear inequalities. Finally, the constraint in Eq. (10) reminds that we conveniently model the computing task placement problem with a binary integer variable.

The formulated model can be transformed into a 0-1 ILP problem by considering the following argument: for each pair  $(i, j)$ , if  $X_{ij} = 0$  then the inequality in Eq. (9) is automatically verified, whilst if  $X_{ij} = 1$  it is verified for  $\overline{D}_i \leq D_j^{max}$ . This allows to rewrite Eq. (9) as the following linear inequality:

$$D_j^{max} / \overline{D}_i \geq X_{ij}, \quad (12)$$

2. The model could be extended to account for the exchanged output data, from the executor to the requesting node, if its size is not negligible compared to that of the input data.

that is Eq. (9) can be substituted by:

$$X_{ij} \leq D_j^{max} \left( \frac{\mu_i}{\overline{l}} - \sum_{j \in M} X_{ij} \lambda_j^{exec} \right), \quad \forall i \in \tilde{N}; \quad \forall j \in M. \quad (13)$$

## 5 THE CONCEIVED HEURISTICS

**Proposition 1.** *The formulated task placement problem is NP-hard.*

*Proof:* Our problem formulated in Eq. (7) is equivalent to the Generalized Assignment Problem (GAP) [48], which is a well known NP-hard problem in the combinatorial optimization literature. It consists in finding the best assignment of items to bins (i.e., computing tasks to edge nodes in our formulation) while minimizing the overall cost. Bins have different capacities and each item has a different size and a different cost according to the bin it is assigned to. The solution should respect the following constraints: (i) each item should be assigned to one bin (i.e., a task can be executed by one node only, Eq. (8)), (ii) bins' capacities as well as items' constraints should not be, respectively exceeded and neglected (i.e., the CPU load of a node should not be greater than the threshold required to respect the task constraint on computing delay, Eq. (13)).  $\square$

### 5.1 Basics

Although the ILP formulated in Eqs. (7)-(10) can obtain the optimal solution for the task placement problem, it suffers from high computational complexity as problem instances increase.

In order to improve time efficiency in finding the task placement solution, we propose to leverage a heuristic algorithm. The latter one is designed in a centralized way and executed at the Controller, thus reflecting the SDN approach our proposal builds upon. Given a set of computing tasks to be executed, denoted as  $M_r$  (with  $M_r \subseteq M$ ), the target of the heuristic algorithm is to allocate each of them in the nodes which ensure the minimum cost (see Eq. (16)) expressed as the amount of exchanged data to carry on the processing operations, in alignment with the objective function in Eq. (7), while meeting the delay constraint in Eq. (13). We can quantify the amount of data exchanged to execute task  $j$  on node  $i$  as:

$$Data(i, j) = \lambda_j (1 - p_j^v) \sum_{k_j=1}^{K_j} s_{k_j} \Omega_{i,k_j}^{min}. \quad (14)$$

We can further compute the number of operations (in terms of CPU cycles) required for the execution of a task  $j$ ,  $WL(j)$ , as follows:

$$WL(j) = (1 - p_j^v) \cdot l_j. \quad (15)$$

Finally, we define the cost of executing the task  $j$  on node  $i$ ,  $c(i, j)$ , as the amount of data exchanged for each unit of CPU of node  $i$  assigned to task  $j$  (normalization is considered to account for heterogeneous task demands and is perfectly aligned with the common practice adopted to design heuristics for allocation problems [49]):

$$c(i, j) = \frac{Data(i, j)}{WL(j)} = \frac{\lambda_j \sum_{k_j=1}^{K_j} s_{k_j} \Omega_{i,k_j}^{min}}{l_j}. \quad (16)$$

## 5.2 Heuristic algorithm

To approximate the optimal task placement solution, the proposed heuristic algorithm follows a greedy approach that is explained below and summarized in Algorithm 1.

The algorithm proceeds iteratively. It starts by considering the sets of requested tasks and of potential executors (sorted in ascending order w.r.t. their processing capabilities in line 1) and calculates the cost of assigning each task  $j \in M_r$  to each executor  $i \in \tilde{N}$  according to Eq. (16). For each  $j \in M_r$ , the cost and executor pairs are included in ascending order into a data structure called *list of best executors for j*,  $BE_j$  (lines 2-8). Basically, the first element of the list identifies the best executor and the corresponding cost, the second element identifies the second best executor and the corresponding cost, etc.

At each iteration, the algorithm selects a node  $i$  and tries to allocate there the tasks that have identified  $i$  as best executor (lines 11-32). In particular, the algorithm creates a list  $T_i$  (line 14) that includes the potential tasks to be allocated in node  $i$  and orders the items in descending order based on the cost  $c(i, j)$  (line 17). More costly tasks (as per Eq. (16)) are those that foresee the higher amount of exchanged data, because of the higher input content size and number of hops they should traverse from the source to the candidate executor and the more frequent requests. In particular, the basic idea we pursued is to prioritize serving the tasks having the higher cost in order to save network resources, as placing them into another node would result in a further increase in the overall edge resource usage. Then, the algorithm checks if the constraint reported in Eq. (13) is verified (line 21) and, based on this condition, decides if the selected tasks in  $T_i$  can be actually allocated in node  $i$ . In case of a successful outcome, the tasks are purged from the set  $M_r$ ; otherwise other placement options must be considered according to the information in  $BE_j$ . The loop continues until all the tasks are allocated at the edge nodes or in the cloud.

## 5.3 Heuristics complexity and approximation bound

If we consider an efficient sorting algorithm like QuickSort, the complexity of our Algorithm scales linearly with the number of nodes and is bounded by  $O(|M_r| \cdot |\tilde{N}| \cdot \log |M_r|)$ .

In addition, an approximation bound can be proven for the devised heuristic algorithm. The formulated problem is equivalent to the maximization of a submodular function under a knapsack constraint. We can demonstrate that the conceived heuristic, according to [50], can achieve (close to)  $(1 - \frac{1}{e})$ -approximation, which is the best result due to Svridenko [51]. Before demonstrating that the approximation holds, similarly to [52], we first carry out a transformation by introducing a new reward function (to be maximized) for each task, i.e.,  $f_r(i, j) = c(d, j) - c(i, j)$ , where  $c(d, j)$  is a constant which denotes the expected cost of a given task when it is allocated in the cloud (denoted as  $d$ ).

We can demonstrate that the objective function,  $f_r(i, j)$  is a monotone non decreasing submodular function. Monotonicity is trivial because any new placement of a task cannot decrease the value of the objective function. Indeed, the following cases can occur:

- The new task has been already executed, the relevant output is still valid, hence there will be not additional gain.
- The new task is executed by an edge node; hence, the additional gain is non negative (the task incurs a data transfer cost if it needs to be executed since the input needs to reach the executor and this cost is lower than  $c(d, j)$ ).
- The new task is executed in the cloud; hence, the additional gain is zero.

For what concerns submodularity, since the sum of submodular functions is a submodular function, it suffices to prove that the objective function  $f_r(i, j)$  is submodular  $\forall j \in M$ . Submodular functions capture the concept of diminishing returns: as the set becomes larger the benefit of adding a new element to the set will decrease [53]. Intuitively, this applies to our problem, since as the set of tasks becomes larger, new tasks are more likely executed to further edge nodes (the closer ones get saturated) and ultimately, to the cloud, for which the gain will not increase at all, since input data need to traverse more hops (the corresponding cost  $c(i, j)$  increases).

## 5.4 Implementation aspects

In a practical design, the heuristic algorithm is executed by the Controller that leverages its global view of the edge domain. As in traditional SDN deployments [54], the Controller maintains a Network Information Base (NIB), which records all the information about the edge network topology together with the nodes' capabilities. Therefore, the Controller may estimate the number of hops separating a generic edge node  $i$  from another end-point.

We assume the ingress nodes receive the task request messages from the different consumers, send them to the Controller and wait for the configuration of the flow tables. A request message for a task  $j \in M_r$  includes the attributes  $s_j, D_j^{max}, l_j, T_j^{max}$ . By accessing this information and the NIB content, the Controller calculates, per each  $j \in M_r$ , the correspondent cost of executing it in each edge node  $i \in \tilde{N}$ , according to Eq. 16, and executes Algorithm 1. Based on the obtained output,  $X_{i,j}$ , the Controller fills the flow tables to set the routing path from the ingress nodes to the executors and, in parallel, from the executors to the input data providers. After receiving the task request, the executors retrieve the input data from the providers, perform the computation, and finally return the output back to the consumers.

# 6 PERFORMANCE EVALUATION

## 6.1 Main settings

The conducted evaluation study aims to assess the performance of the proposed optimal task placement strategy. In a first campaign, we validate the heuristic algorithm, comparing it against the results of the solved ILP problem. The latter one has been computed by using the standard solver provided through the integration of Matlab® and IBM CPLEX Optimization Studio tool. Heuristic solutions are instead derived by using only Matlab®. An additional

---

**Algorithm 1:** The proposed heuristic algorithm
 

---

```

input : Set of candidate executors  $\tilde{N}$ ; set of
         requested computing task  $j \in M_r$ ;
         processing capabilities  $\mu_i \forall i \in \tilde{N}$ ;
output:  $X_{ij} \forall i \in \tilde{N}, \forall j \in M_r$ 
1 sort( $\tilde{N}$ , according to  $\mu_i$ , increasing);
2 for  $j \in M_r$  do
3   for  $i \in \tilde{N}$  do
4     calculate  $c(i, j)$ ;
5     insert  $[i, c(i, j)]$  in  $BE_j$ ;
6   end
7   sort( $BE_j$ , according to  $c(i, j)$ , increasing);
8 end
9 while  $M_r \neq \emptyset$  do
10   $i \leftarrow 1$ ;
11  while  $i \leq |\tilde{N}|$  do
12    for  $j \in M_r$  do
13      if the first element of  $BE_j$  contains  $i$  then
14        insert  $[j, c(i, j)]$  in  $T_i$ 
15      end
16    end
17    sort( $T_i$ , according to  $c(i, j)$ , decreasing);
18    while  $T_i \neq \emptyset$  do
19       $j \leftarrow \text{read}(T_i, \text{first element})$ 
20      update  $\bar{D}_i$  to include  $j$ 's load;
21      if  $\bar{D}_i < D_j^{\max}$  then
22         $X_{ij} \leftarrow 1$ ;
23        purge  $j$  from  $M_r$ ;
24      end
25      else
26        purge  $i$  from  $BE_j$ ;
27        restore  $\bar{D}_i$  by excluding  $j$ 's load;
28      end
29      purge  $j$  from  $T_i$ ;
30    end
31     $i++$ ;
32  end
33 end
34 return  $X_{ij} \forall i \in \tilde{N}, \forall j \in M_r$ 

```

---

simulation campaign has been conducted to assess the performance of the proposal when compared against benchmark placement solutions.

All simulation campaigns have been run on an Intel Core i7-6700HQ, 2.6 GHz (CPU), 16 GB (RAM), 512 GB (SSD), 1TB (HDD).

**Network topology.** Similarly to [4], [55], we reproduce in Matlab® a medium size network topology with 29 edge nodes that resembles a metropolitan area network (MAN). Four upper-layer nodes, interconnected in a full meshed topology, form a backbone ring and are the roots of three-layered fat tree topologies. The leaf nodes at the bottom of each tree topology act as ingress nodes which providers and consumers are connected to. A border router acts as the egress node, connecting the backbone ring to the cloud. Nodes exhibit different computing capabilities (Table 2), in particular, we assume that the closer the edge node is to the

provider, the lower is its computational capacity [55]. Note that the formulated model and proposed heuristic algorithm are general, and can be applied in other network scenarios with different parameters settings.

**Task settings.** We consider a catalog of 1000 computing tasks differing in terms of computation load, maximum computation delay, time validity of their output. All tasks' features are summarized in Table 2. Similarly to [56], we model the popularity of each task  $j$  according to the Zipf's law, unless differently stated in the text.

**Metrics.** The following metrics are derived:

- *Exchanged data*: it measures the network resource usage in the domain as the overall amount of input data (in bytes) transmitted by providers to the selected executors multiplied by the number of hops traversed by such data in the network<sup>3</sup>.
- *Task offloading to the cloud*: it measures the percentage of requested computing tasks that are offloaded to the cloud.
- *Edge computation hit*: it measures the fraction of the outputs of the task computed by the edge nodes which are reused to serve different requests over the total number of requested tasks.
- *Average queuing delay*: it measures the average queuing delay experienced by tasks before their execution at the selected edge node.
- *Average execution delay*: it measures the average of the actual time needed for the execution of the task.
- *Average computation delay*: it measures the average computation delay, including both the queuing delay as well as the execution delay.

The first three metrics reflect the effectiveness of the proposal in minimizing the amount of exchanged data as well as in improving the utilization of edge resources. Measured values for the above metrics are averaged over 200 runs and reported with 95% confidence intervals.

## 6.2 Heuristics validation

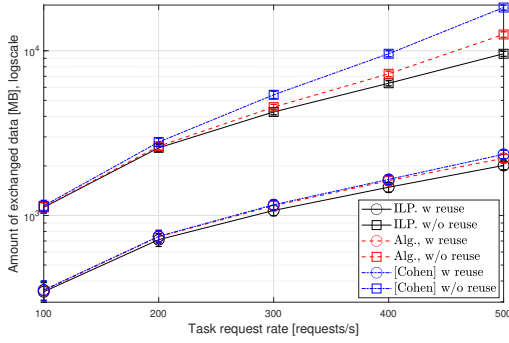
In this section we evaluate the performance of the formulated optimal placement strategy (curves labeled as *ILP* in Figures) against our heuristic algorithm (curves labeled as *Alg.* in Figures), when varying the rate of computing task requests from 100 to 500.

We consider a further heuristics (curves labeled as [*Cohen*] in Figures) built upon the approximation algorithm in [59], which describes a family of heuristics for the GAP, obtained by applying the local-ratio technique to any algorithm for the single 0-1 knapsack problem. The optimality bound for it has been theoretically proven in [59]. In particular, in [59] it is demonstrated how any polynomial time  $\alpha$ -approximation algorithm for the knapsack problem can be translated into a polynomial time  $(1 + \alpha)$ -approximation algorithm for GAP. The approximation algorithm in [59] is among the most well-established ones for solving the GAP, also in the edge computing domain, as witnessed by

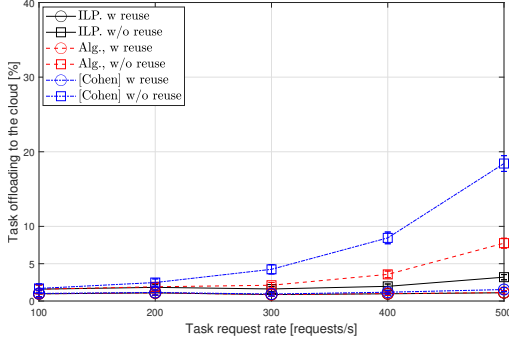
3. In case of task offloading to the remote cloud, the number of hops is set to 14, in agreement with other literature works [57], [58]. The range of the number of hops between content sources and executors belonging to the edge domain varies in [1,5].

TABLE 2  
Main simulation settings.

Parameter	Setting
Processing capability ( $\mu_i$ )	<ul style="list-style-type: none"> <li>Leaf nodes: 250 CPU cycles/s</li> <li>Intermediate nodes: 500 CPU cycles/s</li> <li>Upper-layer nodes: 1000 CPU cycles/s</li> <li>Cloud: 20000 CPU cycles/s</li> </ul>
Task catalog size	1000
Number of hops	<ul style="list-style-type: none"> <li>towards an executor in the edge [1,5]</li> <li>towards the cloud [14]</li> </ul>
Maximum computation delay ( $D_j^{max}$ )	Uniformly distributed in [10, 100] ms
Time validity of the output of a task ( $T_j^{max}$ )	Exponentially distributed with average $T_j^{max} = 300, 500, 1000$ ms
Average task computing workload ( $\bar{l}$ )	10 CPU cycles
Input data size ( $s_j$ )	10 MB
Arrival rate of requests for task ( $\lambda_j$ )	Poisson distributed with average 10 requests/s
Task requests distribution	<ul style="list-style-type: none"> <li>Uniform</li> <li>Zipf (<math>\alpha=0.8, 1.2</math>)</li> </ul>



(a) Exchanged data.

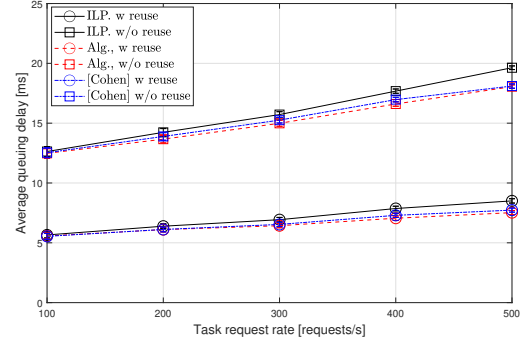


(b) Task offloading to the cloud

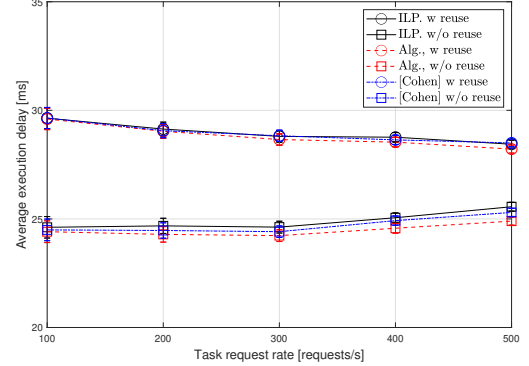
Fig. 3. Heuristics validation: effectiveness metrics ( $\alpha=0.8, \bar{T}_j^{max} = 300$  ms).

very recent literature [60], [61]. In our case, the heuristic algorithm receives as input the cost function (Eq. (16)), the node capabilities and the task constraints and, after reducing the GAP by iteratively solving the 0-1 knapsack problem for each bin, through the standard greedy algorithm [59], returns the set of tasks assigned to each node.

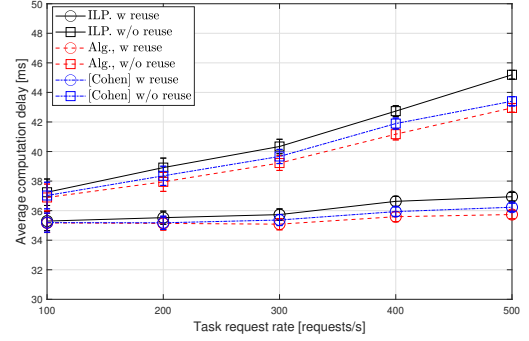
Results are reported both for the case when reusability of the output of the computing tasks is considered (curves labeled as *w reuse*) and when not (curves labeled as *w/o reuse*). It can be observed that both the considered heuristics



(a) Average queuing delay.



(b) Average execution delay.



(c) Average computation delay.

Fig. 4. Heuristics validation: delay metrics ( $\alpha=0.8, \bar{T}_j^{max} = 300$  ms).

tics well approximate the optimal solution. In particular, our heuristic algorithm is slightly closer to the ILP results compared to the benchmark.

For the sake of validation, metrics in Figs. 3 and 4 are considered when fixing the skewness parameter  $\alpha$  of the Zipf distribution equal to 0.8. Fig. 3(a) shows the amount of exchanged data within the edge domain. Reasonably, for both compared solutions, the metric increases as the rate of requests increases, due to the higher amount of input data that need to be transferred. Furthermore, the amount of exchanged data traffic is lower by almost one order of magnitude in case of reuse. Not surprisingly, the ILP provides the lowest values for the considered metric of exchanged data. By exploring different solutions, the ILP is more effective in finding the optimal task placement solution which ensures to minimize the objective function.

Once loaded, not all edge nodes are able to process the requested task ensuring the computation delay constraint in Eq. (13), hence some of the requested tasks are offloaded

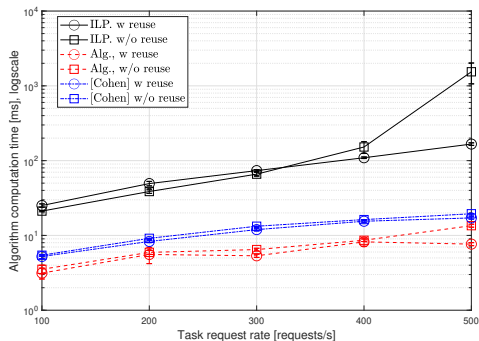


Fig. 5. Heuristics Vs. ILP: algorithm computation delay.

to the cloud (Fig. 3(b)). Not surprisingly, this occurs with a high probability when reuse of the task is not possible, as the task request rate increases. Moreover, such a percentage is lower for our heuristics confirming its superiority compared to the benchmark one. Our algorithm in fact, fills each node by greedily taking only from the list of tasks (line 16) that have selected that node as their best executor. Contrarily, in the considered benchmark, according to [59], each node is filled by greedily taking from the entire set of tasks to be allocated, no matter if the considered node represents the best executor for them. There, non-optimal assignments may occur at some iterations. They are subsequently fixed by moving a task initially assigned to a node to another one which provides a lower cost, as inferred through the marginal cost computation. This workaround may result in wasting some resources in the node where the task is unallocated.

The reduction of the amount of exchanged data traffic in case of reuse is paid in terms of higher execution delays, as reported in Fig. 4(b). This is because for some tasks in case of reuse, less capable edge nodes (instead of the remote cloud) act more as executors. Contrarily, thanks to the fact that there is no need to execute some tasks, which can be reused for a while, the queuing delays (Fig. 4(a)) are lower. Consequently, computation delay values (Fig. 4(c)) are lower in the case of reuse compared to the case in which all requests are served with a new task execution.

As a further result, in Fig. 5 we report the time required to run the heuristic algorithms as well as to solve the ILP through the standard solver. In both cases, with and without the reuse of the output, the computation times of the two heuristics are significantly lower than those measured for the ILP solution, reasonably confirming their greater efficiency. They are both well below 20 ms, with our heuristic algorithm which is slightly faster. Hence, in the following, we will report results only for it. Moreover, the proposed heuristics shows even shorter computation times when reuse is enabled: fewer alternatives need to be explored compared to the case in which reuse is not enabled and the task placement solution can be found more quickly. Hence, in the following, we will refer only to it.

### 6.3 Our proposal Vs. benchmark placement solutions

Results of the proposed placement strategy are compared against two benchmark solutions. The first one is representative of a legacy MEC approach (it is labeled as *MEC w*

*cloud* in the plots) according to which tasks are executed *only in purpose-built servers*, which are exclusively attached to the access nodes of the edge domain, and offloaded to the cloud, whenever delay constraints cannot be met due to the depletion of their resources. The second one refers to the case in which *all tasks are executed in the remote cloud* (it is labeled as *onlyCloud* in the plots).

#### 6.3.1 Impact of tasks popularity

The first simulation campaign is conducted to evaluate the impact of the tasks' popularity on the achieved performance. We compare results achieved in presence of a uniform task request distribution (curves labeled as *Uniform* in Figures) against the ones achieved for the Zipf distribution, for skewness parameter  $\alpha$  equal to 0.8 and 1.2 (curves labeled as *Zipf* in Figures). The campaign is conducted when varying the rate of requested tasks from 100 to 2000 requests/s.

The higher the popularity of tasks (i.e., the higher  $\alpha$ ) the better the performances: a lower amount of data is exchanged in the domain (Fig. 6(a)), a lower percentage of tasks is offloaded to the cloud (Fig. 6(b)), a higher computation hit is experienced (Fig. 6(c)), a lower computation delay is achieved (Fig. 7). This is because requests concentrate on a few tasks and once these are executed, their output can be reused to serve different consumers, with no need to exchange traffic to retrieve the input data and with a lower processing burden on edge nodes.

Results further show that a remarkable reduction of the traffic crossing the edge domain (higher as the popularity increases) is experienced by our proposal compared to the *MEC w cloud* solution. The reduction is up to around 91%, measured for  $\alpha = 1.2$  and task request rate equal to 2000 requests/s. Limiting the task execution to purpose-built edge servers is significantly less effective than our proposal, which distributes tasks across the whole edge domain. Edge servers attached to the leaf nodes are forced to offload to the cloud at least around half of the tasks, regardless of the considered popularity (Fig. 6(b)). The tasks executed locally experience a long queuing delay, which contributes to a higher overall computation delay (Fig. 7)<sup>4</sup>.

Gains in terms of reduced amount of exchanged traffic compared to the baseline *onlyCloud* solution are in the order of 93% measured for  $\alpha=1.2$  and task request rate equal to 2000 requests/s.

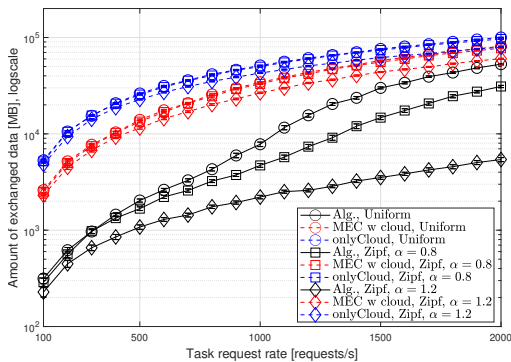
In particular, Fig. 6(c) shows that, on average, more than half of the outputs (up to 85% in the case of  $\alpha=1.2$  and 2000 requests/s) is re-used to address several requests.

To further shed light into the performance of the proposed task allocation strategy, we measure the average computation delay metric as a function of the normalized edge resource usage (NRU), which has been defined as:

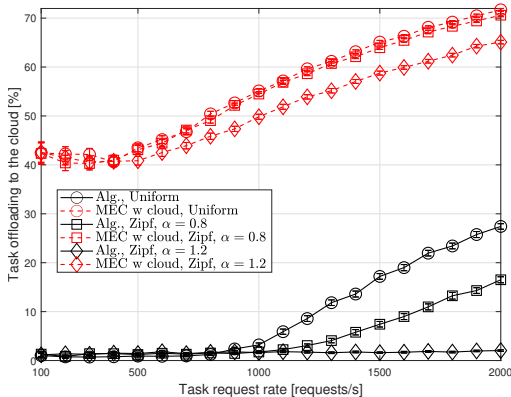
$$NRU = \frac{\sum_{j \in M} \lambda_j^{exec} \cdot l_j}{\sum_{i \in N} \mu_i} \quad (17)$$

More precisely, in Eq. (17) the numerator represents the effective CPU cycles per second required to accomplish the requested tasks, while the denominator represents the

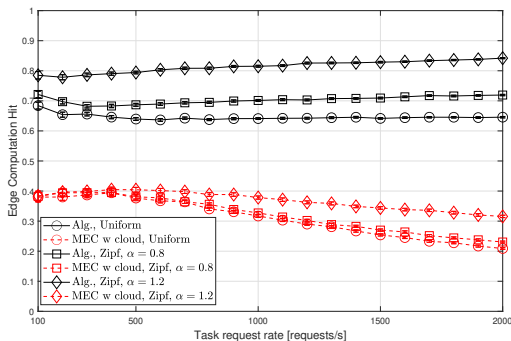
4. Results are not reported for the *cloudOnly* solution, since the delay is negligible thanks to the virtually unlimited capabilities of the cloud.



(a) Exchanged data.



(b) Task offloading to the cloud



(c) Edge computation hit

Fig. 6. Effectiveness metrics when varying tasks popularity ( $\bar{T}_j^{max} = 500 \text{ ms}$ ).

overall amount of CPU cycles per second provided by edge nodes. Hence, NRU gives a measure of how the offered computing load compares with the overall computing resources available in the edge domain. It is worth to be noted that resources made available by the cloud have been intentionally excluded from the denominator of Eq. (17).

The simulations have been conducted by considering a number of requested tasks variable in the range [100, 2000], when fixing  $\bar{T}_j^{max} = 500 \text{ ms}$ , while the rest of the parameters have been set according to Table 2. For each simulation, the average computation delay Vs. the normalized resource usage has been computed and reported as a scatterplot in Fig. 8(a). It can be observed that the higher the popularity (the greater  $\alpha$ ), the lower the edge resource usage, thanks to the higher reuse.

Moreover, it can be observed that the scattered points for

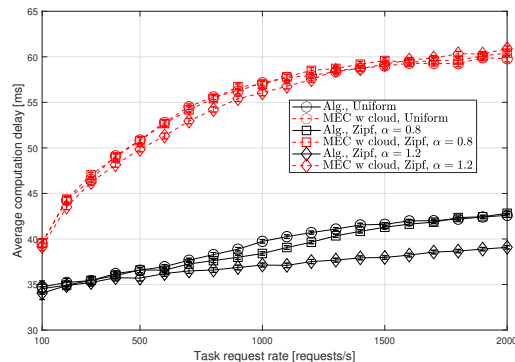


Fig. 7. Average computation delay when varying tasks popularity ( $\bar{T}_j^{max} = 500 \text{ ms}$ ).

the MEC w cloud benchmark have a steeper slope compared to the proposed algorithm as the resource usage increases. This is because the available computing resources at the bottom of the edge topology quickly deplete.

As a further metric, to better capture a network operator-centric perspective, we derive the extra-domain rate (EDR) when varying the normalized edge resource usage, Fig. 8(b). The metric accounts for the bandwidth needed to transfer out of the edge domain the input data feeding the computing tasks to be executed in the cloud. It is derived as follows:

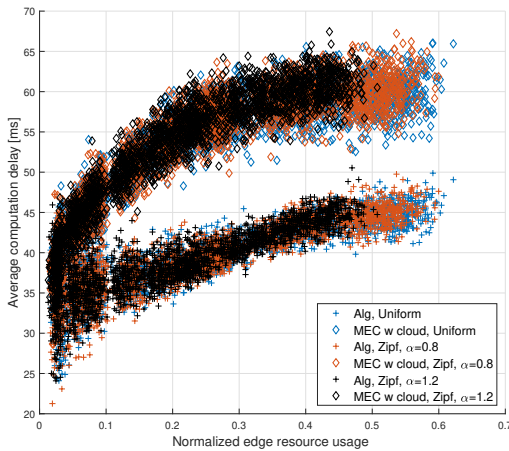
$$EDR = \sum_{j \in M} X_{dj} \lambda_j^{exec} \cdot S_{dj}. \quad (18)$$

The metric linearly increases as the normalized edge resource usage increases for the *cloudOnly* solution and in a similar manner, although less steeply, for the MEC w cloud benchmark. It can be observed that the values for this metric are more than halved when compared to the case in which all tasks are executed in the cloud, by saving precious bandwidth resources.

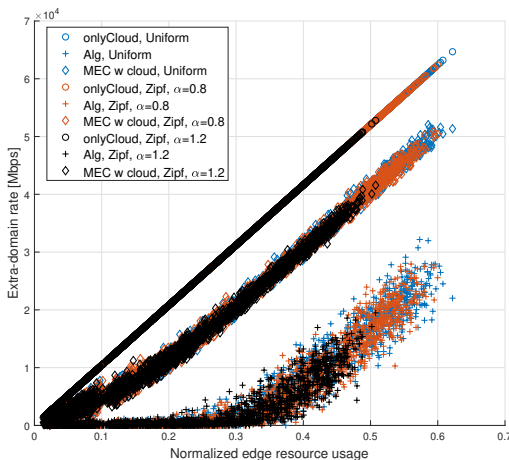
### 6.3.2 Impact of the output time validity

The second simulation campaign is conducted when varying the mean time validity of the reusable output of computing tasks ( $\bar{T}_j^{max} = 0.5 \text{ s}$  and  $1 \text{ s}$ ) and for the case in which the compute reuse is not enabled.

The higher the time validity the better the measured performances, see Figs. 9 and 10, whatever the considered solution, since the reuse is exploited to a greater extent. Also under these settings, the traffic exchanged in the domain by the proposal is significantly reduced compared to the benchmark solutions (Fig. 9(a)). The gains achieved with a higher time validity get more remarkable as the rate of task requests increases. In particular, it is reduced up to 71.2% and 82.9% compared to the MEC w cloud and onlyCloud solutions, respectively, for  $\bar{T}_j^{max}$  equal to  $1 \text{ s}$  and task request rate equal to 2000 requests/s. Under such settings, if reuse is not possible, instead, the edge domain is not able to handle the load and a high number of tasks is offloaded to the cloud: more than half of them are offloaded above 1500 requests/s (Fig. 9(b)). The MEC w cloud benchmark scheme offloads more than 60% of task requests already for an offered load of 100 requests/s. The higher load in absence of reuse is also confirmed by the higher computation delay,



(a) Average computation delay



(b) Extra-domain rate

Fig. 8. Metrics vs normalized edge resource usage ( $\overline{T}_j^{max} = 500$  ms).

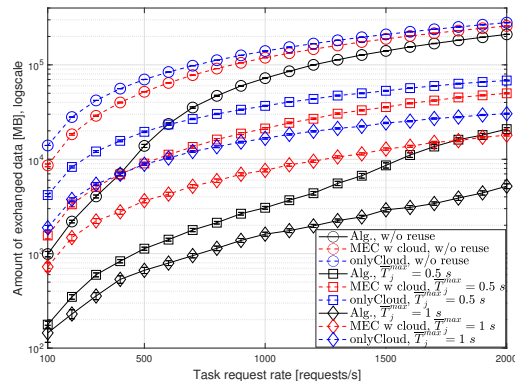
Fig. 10, mainly due to the higher queuing delay contribution.

## 7 CONCLUSIONS AND FUTURE WORKS

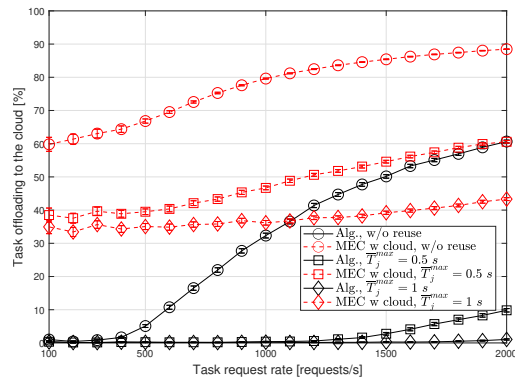
In this paper we have proposed a novel strategy for the placement of computing tasks with time-limited reusable outputs over an SDN-based network edge domain. The optimal solution is formulated through an ILP problem, which targets the network resource usage minimization in the selection of the candidate executors. The latter ones include both SDN edge nodes and the remote cloud as the ultimate choice, not to saturate edge nodes and bound the task computation latency.

A novel heuristic algorithm is proposed which is shown to implement the decision in an affordable manner. It provides a placement solution, by well approximating the optimal one provided by the ILP. This is especially true when reuse is enabled, which is the setting we are most interested in. The formulated optimization problem is also solved, with a similar efficiency in terms of computation time, through a near-optimal heuristic algorithm derived from a greedy approach in the literature.

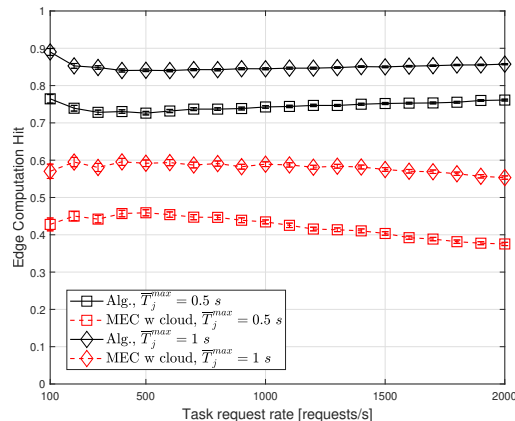
A wide simulation campaign has been conducted to assess the behaviour of our proposal, when compared with



(a) Exchanged data.



(b) Task offloading to the cloud



(c) Edge computation hit

Fig. 9. Effectiveness metrics when varying the time validity of the output ( $\alpha = 0.8$ ).

other placement strategies, through several valuable metrics under different settings in terms of task requests, task popularity distribution and time validity of the output of the computation. Results show that the proposal well meets the task constraints and properly exploits network edge resources, while offloading only a small amount of tasks to the remote cloud. Benefits can be particularly appreciated especially when popular tasks are considered (higher Zipf skewness parameter) with a higher time validity of the task output, nonetheless the increasing number of task requests. Interestingly, the amount of traffic exchanged in the edge domain by our proposal reduces up to 93% compared to the baseline *onlyCloud* solution.

Achieved findings pave the way for enabling the reuse

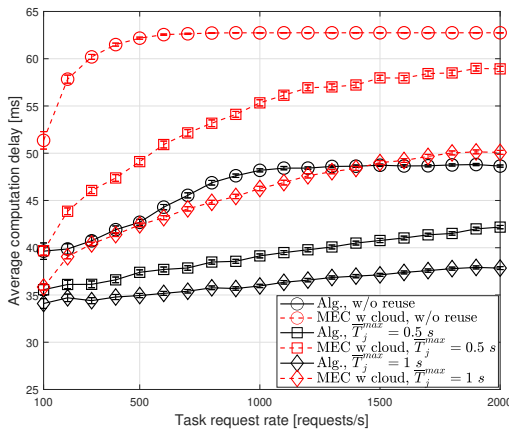


Fig. 10. Average computation delay when varying the time validity of the output ( $\alpha = 0.8$ ).

of the output of the computing tasks, in order to make the best of the network edge resources.

However, several issues still lie ahead which entail further investigations as detailed in the following.

**Objective function.** The conceived framework can be modified to account for the optimization of different objectives, like the network latency experienced in data exchange, while still benefiting from time-limited compute reuse.

**Practical deployment.** To enable compute reuse, edge nodes should be able to recognize distinct requests for the same task. This is viable either by leveraging (i) application layer solutions like Deep Packet Inspection (DPI) or Proxy-based mechanisms [37], or (ii) the information-centric Named Data Networking (NDN) architecture, which leverages names, directly at the network layer, to uniquely identify contents and also generic tasks/services [38], [62]. A higher overhead is incurred by the former solutions, compared to the latter one in which a lookup operation in the NDN tables, during the request forwarding process, is only needed, which typically introduces a negligible delay if names are properly encoded [63]. At a practical level, this would imply the definition of semantic-rich task names, which consumers, SDN Controllers and candidate task executors should agree upon.

**Time validity estimation.** In our design we have assumed the time validity of the output of the computing task to hold for a specific task request instance. It can vary from a task instance to another. However, under some circumstances, the actual parameter may be shorter than what initially set (if the conditions of the environment, e.g., the input data, suddenly change) and hence, causing the output of the computing task to be no longer valid. It can be also longer (if the conditions of the environment, e.g., the input data, do not change) making the output of the computation valid for a longer time. Artificial intelligence solutions could be deployed at the executor to properly predict the validity time of the computation output. Alternatively, the validity time could be calculated by the input data source(s), as we assumed in our design, and announced to the executor together with some additional cache control directives. For instance, information like “no-cache” and “must-revalidate” directives, currently leveraged by the Hyper Text Transfer

Protocol (HTTP) cache control mechanism, could be introduced in our design to specify that task results need to be re-validated with the input data source(s), respectively, upon each reuse and at the estimated expiration. Further studies are needed to appropriately trade-off between the incurred delay and signalling traffic introduced in the network and the accuracy of the results.

## REFERENCES

- [1] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, “A survey on computation offloading modeling for edge computing,” *Journal of Network and Computer Applications*, p. 102781, 2020.
- [2] M. McBride *et al.*, “Edge Data Discovery for COIN, IETF Computing in the Network Research Group,” November 2020.
- [3] B. Nour, S. Mastorakis, and A. Mtibaa, “Compute-less networking: Perspectives, challenges, and opportunities,” *IEEE Network*, vol. 34, no. 6, pp. 259–265, 2020.
- [4] M. Amadeo, C. Campolo, G. Ruggeri, A. Molinaro, and A. Iera, “SDN-managed provisioning of named computing services in edge infrastructures,” *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1464–1478, 2019.
- [5] A. Al-Shuwaili and O. Simeone, “Energy-efficient resource allocation for mobile edge computing-based augmented reality applications,” *IEEE Wireless Comm. Letters*, vol. 6, no. 3, pp. 398–401, 2017.
- [6] Q. Fan and N. Ansari, “Application aware workload allocation for edge computing-based IoT,” *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2146–2153, 2018.
- [7] G. Lia, M. Amadeo, C. Campolo, G. Ruggeri, and A. Molinaro, “Optimal placement of delay-constrained in-network computing tasks at the edge with minimum data exchange,” in *IEEE 4th 5G World Forum (5GWF) 2021*, pp. 481–486.
- [8] S. Shannigrahi, S. Mastorakis, and F. R. Ortega, “Next-generation networking and edge computing for mixed reality real-time interactive systems,” in *IEEE ICC Workshops 2020*, pp. 1–6.
- [9] S. Bi, L. Huang, and Y.-J. A. Zhang, “Joint optimization of service caching placement and computation offloading in mobile edge computing systems,” *IEEE Trans. on Wireless Communications*, vol. 19, no. 7, pp. 4947–4963, 2020.
- [10] J. Xu, L. Chen, and P. Zhou, “Joint service caching and task offloading for mobile edge computing in dense networks,” in *IEEE INFOCOM, 2018*, pp. 207–215.
- [11] J. Lee, A. Mtibaa, and S. Mastorakis, “A case for compute reuse in future edge systems: An empirical study,” in *IEEE GLOBECOM Workshops, 2019*, pp. 1–6.
- [12] S. Mastorakis, A. Mtibaa, J. Lee, and S. Misra, “ICedge: when Edge Computing meets Information-Centric Networking,” *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4203–4217, 2020.
- [13] M. W. Al Azad and S. Mastorakis, “Reservoir: Named data for pervasive computation reuse at the network edge,” in *2022 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2022, pp. 141–151.
- [14] B. Ottenwalder, B. Koldehofe, K. Rothermel, K. Hong, and U. Ramachandran, “Recep: Selection-based reuse for distributed complex event processing,” in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, 2014*, pp. 59–70.
- [15] A. C. Baktir, A. Ozgovde, and C. Ersoy, “How can edge computing benefit from software-defined networking: A survey, use cases, and future directions,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2359–2391, 2017.
- [16] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A survey on mobile edge computing: The communication perspective,” *IEEE Comm. Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.
- [17] X.-Q. Pham, T.-D. Nguyen, E.-N. Huh *et al.*, “Joint service caching and task offloading in multi-access edge computing: A QoE-based utility optimization approach,” *IEEE Communications Letters*, 2020.
- [18] M. Kim, H. Cho, Y. Cui, and J. Lee, “Service caching and computation resource allocation for large-scale edge computing-enabled networks,” in *IEEE GLOBECOM, 2020*, pp. 1–6.
- [19] P. Liu, G. Xu, K. Yang, K. Wang, and X. Meng, “Jointly optimized energy-minimal resource allocation in cache-enhanced mobile edge computing systems,” *IEEE Access*, vol. 7, pp. 3336–3347, 2018.
- [20] G. Lee, W. Saad, and M. Bennis, “Online optimization for low-latency computational caching in fog networks,” in *IEEE Fog World Congress (FWC) 2017*, pp. 1–6.

- [21] Y. Cui, W. He, C. Ni, C. Guo, and Z. Liu, "Energy-efficient resource allocation for cache-assisted mobile edge computing," in *IEEE Conf. on Local Computer Networks (LCN)*, 2017, pp. 640–648.
- [22] P. Guo, B. Hu, R. Li, and W. Hu, "Foggycache: Cross-device approximate computation reuse," in *Int. Conf. on Mobile Computing and Networking*, 2018, pp. 19–34.
- [23] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.
- [24] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, "Cloud gaming: architecture and performance," *IEEE Network*, vol. 27, no. 4, pp. 16–21, 2013.
- [25] T. N. B. Duong and S. Zhou, "A dynamic load sharing algorithm for massively multiplayer online games," in *The 11th IEEE International Conference on Networks, 2003. ICON2003.* IEEE, 2003, pp. 131–136.
- [26] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.
- [27] S. Goudarzi, M. H. Anisi, H. Ahmadi, and L. Musavian, "Dynamic resource allocation model for distribution operations using SDN," *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 976–988, 2020.
- [28] S. Choo, J. Kim, and S. Pack, "Optimal task offloading and resource allocation in software-defined vehicular edge computing," in *Int. Conf. on information and communication technology convergence (ICTC)*, 2018, pp. 251–256.
- [29] Y. Sahni, J. Cao, and L. Yang, "Data-aware task allocation for achieving low latency in collaborative edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3512–3524, 2019.
- [30] S. Yang, F. Li, M. Shen, X. Chen, X. Fu, and Y. Wang, "Cloudlet placement and task allocation in mobile edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5853–5863, 2019.
- [31] L. Chen, J. Xu, and S. Zhou, "Computation peer offloading in mobile edge computing with energy budgets," in *IEEE GLOBECOM*, 2017, pp. 1–6.
- [32] X. Lyu, H. Tian, W. Ni, Y. Zhang, P. Zhang, and R. P. Liu, "Energy-efficient admission of delay-sensitive tasks for mobile edge computing," *IEEE Transactions on Communications*, vol. 66, no. 6, pp. 2603–2616, 2018.
- [33] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. on Vehi. Technology*, vol. 68, no. 1, pp. 856–868, 2018.
- [34] S. Josilo and G. Dán, "Computation offloading scheduling for periodic tasks in mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 667–680, 2020.
- [35] J. Yan, S. Bi, L. Duan, and Y.-J. A. Zhang, "Pricing-driven service caching and task offloading in mobile edge computing," *IEEE Transactions of Wireless Communications*, 2021.
- [36] C. Campolo, G. Genovese, A. Iera et al., "Virtualizing AI at the distributed edge towards intelligent IoT applications," *Journal of Sensor and Actuator Networks*, vol. 10, no. 1, p. 13, 2021.
- [37] Y. Cui, J. Song, M. Li, Q. Ren, Y. Zhang, and X. Cai, "SDN-based big data caching in ISP networks," *IEEE Transactions on Big Data*, vol. 4, no. 3, pp. 356–367, 2017.
- [38] M. Król and I. Psaras, "Nfaas: named function as a service," in *ACM Conf. on Information-Centric Networking*, 2017, pp. 134–144.
- [39] J. Lu, Z. Zhang, T. Hu, P. Yi, and J. Lan, "A survey of controller placement problem in software-defined networking," *IEEE Access*, vol. 7, pp. 24 290–24 307, 2019.
- [40] S.-W. Ko, K. Han, and K. Huang, "Wireless networks for mobile edge computing: Spatial modeling and latency analysis," *IEEE Trans. on Wireless Comm.*, vol. 17, no. 8, pp. 5225–5240, 2018.
- [41] Z. Li, V. Chang, H. Hu, D. Yu, J. Ge, and B. Huang, "Profit maximization for security-aware task offloading in edge-cloud environment," *Journal of Parallel and Distributed Computing*, 2021.
- [42] T. Zhao, S. Zhou, X. Guo, and Z. Niu, "Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing," in *IEEE ICC*, pp. 1–7.
- [43] C. Jiang, Y. Chen, Q. Wang, and K. R. Liu, "Data-driven stochastic scheduling and dynamic auction in IaaS," in *IEEE GLOBECOM*, pp. 1–6.
- [44] S. Vural, N. Wang, P. Navaratnam, and R. Tafazolli, "Caching transient data in internet content routers," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1048–1061, 2016.
- [45] Y. Mao, J. Zhang, and K. B. Letaief, "Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems," in *2017 IEEE wireless communications and networking conference (WCNC).* IEEE, 2017, pp. 1–6.
- [46] Y. Zhang, Y. Liu, J. Zhou, J. Sun, and K. Li, "Slow-movement particle swarm optimization algorithms for scheduling security-critical tasks in resource-limited mobile edge computing," *Future Generation Computer Systems*, vol. 112, pp. 148–161, 2020.
- [47] M. Zukerman, "Introduction to queueing theory and stochastic teletraffic models," *arXiv preprint arXiv:1307.2968*, 2013.
- [48] D. G. Cattrysse and L. N. Van Wassenhove, "A survey of algorithms for the generalized assignment problem," *European journal of operational research*, vol. 60, no. 3, pp. 260–272, 1992.
- [49] Silvano Martello and Paolo Toth, *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & Sons Inc, 1990.
- [50] P. Sermppezis, T. Giannakas, T. Spyropoulos, and L. Vigneri, "Soft cache hits: Improving performance through recommendation and delivery of related content," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1300–1313, 2018.
- [51] M. Sviridenko, "A note on maximizing a submodular set function subject to a knapsack constraint," *Operations Research Letters*, vol. 32, no. 1, pp. 41–43, 2004.
- [52] S. Lu, J. Wu, Y. Duan, N. Wang, and J. Fang, "Towards cost-efficient resource provisioning with multiple mobile users in fog computing," *Journal of Parallel and Distributed Computing*, vol. 146, pp. 96–106, 2020.
- [53] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femto-caching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.
- [54] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, "Multi-controller based software-defined networking: A survey," *IEEE Access*, vol. 6, pp. 15 980–15 996, 2018.
- [55] T. Subramanya, D. Harutyunyan, and R. Riggio, "Machine learning-driven service function chain placement and scaling in MEC-enabled 5G networks," *Computer Networks*, vol. 166, p. 106980, 2020.
- [56] M. S. Elbamby, M. Bennis, and W. Saad, "Proactive edge computing in latency-constrained fog networks," in *EuCNC 2017*, pp. 1–6.
- [57] J. Baliga, R. W. Ayre, K. Hinton, and R. S. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, 2010.
- [58] P. Van Mieghem, *Performance analysis of communications networks and systems.* Cambridge University Press, 2009.
- [59] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Information Processing Letters*, vol. 100, no. 4, pp. 162–166, 2006.
- [60] T. Zhu, J. Li, Z. Cai, Y. Li, and H. Gao, "Computation scheduling for wireless powered mobile edge computing networks," in *IEEE INFOCOM 2020*, pp. 596–605.
- [61] A. Mukhopadhyay, G. Iosifidis, and M. Ruffini, "Migration-aware network services with edge computing," *IEEE Transactions on Network and Service Management*, 2021.
- [62] M. Amadeo, G. Ruggeri, C. Campolo, and A. Molinaro, "IoT services allocation at the edge via named data networking: From optimal bounds to practical design," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 661–674, 2019.
- [63] D. He, D. Zhang, K. Xu, K. Huang, and Y. Li, "A fast and memory-efficient approach to NDN name lookup," *China Communications*, vol. 14, no. 10, pp. 61–69, 2017.



**Marica Amadeo** is an assistant professor at University of Reggio Calabria. She received a master degree (2008) in telecommunications engineering from the University of Reggio Calabria, and a Ph.D. degree in 2013 from the same University. Her major research interests are in the field of Information-Centric Networking, Internet of Things and Edge Computing.



**Claudia Campolo** is an associate professor of telecommunications at the University Mediterranea of Reggio Calabria. She received a master degree (2007) and a Ph.D. degree (2011) in telecommunications engineering from the same university. Her main research interests are in the field of vehicular networking, future Internet architectures, 5G and beyond systems.



**Gianmarco Lia** received the master degree in telecommunications engineering in 2019 from the University Mediterranea of Reggio Calabria, Italy. Since November 2019 he is a PhD student at the same university. His current interests are task offloading to the edge and Edge AI.



**Antonella Molinaro** is a full professor of telecommunications at the University Mediterranea of Reggio Calabria, Italy, and has a double affiliation with CentraleSupélec/L2S, Université Paris-Saclay, France. Previously, she was an assistant professor with the University of Messina (1998-2001) and the University of Calabria (2001-2004), and a research fellow at the Politecnico di Milano (1997-1998). She was with Siemens, Munich (1994-1995). Her current research focuses on 5G, vehicular networking and

future Internet architectures.



**Giuseppe Ruggeri** received the master degree in electronics engineering in 1998 from the University of Catania, Italy and, in 2002, he received the Ph.D. in electronics, computer science and telecommunications engineering from the University of Palermo, Italy. He is currently associate professor at the University Mediterranea of Reggio Calabria. His current interests include self organizing networks, Internet of Things, Social Internet of Things.