KPI Guarantees in Network Slicing

KPI Guarantees in Network Slicing

Jorge Martín-Pérez, *Member, IEEE,* Francesco Malandrino, *Senior Member, IEEE,*
Carla Fabiana Chiasserini, *Fellow, IEEE,* Milan Groshev, *Member, IEEE,* Carlos J. Bernardos

***Abstract*—Thanks to network slicing, mobile networks can now support multiple and diverse services, each requiring different key performance indicators (KPIs). In this new scenario, it is critical to allocate network and computing resources efficiently and in such a way that all KPIs targeted by a service are met. Accounting for all sorts of KPIs (e.g., availability and reliability, besides the more traditional throughput and latency) is an aspect that has been scarcely addressed so far and that requires tailored models and solution strategies. We address this issue by proposing a novel methodology and resource orchestration scheme, named OKpi, which provides high-quality decisions on VNF (Virtual Network Function) placement and data routing, including the selection of radio points of attachment. Importantly, OKpi has polynomial computational complexity and accounts for *all* KPIs required by each service, and for any resource available from the fog to the cloud. We prove several properties of OKpi and demonstrate that it performs very closely to the optimum under real-world scenarios. We also implement OKpi in a testbed supporting a robot-based, smart factory service, and we present some field tests that further confirm the ability of OKpi to make high-quality decisions.**

***Index Terms*—Network slicing, Network function virtualization, Service orchestration, multidimensional graphs.**

I. Introduction

Network slicing is a powerful concept that paves the way to the support of multiple, diverse vertical services in mobile networks. Vertical industries, such as automotive, e-health, and smart factories, can define the services to offer to mobile users through a set of virtual network functions (VNFs), connected according to the so-called VNF graph, and a set of performance requirements, i.e., the Key Performance Indicators (KPIs). The mobile network is then in charge of deploying and running such services, i.e., of *placing* and *connecting* the VNFs using suitable computing and network resources[1].

Importantly, the above task implies the selection of the radio points of attachment (PoAs) and the use of resources that may span from the cloud to the edge of the network infrastructure (including multi-access edge computing (MEC)), to the fog (e.g., connected cars and robots). Also, it is critical that VNFs are placed and connected, so as to (i) meet the target KPI values, (ii) make an efficient use of the different available resources, thus avoiding resource shortage, and (iii) minimize the service deployment cost, one of the main concerns for both mobile operators and vertical industries [2]–[5].

J. Martín-Pérez, M. Groshev, and C. J. Bernardos are with Universidad Carlos III de Madrid, Spain. F. Malandrino and C.-F. Chiasserini are with CNR-IEIIT, Italy. C.-F. Chiasserini is with Politecnico di Torino, Italy.

A preliminary version [1] of this work has been presented at the IEEE INFOCOM 2020 conference.

This work was supported by the EU Commission through the 5Growth project (grant agreement no. 856709).

[1]We focus on computing and networking resources, however our approach can handle memory and storage as well.

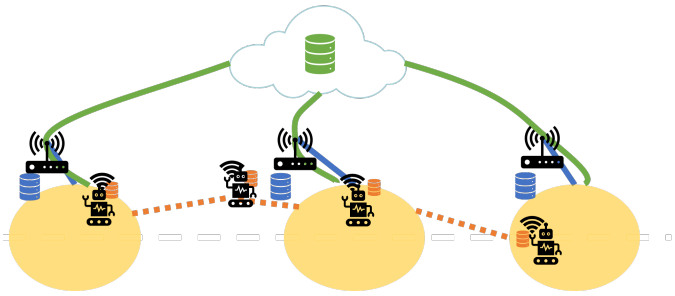


Fig. 1. As per geographical availability requirements, a mobile robot, smart factory service must be provided within the yellow areas, with high reliability and low latency. This can be obtained by deploying: (a) service instances at the robots (fog resources), at lower cost but also lower reliability, hence, needing redundancy to meet reliability constraints (orange option); (b) three instances at the points of attachment (PoA), e.g., access points (APs), covering the target areas (edge resources, blue option); (c) deploying only one instance in the cloud, but with larger delay (green option).

Although several works have already addressed the VNF placement problem (see Sec. VII for a detailed discussion), the variety of KPIs introduced by 5G poses some relevant challenges that still need to be solved. Such KPIs are indeed both diverse and heterogeneous over different services, as well depicted by the ubiquitous ITU "pyramid" [6]: *diverse* as, for instance, the latency requirements of different use cases can vary by several orders of magnitude, while *heterogeneous* reflects the fact that 5G introduces several new performance metrics, including service availability (in both space and time) and service reliability, as exemplified in Fig. 1. Satisfying all the relevant KPIs through an efficient resource allocation thus requires casting the problem into a new formulation and envisioning a totally new solution.

Additionally, existing studies have tackled to a limited extent specific aspects of network slicing, including (i) the possibility that already-deployed VNF instances can be reused for newly requested services, with [7] only accounting for cost, (ii) the opportunity of combining cloud- and edge-based services (with the exception of [8], which however only deals with caching), and (iii) the need to make decisions on how to place *and* connect VNFs, thus jointly addressing VNF placement and data routing ([9]–[12] do so, but without considering PoAs or VNF re-usage, and under some limiting assumptions, e.g., on the number of VNF instances). Overall, extending existing solutions to account for all 5G KPIs and the need for an efficient and low-cost resource utilization, would not be trivial and would, in general, jeopardize the complexity and/or competitive ratio properties of such solutions.

In this work, we propose a novel methodology to model the system, as well as the main features of network slicing. Exploiting such methodology, we develop OKpi, an effi-

cient solution that can create high-quality, end-to-end network slices. Specifically, our main contributions are as follows:

(i) we develop a system model that captures the main aspects of Network Function Virtualization (NFV)-based networks and can represent the availability of resources at different layers of the network topology, namely, cloud, edge, and fog, as well as the fact that existing VNF instances can be reused for newly-requested services[2];

(ii) we formulate an optimization problem that minimizes the resource cost, while meeting all target KPIs. We prove that the problem is NP-hard, and propose the OKpi solution, which has instead polynomial complexity. Leveraging a graph-based representation of the available resources, the possible decisions, and their impact on the KPIs, our scheme can make *joint* decisions on VNF placement and traffic routing that minimize the resources cost, by applying a shortest path algorithm over a multi-dimensional graph. Importantly, such a graph can be built with different levels of detail and size, which results into a tuneable trade-off between computational complexity and decision quality;

(iii) we analyze the properties of our solution and, through numerical results derived under real-world automotive and robot scenarios, we show that OKpi closely matches the optimal performance. Furthermore, we show its functionality by implementing it in a testbed supporting a mobile robot, smart factory service.

The rest of the paper is organized as follows. We introduce the system model in Sec. II, and the problem formulation in Sec. III. The OKpi solution and algorithm are described in Sec. IV, where we also prove several properties of OKpi and discuss its computational complexity. Sec. V shows the performance of our solution through simulations in both small- and large-scale scenarios referring to 5G use-cases, while Sec. VI presents some field tests obtained through a real-world testbed. Finally, we review related works in Sec. VII and conclude the paper in Sec. VIII.

II. System model

Our model concisely describes the two main components of mobile, slicing-based networks: the services they support (Sec. II-A), and the computing and network resources they include (Sec. II-B). Each of them is modeled through a graph – the service graph and the physical graph, respectively. We then describe how such graphs can be combined in Sec. III-C. Further, we consider that a monitoring platform is in place, with the aim to periodically monitor both the service performance and the status of the system resources. In the following, we denote the generic time interval over which the system metrics are periodically monitored by t, and the set of such intervals by $\mathcal{T}$. The notations used throughout the paper are also summarized in Tab. I.

[2]This is feasible if services share a common subset of VNFs and no service isolation constraints exist.

TABLE I
Notation table

Symbol	Type	Meaning
$\mathcal{T}=\{t\}$	Set	Set of time intervals
$\mathcal{S}=\{s\}$	Set	Set of vertical services
$\mathcal{V}=\{v\}$	Set	Set of VNFs
$\mathcal{A}=\{\alpha\}$	Set	Set of locations
$\mathcal{E}=\{\psi\}$	Set	Set of endpoints
$\mathcal{C}=\{c\}$	Set	Set of network nodes
$\mathcal{K}=\{\kappa\}$	Set	Set of resources
$\mathcal{I}=\{i\}$	Set	Set of radio interfaces
$\mathcal{L}=\{(i,j)\}$	Set	Set of physical links
$\mathcal{W}=\{w\}$	Set	Set of *strings*/paths
$k(\kappa,c)$	Parameter	Quantity of κ resources at node c
$r_\kappa(v)$	Parameter	Quantity of κ resources required by VNF v to process a unit of traffic
$R_i(c)$	Parameter	Whether node c is equipped with radio interface i
$D_{i,j}$	Parameter	Delay of link (i,j)
$C_{i,j}$	Parameter	Traffic capacity of link (i,j)
$\eta(c,t),\eta(i,j,t)$	Parameter	Reliability of node c and link (i,j) at time interval t
$l(\psi,v_1,v_2)$	Parameter	Traffic originated at π, processed last at v_1, before being processed at v_2
$\chi(v_1,v_2,v_3)$	Parameter	Fraction of traffic processed at v_1, currently processed at v_2, latter processed at v_3
$\rho(v,c)$	Variable	Whether node c hosts VNF v
$a_c(\psi,v,\kappa)$	Variable	Quantity of κ resources assigned to VNF v at node c to process traffic from ψ
$\hat{f}_c(\psi,v_1,v_2)$	Variable	Fraction of flow $l(\psi,v_1,v_2)$ processed at VNF v in node c
$p_{i,j}(\psi,v_1,v_2)$	Variable	Traffic from ψ, traversing link (i,j), processed at v_1, latter processed by v_2 at node j
$t_{i,j}(\psi,v_1,v_2)$	Variable	Traffic originated at ψ, last processed at v_1, just transiting link (i,j), and to be processed by v_2
$p_{i,j}(\psi,v_1,v_2,w)$	Variable	Processing traffic $p_{i,j}(\psi,v_1,v_2)$ traversing *string* w
$t_{i,j}(\psi,v_1,v_2,w)$	Variable	Transiting traffic $t_{i,j}(\psi,v_1,v_2)$ traversing *string* w
$f(\psi,v_1,v_2,w)$	Variable	Fraction of service flow $l(\psi,v_1,v_2)$ traversing *string* w
$\lambda_c(\psi,v)$	Auxiliary Variable	Quantity of traffic originated at ψ and processed by v at node c
$\tau_{i,j}(\psi,v_1,v_2)$	Auxiliary variable	Traffic originated at ψ, traversing link (i,j), last processed at v_1, and to be processed by v_2
$\tau_{i,j}(\psi,v_1,v_2,w)$	Auxiliary variable	Traversing traffic $\tau_{i,j}(\psi,v_1,v_2)$ traveling over *string* w

A. Services

A vertical service $s \in \mathcal{S}$ is described through a *service graph* where vertices are VNFs, $v \in \mathcal{V}$, and edges specify in which order the VNFs should process the related data traffic (i.e., how data shall be routed from a VNF instance running on a network node to the next). Note that VNFs can also represent database-related functionalities [13], requiring storage resources: like other VNFs, they must be placed on a node and consume resources therein. An example of service graph for a mobile robot, smart factory use case[3] is depicted in Fig. 2(left).

A service s is associated with one or more ***KPIs***, namely,

- the required bandwidth, or expected traffic load l to be transferred and handled by the VNFs composing the service;
- the maximum allowed delay $D(s)$;
- the minimum level of reliability $H(s)$;
- the required geographical availability at a subset of locations, $A(s) \subseteq \mathcal{A}$, where $\mathcal{A} = \{\alpha\}$ represents the set

[3]http://wiki.ros.org

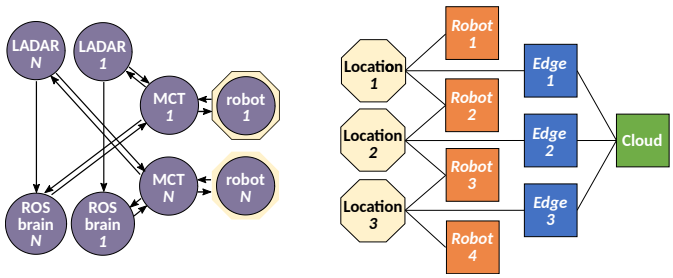


Fig. 2. Service (left) and physical (right) graphs corresponding to the example in Fig. 1. In the mobile robot, smart factory graph, each robot transmits its sensors data to the LADAR and the the Robot OS (ROS) brain. The former provides a probabilistic localization of the robots, the latter leverages such a localization and the sensors data to control the robots. Messages are transferred through the Mobile Communication Transport (MCT), e.g., a virtual AP. In the service graph, vertices are endpoints (yellow) or VNFs (purple), edges are directed and correspond to flows l. In the physical graph, vertices are endpoints in $\mathcal{E}$ or nodes in $\mathcal{C}$, and edges are undirected and represent links in $\mathcal{L}$. Colors correspond to those in Fig. 1 and refer to the different resource locations: fog (orange), edge (blue), cloud (green).

of all possible locations in the considered region. As an example, $A(s)$ can represent the urban intersections where an automotive vertical wants to provide a safety service, or the areas where robots should move within a warehouse (Fig. 1). We refer to the combination of a service and a location as an *endpoint* $\psi = (\alpha, s) \in \mathcal{E} \subseteq \mathcal{A} \times \mathcal{S}$;

- the lifetime (or temporal availability) $\varphi(\psi) \subseteq \mathcal{T}$, corresponding to a subset of all time intervals $\mathcal{T}$ during which the service must be available at endpoint ψ.

As foreseen by standards [14], services may be associated with one or more of these requirements, i.e., not all KPIs have to be specified for all services. Also, without loss of generality, we consider that the traffic associated with a service is generated at endpoint ψ and has to be processed by the VNFs in the service graph; in Fig. 2(left), this would correspond to uplink data transfers. Note however that, as discussed later, our model is general and can also capture downlink as well as bidirectional traffic patterns.

The quantity of traffic originated at endpoint $\psi \in \mathcal{E}$, that has been processed last at VNF v_1, and will be next processed at VNF v_2 is denoted by $l(\psi, v_1, v_2)$ (with $l(\psi, v, v)$ being the traffic that will be processed for the first time at v). After a traffic flow is processed at a VNF, the outgoing traffic can increase, decrease, or be split among several other VNFs, according to the service graph. Parameters $\chi(v_1, v_2, v_3)$ express the fraction of the traffic that was last processed (or originated) at $v_1 \in \mathcal{V} \cup \mathcal{E}$, that is currently processed at v_2, and that will next be processed at v_3. For instance, if v_2 is a deep packet inspector, $\chi(v_1, v_2, v_3) = 1$; but if v_2 is a firewall, then $\chi(v_1, v_2, v_3) \leq 1$.

B. Radio coverage and Fog/Edge/Cloud resources

Network nodes, with switching or computing capabilities, are denoted by $c \in \mathcal{C}$, while endpoints, which are origins or destinations of service traffic, are denoted by $\psi \in \mathcal{E}$. Nodes may be equipped with different resources, e.g., CPU or memory; the set of resources is identified by $\mathcal{K} = \{\kappa\}$. The quantity of resource type κ available at node c is specified through parameters $k(\kappa, c)$, hence, $k(\kappa, c) = 0\ \forall \kappa$ for pure network equipment like traditional, non-software, switches. Also, binary parameters $R_i(c)$ express whether node c is equipped with radio interface $i \in \mathcal{I}$ or not. A radio interface available at node c determines which locations, hence endpoints, node c covers – an important feature of fog and edge nodes.

Radio coverage, fog, edge, and cloud resources can then be represented through a *physical graph* whose vertices are the network nodes and the endpoints, and the edges $(i, j) \in \mathcal{L} \subseteq (\mathcal{C} \cup \mathcal{E})^2$ represent the physical links connecting them, as per the network topology and the coverage provided by the radio interfaces. Each edge (i, j) is associated with delay $D_{i,j}$ and traffic capacity $C_{i,j}$. Also, we denote the *reliability* level of any node c and link (i, j) monitored over a time interval $t \in \mathcal{T}$, by $\eta(c, t)$ and $\eta(i, j, t)$, respectively. Specifically, these quantities express the probability that a specific node or link works as intended by averaging their behavior over $t \in \mathcal{T}$, thus accounting for the time-varying quality of communication links involving fog nodes, e.g., robots or cars. We remark that, in general, all the parameters introduced above may differ across fog, edge, and cloud resources.

C. Service support over the physical graph

To express whether a node c in the physical graph hosts VNF v, we introduce a binary variable, $\rho(v, c) \in \{0, 1\}$. Variables $a_c(\psi, v, \kappa)$, instead, express the quantity of resources of type κ assigned to that VNF v at node c and used to process traffic generated at endpoint ψ.

We also introduce variables $\tau_{i,j}(\psi, v_1, v_2)$ representing the flows over the physical graph, or, more specifically, the traffic originated at $\psi \in \mathcal{E}$, traversing $(i, j) \in \mathcal{L}$, last processed at v_1, and to be next processed at v_2. Such traffic can be either processed at j, or just transiting through j; these two options are described through the two real variables $p_{i,j}(\psi, v_1, v_2)$ and $t_{i,j}(\psi, v_1, v_2)$, and by imposing: $\tau_{i,c}(\psi, v_1, v_2) = p_{i,c}(\psi, v_1, v_2) + t_{i,c}(\psi, v_1, v_2)$.

Further, to handle the service KPIs more easily, we define a *string*, $w \in \mathcal{W}$, over the physical graph as a sequence of physical links traversed by a flow, with the first component of the string being an endpoint. Similarly to [15], the possible strings can be pre-computed and stored for later usage. Since a service flow can be split across different strings, we define $f(\psi, v_1, v_2, w)$ as the fraction of service flow $l(\psi, v_1, v_2)$ traversing string w. Clearly, such fractions must sum to 1.

The string-wise equivalents to $\tau_{i,j}(\psi, v_1, v_2)$, $t_{i,j}(\psi, v_1, v_2)$, and $p_{i,j}(\psi, v_1, v_2)$ are then $t_{i,j}(\psi, v_1, v_2, w)$ and $p_{i,j}(\psi, v_1, v_2, w)$, respectively. Specifically, $\tau_{i,j}(\psi, v_1, v_2, w)$ represents the traffic of service flow $l(\psi, v_1, v_2)$ traversing link (i, j) on its journey through string $w \in \mathcal{W}$, and then we impose $\tau_{i,j}(\psi, v_1, v_2) = \sum_{w \in \mathcal{W}} \tau_{i,j}(\psi, v_1, v_2, w)$. Similar definitions and conditions hold for $t_{i,j}(\psi, v_1, v_2, w)$ and $p_{i,j}(\psi, v_1, v_2, w)$.

Furthermore, the fraction of service flow over a certain string w must match the physical traffic on the corresponding links, i.e., for all endpoints, VNFs v_1 and v_2, links, and strings, we have: $f(\psi, v_1, v_2, w) l(\psi, v_1, v_2) = \tau_{i,j}(\psi, v_1, v_2) \mathbb{1}_{w(i,j)}$, where $\mathbb{1}_{w(i,j)}$ denotes that link $(i, j) \in \mathcal{L}$ belongs to w.

Now that we have defined the *string*-related variables, we provide an expression for the latency and reliability KPIs of a service as set forth below.

The service latency comprises of network delay, due to traffic traversing links and switches, and processing times at the nodes hosting VNF instances. Given endpoint ψ, the average network delay can be computed as the weighted sum of the delays associated with the individual strings taken by the traffic originated at ψ:

$$d_{\text{net}}(\psi) = \sum_{w \in \mathcal{W}} \sum_{v_1, v_2 \in \mathcal{V}} f(\psi, v_1, v_2, w) \sum_{(i,j) \in w} D_{i,j}. \quad (1)$$

As for the processing time, let $\hat{f}_c(\psi, v_1, v_2)$ be the fraction of the service traffic flow $l(\psi, v_1, v_2)$ processed at the instance of VNF v_2 located at node c. Then the quantity of traffic $\lambda_c(\psi, v_2)$ originating at ψ and processed at the instance of v_2 in c is:

$$\lambda_c(\psi, v_2) = \sum_{v_1 \in \mathcal{V}} \hat{f}_c(\psi, v_1, v_2) l(\psi, v_1, v_2).$$

Note that such traffic may come from different physical links.

Next, we model VNF instances as M/M/1-PS queues (see, e.g., [16]–[18]); the choice of the processor sharing (PS) policy closely emulates the behavior of a multi-threaded application running on a virtual machine. The total processing time at the instance of v_2 deployed at node c can thus be written as: $1/(a_c(\psi, v_2, \text{cpu}) - r_{\text{cpu}}(v_2)\lambda_c(\psi, v_2))$, with $r_{\text{cpu}}(v_2)$ denoting the amount of CPU needed by VNF v_2 to process one unit of traffic. Summing over all flows, the total processing delay incurred by traffic originating at ψ is given by:

$$d_{\text{proc}}(\psi) = \sum_{v_1, v_2 \in \mathcal{V}, c \in \mathcal{C}} \hat{f}_c(\psi, v_1, v_2) \frac{1}{a_c(\psi, v_2, \text{cpu}) - r_{\text{cpu}}(v_2)\lambda_c(\psi, v_2)}. \quad (2)$$

Finally, notice that the reliability of a string can be computed as the product between the reliability values of all links and nodes belonging to it.

III. Problem formulation

In this section, we formalize the problem of creating end-to-end network slices that meet all the required KPI targets (Sec. III-E) while minimizing the total cost (Sec. III-F). First, we introduce the system constraints related to service processing, data routing, and KPI fulfillment. The problem complexity is then discussed in Sec. III-G.

A. Flow conservation on the service graph

First, as remarked by the example on the $\chi(\cdot)$ values, we note that there is no flow conservation on the service graph. Instead, the following *generalized flow conservation* law holds:

$$l(\psi, v_2, v_3) = \sum_{v_1:\, v_1 \neq v_2} l(\psi, v_1, v_2)\chi(v_1, v_2, v_3) + l(\psi, v_2, v_2)\chi(\psi, v_2, v_3), \quad \forall v_2, v_3 \in \mathcal{V} : v_2 \neq v_3\,. \quad (3)$$

The intuitive meaning of (3) is that either traffic traveling from VNF v_2 to VNF v_3 must come from another VNF v_1 and then it is transformed in v_2 according to the χ-coefficients (first term of the second member), or it has just originated at ψ and is processed for the first time at v_2 (second term).

B. Flow conservation and link capacity on the physical graph

The traffic going out of node c must be equal to the sum of that transiting through c and that just processed at c, i.e.,

$$\sum_{(c,h) \in \mathcal{L}} \tau_{c,h}(\psi, v_2, v_3) = \sum_{(i,c) \in \mathcal{L}} \Big[t_{i,c}(\psi, v_2, v_3) + p_{i,c}(\psi, v_2, v_2) \cdot \chi(\psi, v_2, v_3) + \sum_{v_1 \in \mathcal{V}} p_{i,c}(\psi, v_1, v_2)\chi(v_1, v_2, v_3) \Big]. \quad (4)$$

Finally, each physical link (i, j) cannot carry more traffic than its capacity, i.e., $\sum_e \sum_{v_1, v_2} \tau_{i,j}(\psi, v_1, v_2) \leq C_{i,j}$.

C. Deploying VNFs and assigning resources

Given a set of VNFs, each consuming an amount of resources $a_c(\psi, v, \kappa)$ of type κ at node c, we have to impose that the node capabilities are never exceeded, i.e., for any c and κ, $\sum_{\psi \in \mathcal{E}} \sum_{v \in \mathcal{V}} a_c(\psi, v, \kappa) \leq k(\kappa, c)$.

Importantly, for any $\kappa \in \mathcal{K}$, the quantity of traffic processed by v at node c cannot exceed the ratio between the quantity $a_c(\psi, v, \kappa)$ of resource type κ assigned to the VNF, and the quantity $r_\kappa(v)$ of resource type k needed by VNF v to process one unit of traffic, i.e.,

$$\sum_{(i,c) \in \mathcal{L}} \sum_{\psi \in \mathcal{E}} \sum_{v_1 \in \mathcal{V}} p_{i,c}(\psi, v_1, v_2) \leq \frac{a_c(\psi, v_2, \kappa)}{r_\kappa(v_2)} \quad \forall \kappa \in \mathcal{K}\,. \quad (5)$$

Also, node c's resources can be assigned to a VNF v only if the latter is deployed therein: $a_c(\psi, v, \kappa) \leq \rho(v, c)k(\kappa, c)$, for any c, κ, and v. These conditions imply that no traffic is processed at a node where no instance of a VNF is deployed.

Last, we ensure that VNFs are placed only at nodes where all the needed radio interface(s) are available, e.g., a Mobile Communication Transport (MCT) may work only at nodes equipped with specific radio interfaces. Thus, for any node c, interface i, and VNF v, we have: $\rho(v, c)r_i(v) \leq R_i(c)$, where $r_i(v) \in \{0, 1\}$ are parameters specifying whether interface i is needed by VNF v, and $R_i(c)$ specifies whether such an interface is available at c.

D. Matching service and physical flows

Since our system model includes two graphs, a service graph and a physical graph, we must ensure that service flows l and physical flows τ match. To this end, we impose that the flow entering the first VNF of a service graph corresponds to one or more traffic flows on the physical graph:

$$l(\psi, v, v) = \sum_{(\psi, c) \in \mathcal{L}} \tau_{e,c}(\psi, v, v), \forall \psi \in \mathcal{E}, v \in \mathcal{V}. \quad (6)$$

Once (6) is met, then (3) and (4) ensure that the traffic on subsequent links is processed as specified by the χ-parameters.

E. Meeting service KPIs

1) Service latency: the latency experienced by a service s is given by the sum of the network delay (as in (1)) and the processing time (as in (2)). Recalling that $D(s)$ is

the maximum target delay for service s, the service latency constraint for its endpoints can be stated as:

$$d_{\text{net}}(\psi) + d_{\text{proc}}(\psi) \leq D(s), \quad \forall \psi \in \mathcal{E}.$$

Note that the relationship between assigned CPU and processing time in the expression of $d_{\text{proc}}(\psi)$ also means that the CPU has a different role from the other types of resources. Indeed, for resources other than CPU, we can assign to each VNF instance exactly the amount needed to honor (5), as a greater amount would yield no benefit. With CPU, instead, there is an additional degree of freedom we can play with: assigning more CPU results in shorter processing times, but higher costs.

2) Service geographical availability: by service availability requirements, all locations in $A(s) \subseteq \mathcal{A}$ must be covered by service s. In other words, for all endpoints $\psi = (\alpha, s)$: $\alpha \in A(s)$, there must be a link (ψ, c) on the physical graph to a node c that is equipped with a radio interface covering α and that runs (or it is connected to another node running) the first VNF of the service graph.

3) Service reliability and temporal availability: We can ensure that at every monitoring slot the reliability $H(s)$ required for service s is honored by considering a weighted sum of the per-string reliability values. In symbols, $\forall \psi \in \mathcal{E}, t \in \varphi(\psi)$,

$$\prod_{v_1, v_2 \in \mathcal{V}} \sum_{w \in \mathcal{W}} f(\psi, v_1, v_2, w) \prod_{(i,j) \in w} \eta(j,t)\eta(i,j,t) \geq H(s)\,.$$

Note that imposing the above constraint for every monitoring slot during the service lifetime also ensures that the service target temporal availability is met.

F. Objective

As mentioned in Sec. I, cost is one of the main concerns related to service virtualization and network slicing. Such cost mainly comes from using network and computation resources. To model this issue, we define:

- a fixed cost $c_c(v)$, due to the creation at node c of a VNF instance v; this cost is null if an existing VNF instance can be reused;
- a cost $c_c(\kappa)$, incurred when using a unit resource κ at node c;
- a cost $c_{i,j}$, incurred when one unit traffic traverses link (i, j).

Then, upon receiving a request to deploy a service instance s, we formulate the following cost-minimization problem:

$$\min \sum_c \sum_v \left[c_c(v) + \sum_e \sum_\kappa c_c(\kappa) a_c(\psi, v, \kappa) \right] + \sum_{(i,j)} \sum_e \sum_{v_1, v_2} c_{i,j} \tau_{i,j}(\psi, v_1, v_2)$$

subject to the constraints reported in Secs. III-A–III-E.

We recall that the endpoints ψ to consider depend on the service and on its geographic availability requirements, while the VNFs are those specified by the service graph. Furthermore, a solution to the above problem will always opt for reusing an existing instance of a VNF, whenever possible, as this would nullify the instantiation cost $c_c(v)$.

G. Nature and complexity of the problem

The problem of jointly making VNF placement and data routing decisions is notoriously hard, even when only one KPI is considered [5], [16], [19]. We now prove through a reduction from bin-packing the complexity of the problem described in Sec. II, showing that directly solving such a problem is impractical for all but very small instances.

Theorem 1. *The VNF-placement and data routing problem described in Sec. II is NP-hard.*

Proof: To prove the thesis, we need to reduce an NP-hard problem to VNF-placement in polynomial time. Let us consider bin-packing, which is known to be NP hard [20]: given a set of *items* weighting ω_i each, we have to place them throughout a set of *bins*, each having size σ_b, using as few bins as possible. We transform a bin-packing instance into a corresponding VNF placement one, by considering:

- one single VNF v and one single location;
- infinite-capacity, zero-cost, zero-delay, unitary-reliability links;
- as many nodes as there are bins, also with unitary reliability;
- the CPU available at each node is the same as the size of the corresponding bin;
- one single location and as many services (hence, endpoints) as there are items;
- all services include only one VNF, i.e., VNF v;
- the traffic $l(\psi, v, v)$ and the target delay $D(s)$ of each service are such that it requires ω_i CPU units to process the service traffic in time, i.e., $\frac{1}{\omega_i - l(\psi,v,v) r_{\text{cpu}}(v)} = D(s)$;
- all costs are set to zero, except for the VNF creation costs $c_c(v)$, which can be set to any positive value.

In this case, VNF placement and bin-packing decisions are equivalent: the former places VNFs in nodes, the latter places items in bins. The size of bins corresponds to the capacity needed by the VNF instances, and minimizing the cost is tantamount to minimizing the number of bins. The translation from bin-packing to the above simple VNF placement (with only one VNF, single-VNF services, and uniformly-priced nodes) takes polynomial (indeed, linear) time, hence, we can conclude that VNF placement is (at least) as hard as bin-packing, i.e., NP-hard. Additionally, due to the infinite-capacity, zero-cost, zero-delay, unitary-reliability links, any data routing solution would be optimal. This suggests that, in practice, solving to optimality the problem described in Sec. II would be substantially harder than bin-packing. ■

We also observe that our problem can be seen as a more complex version of a multi-constrained path (MCP) problem, where the cost (hence, the weight of the edges in the MCP graph) *changes* at every hop. Although known solutions to the MCP problem, e.g., [21], are not applicable, such a similarity motivates us to propose an effective and efficient heuristic, called OKpi, for which we can prove that:

- it provides high-quality VNF placement and data routing decisions, with guaranteed feasibility;
- such decisions are made in polynomial time;
- under mild homogeneity assumptions, decisions are optimal;

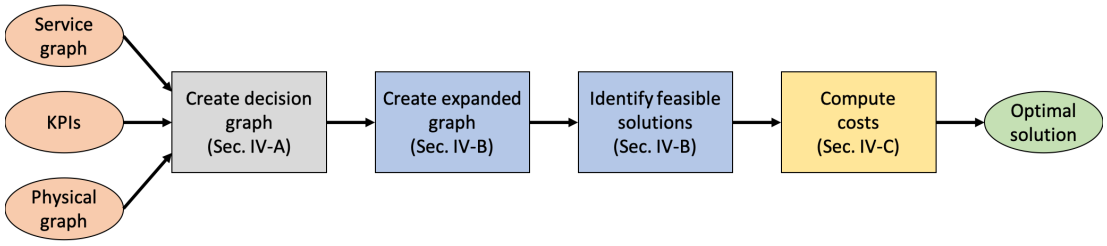


Fig. 3. The main steps of the OKpi solution concept.

- in the general case, decisions can be arbitrarily close to the optimum.

IV. The OKpi solution

Our solution includes four main steps, as summarized in Fig. 3. First, we combine the physical graph, the service graph, and KPI targets into a *decision graph* $\widetilde{G} = (\widetilde{N}, \widetilde{E})$, summarizing the service deployment decisions that can be made and their effect on the KPIs (Sec. IV-A). Then we translate this graph into an *expanded graph*, and use the latter to identify a set of feasible decisions as well as to select, among them, the lowest-cost one (Secs. IV-B and IV-C).

For clarity, we present OKpi in the case where the service graph is a chain with uplink traffic starting from an endpoint ψ, and including N VNFs $v_1 \dots v_N$, each requiring only one instance. As discussed in Sec. IV-D, all such limitations can be dropped: OKpi works with arbitrary service graphs requiring any number of instances for each VNF.

A. The decision graph

Given the physical graph modeling the service endpoints and the fog, edge, and cloud resources, we build the decision graph $\widetilde{G}$ with the aim to represent the possible service deployment decisions and their effects on the service KPIs.

As a preliminary step, we consider the *computation-capable* nodes in the physical graph (hence, a subset of $\mathcal{C}$), and for each of them we create $(|\mathcal{V}| - 1)$ replicas. Consistently, we create auxiliary edges (i) connecting each node c and its replicas in a chain fashion, and assign them zero delay, infinite capacity, and reliability 1, and (ii) connecting any replica of c with any computing node d, for which a link $(c, d) \in \mathcal{L}$ exists. Crucially, introducing node replicas enables us to account for the possibility to deploy multiple VNFs at the same computation-capable node without introducing self-loops in the decision graph. Indeed, as it will be more clear later, given that a VNF is placed in c, each replica thereof represents the possibility to deploy the next VNF again in c.

Let then $\widetilde{G} = (\widetilde{N}, \widetilde{E})$ be the decision graph where:

- $\widetilde{N}$ includes the endpoints in $\mathcal{E}$, and the computation-capable nodes in the physical graph as well as their replicas;
- $\widetilde{E}$ is the set of (i) the aforementioned auxiliary links, and (ii) the virtual links (i.e., single physical links or sequences thereof) connecting the vertices in $\widetilde{N}$.

Every edge $(\tilde{n}_1, \tilde{n}_2)$ in $\widetilde{E}$ representing a virtual link has the following properties:

- its capacity $\widetilde{C}_{\tilde{n}_1,\tilde{n}_2}$ is set to the minimum of the individual capacities of the physical links composing the virtual link;
- its delay $\widetilde{D}_{\tilde{n}_1,\tilde{n}_2}$ is set to the sum of the individual delays of the physical links composing the virtual link;
- its reliability $\widetilde{\eta}_{\tilde{n}_1,\tilde{n}_2}$ is set to the product of the reliability values of physical links and nodes (both computation and pure-routing capable) included in the virtual link.

Let us now consider the additive KPIs and, for simplicity, let us focus on two of them, e.g., delay and reliability. To any edge $(\tilde{n}_1, \tilde{n}_2)$ in the decision graph, we assign a *multi-dimensional weight* $\tilde{w}(\tilde{n}_1, \tilde{n}_2)$, defined as:

$$\tilde{w}(\tilde{n}_1, \tilde{n}_2) = \left(\frac{\widetilde{D}_{\tilde{n}_1,\tilde{n}_2}}{D(s)}, \frac{\log \widetilde{\eta}_{\tilde{n}_1,\tilde{n}_2}}{\log H(s)} \right). \tag{7}$$

The intuition behind (7) is that the weight of edge $(\tilde{n}_1, \tilde{n}_2)$ corresponds to the fraction of the target delay and reliability that will be "consumed" by taking that edge, i.e., by deploying a VNF at $\tilde{n}_1$ and the subsequent one at $\tilde{n}_2$. We stress that using logarithms in the second term of the weight allows us to translate a multiplicative performance index (namely, reliability) into an additive one[4].

We stress that, when some services are already active in the network, we build the decision graph considering the *residual* capabilities of physical links and nodes, i.e., those not assigned to already-running services. Similarly, in case of virtual links sharing the same physical links, their capacity is updated as traffic is allocated to the physical links.

B. The expanded graph: finding decisions honoring availability and additive KPIs

Given the decision graph $\widetilde{G}$, our first purpose is to identify a set of *feasible* service deployment decisions that are consistent with the target KPIs. To this end, as a preliminary step, we ensure to meet the service geographical and temporal availability requirements by pruning from $\widetilde{G}$ the vertices and edges that do not satisfy such constraints.

Next, we take an approach inspired by [21] and build a multi-dimensional, *expanded graph*, with as many dimensions as the number of additive KPIs. Specifically, given a positive integer value of resolution γ:

1) for each vertex $\tilde{n}$ in the decision graph, we create as many corresponding vertices as $(\gamma + 1)^2$, where the exponent 2

[4]It is easy to see that $\widetilde{\eta}_{\tilde{n}_1,\tilde{n}_2} \widetilde{\eta}_{\tilde{n}_2,\tilde{n}_3} \geq H(s)$ translates into $\frac{\log \widetilde{\eta}_{\tilde{n}_1,\tilde{n}_2}}{\log H(s)} + \frac{\log \widetilde{\eta}_{\tilde{n}_2,\tilde{n}_3}}{\log H(s)} \leq 1$.

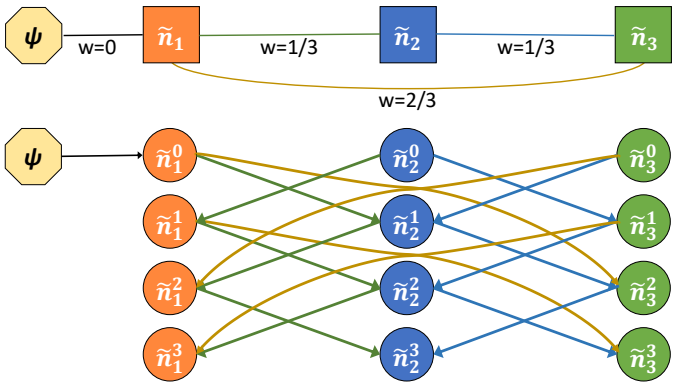


Fig. 4. Decision graph (top) and expanded graph (bottom) when only delay is considered as a KPI and $\gamma = 3$. In the decision graph, edges $(\tilde{n}_1, \tilde{n}_2)$ and $(\tilde{n}_2, \tilde{n}_3)$ have delay of 1 ms, while $(\tilde{n}_1, \tilde{n}_3)$ has delay of 2 ms (vertices representing replica nodes are omitted for simplicity); the target delay is 3 ms.

corresponds to the number of additive KPIs. We denote such vertices by $\tilde{n}^{\mathbf{d}}$ where $\mathbf{d}$ is a vector with as many integer elements as the number of additive KPIs and the value of such elements ranges between 0 and γ, i.e., $\tilde{n}^{\mathbf{d}} = \tilde{n}^{0,0}, \tilde{n}^{0,1} \ldots, \tilde{n}^{0,\gamma} \ldots, \tilde{n}^{\gamma,\gamma}$;

2) for every edge $(\tilde{n}_1, \tilde{n}_2) \in \widetilde{E}$ with capacity $\widetilde{C}_{\tilde{n}_1,\tilde{n}_2}$ greater or equal to the amount of traffic to process, create directed edges from each vertex $\tilde{n}_1^{i,j}$ to vertex $\tilde{n}_2^{i+\lceil \gamma w(\tilde{n}_1,\tilde{n}_2)[0] \rceil, j+\lceil \gamma w(\tilde{n}_1,\tilde{n}_2)[1] \rceil}$ (if such a vertex exists), where the two superscripts refer to delay and reliability, respectively. For example, with $\gamma = 1$, if edge $(\tilde{n}_1, \tilde{n}_2)$ has weight $\tilde{w}(\tilde{n}_1, \tilde{n}_2) = (0.1, 1.5)$ and enough traffic capacity, there will be a directed edge from $\tilde{n}_1^{0,0}$ to $\tilde{n}_2^{1,2}$, but not from $\tilde{n}_1^{0,0}$ to $\tilde{n}_2^{0,1}$.

We stress that the expanded graph has *no weights* on its edges: the delay and reliability information that is expressed by weights in the decision graph is now represented by the topology of the expanded graph. A one-dimensional (i.e., one-KPI) example of decision graph and corresponding expanded graph is depicted in Fig. 4.

Finally, we identify a set of possible service deployments, i.e., VNF-to-compute node assignments and the corresponding data routing. We do so by looking for the shortest paths in the expanded graph that (a) begin at endpoints, and (b) contain as many edges as there are VNFs to place. We underline that the latter is trivially required by the need to deploy all VNFs on the service graph, and by the fact that placing more VNFs at a physical node is allowed thanks to the replica nodes and auxiliary edges, as discussed in Sec. II-B.

In the following, we make several fundamental remarks on the expanded graph and on the paths, hence, the deployment decisions they correspond to. Given KPI, we define as *depth* of a vertex in the expanded graph the value of the element in the superscript corresponding to the KPI. Note that, by construction (see point 1 above), the maximum value of depth is γ. Also, let the *steepness* of an edge be the difference in depth between its target and source vertices. Considering the one-KPI example in Fig. 4(bottom), vertex $\tilde{n}_1^0$ has depth 0, vertex $\tilde{n}_3^2$ has depth 2, and the edge between the two has steepness $2 - 0 = 2$, i.e., equal to $\lceil \gamma w(\tilde{n}_1, \tilde{n}_3) \rceil$.

By construction, for a given KPI, the ratio between the steepness of an edge and γ is greater or equal to the weight component on the corresponding edge of the decision graph, which in turn is the fraction of the KPI target values consumed by making that decision (see (7)). As an example, considering edge $(\tilde{n}_1^0, \tilde{n}_3^2)$ in Fig. 4(bottom), we have:

$$\frac{\text{steepness}}{\gamma} = \frac{2}{3} \geq w(\tilde{n}_1^0, \tilde{n}_3^2) = \frac{D_{\tilde{n}_1,\tilde{n}_3}}{D(s)} = \frac{2}{3}. \tag{8}$$

The observations above allow us to state a very relevant property of the decisions corresponding to the paths on the expanded graph.

Lemma 1. *The decisions corresponding to any path on the expanded graph honor all additive KPIs.*

Proof: By definition, the depth of a vertex corresponds to the total steepness of the path required to reach it from endpoint ψ. Given that the maximum depth in the expanded graph is γ, there is *no path* with total steepness[5] greater than γ. Thanks to the relation between weight and KPI targets (exemplified in (8)), this implies that, given a path on the expanded graph, the sum of the weights of the corresponding edges in the decision graph cannot exceed 1, i.e., the corresponding decisions honor additive KPIs (including, thanks to the logarithmic weights, reliability). ■

Importantly, the smaller the resolution γ, the fewer the possible values of depth and steepness in the expanded graph, the fewer the levels of consumption of the KPI target values we are able to distinguish, which corresponds to introducing an error, akin to quantization. Indeed, $\gamma+1$ can be seen as the number of quantization levels[6] we admit: in the extreme case of $\gamma = 1$, all edges would have a steepness of 1, which also corresponds to exhausting the whole KPI target in one hop. Such a quantization error may lead to discarding some feasible solutions, and thus, in the most general case, may jeopardize the optimality of OKpi. However, two important facts stand out: *(i)* even enumerating all feasible paths in the decision graph is NP-hard, as proven in [21], hence, quantization is necessary; *(ii)* by increasing γ, OKpi can get *arbitrarily close to the optimum* (at the price of higher complexity).

Last, we remark that all paths on the expanded graph honor additive KPIs constraints, *with the possible exception of delay*. Indeed, unlike other KPIs, whether or not the delay target is violated depends not only on the network latency, hence, the VNF placement, but also on the processing time, i.e., the quantity $a_c(\psi, v, \text{cpu})$ of CPU assigned to each VNF, which in turn impacts the deployment cost. We can account for this important aspect thanks to the M/M/1-PS model used for (2). In particular, below we show how to determine, given a possible deployment, whether there is a CPU assignment consistent with the target delay, and the cost thereof.

[5]The steepness of a path should be not confused with the length of a path.

[6]Using logarithms for reliability values, which are all typically very close to 1, is akin to performing adaptive quantization.

C. Minimizing the cost

We now need (i) for every path found in Sec. IV-B, to identify the minimum-cost CPU assignment, i.e., the optimal values of the $a_c(\psi, v, \text{cpu})$ variables – if such an assignment exists –, and (ii) to determine the path that minimizes the overall cost.

To this end, for each path, hence, for a fixed ψ and for VNFs $v_1 \dots v_N$ to be deployed at computing nodes $\tilde{n}_1 \dots \tilde{n}_N$, respectively, we solve the following problem:

$$\min \sum_{\tilde{n},v} a_{\tilde{n}}(\psi, v, \text{cpu}) c_{\tilde{n}}(\text{cpu}) \qquad \text{s.t.} \tag{9}$$

$$\sum_{\tilde{n}_1,\tilde{n}_2} \left(D_{\tilde{n}_1,\tilde{n}_2} + \frac{1}{a_{\tilde{n}_2}(\psi, v_2, \text{cpu}) - r_{\text{cpu}}(v_2)\lambda_{\tilde{n}_2}(\psi, v_2)} \right) \leq D(s),$$

as well as to constraints concerning link capacity, node capability, and flow conservation, equivalent to those presented in Sec. III. If the problem above is infeasible for a given path, then that path (and the corresponding decisions) is incompatible with the target KPIs and must be discarded.

Once the problem in (9) is solved for all paths identified in the expanded graph, we compute the total cost associated with each path (including all components defined in Sec. III-F) and select and enact the lowest-cost deployment, thus fulfilling OKpi's purpose. Importantly, the problem is *convex*, hence, it can be efficiently solved in polynomial time [22]. The proof simply follows from observing that (i) the objective in (9), as well as the flow conservation and capability constraints, are linear, and (ii) the second derivatives of the delay constraint, are positive in the decision variables, hence, the constraint itself is convex.

D. General scenarios

We now show how OKpi tackles arbitrary scenarios.

1) Arbitrary service graphs: If the service graph is more complex than a chain, we can proceed by decomposing the graph into a set of chains (e.g., in Fig. 2(left), one in uplink, from the MCT to the DB, and one in downlink, from the detector back to the MCT). OKpi is then applied subsequently to each chain, and the deployment decisions are cascaded. The case where multiple endpoints have to be covered, as in Fig. 2(left), is handled in the same way.

2) Multiple VNF instances: If the problem in Sec. IV-C is infeasible for *all* possible paths found in Sec. IV-B, a reason could be the need to split the processing burden across multiple instances of the same VNF. This case is handled by first identifying the bottleneck VNF, i.e., taking the longest to process the service traffic, and then increasing by one the number of instances of that VNF in the service graph. OKpi is then re-run on the modified service graph.

E. OKpi analysis

In this section, we prove several properties about OKpi (proofs can be found in the Appendices). We start with the most essential aspect related to its effectiveness, i.e., its ability to meet all service KPIs (see App.-A for the proof):

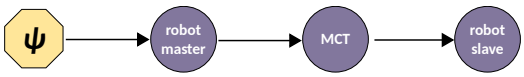


Fig. 5. Service graph specifying the inter-robot communication service. Yellow and purple vertices denote endpoints and VNFs, respectively.

Property 1. *OKpi's decisions honor all KPI targets.*

Next, we address the computational complexity of OKpi (see App.-B for the proof):

Property 2. *The worst-case computational complexity of OKpi (including the graph generation and the solution of (9)) is polynomial.*

Also, in the case where the physical graph is homogeneous, we can prove that OKpi can return the optimal solution (see App.-C for the proof):

Property 3. *If all links and nodes have the same capabilities and cost, then the output of OKpi is optimal.*

Finally, we consider the expanded graph and show that it can be built in polynomial time (see App.-D for the proof):

Property 4. *The worst-case computational complexity of building the expanded graph is* $O\left((\gamma+1)^4 \cdot |\widetilde{N}|^2 \cdot K\right)$.

V. Numerical results

Here, we first focus on a small-scale smart factory scenario and an inter-robot communication service (Sec. V-A), and compare the performance of OKpi against the optimum obtained via brute force. Then, we move to a large-scale smart factory scenario (Sec. V-B), and investigate the impact of the number of robots on the decisions made by OKpi and the resulting performance. Finally, consider a real-world automotive service (Sec. V-C), and we characterize how the quantity of traffic to serve and the maximum delay impact the decisions made by OKpi in a large-scale scenario. The KPI requirements of the services under study are presented in Tab. II.

TABLE II
Service requirements

Requirement	Smart-factory scenarios	Automotive safety scenario	Testbed scenario
bandwidth	1 Mbps	1.5 Mbps	[2, 3] Mbps
reliability	{.999, .9999, .99999}		—
delay	[20, 100] ms	[20, 60] ms	15 ms

A. Small-scale, smart factory scenario: comparison against the optimum

We consider the inter-robot communication service [23], whose graph is depicted in Fig. 5. A room (hence, an endpoint) contains three robots, with different levels of reliability: $\eta(\text{robot1}) = 0.999999$, $\eta(\text{robot2}) = 0.99999$, and $\eta(\text{robot3}) = 0.9999$. Two of these three robots must be used to perform an operation, hence, run the robo-master and robo-slave VNFs.

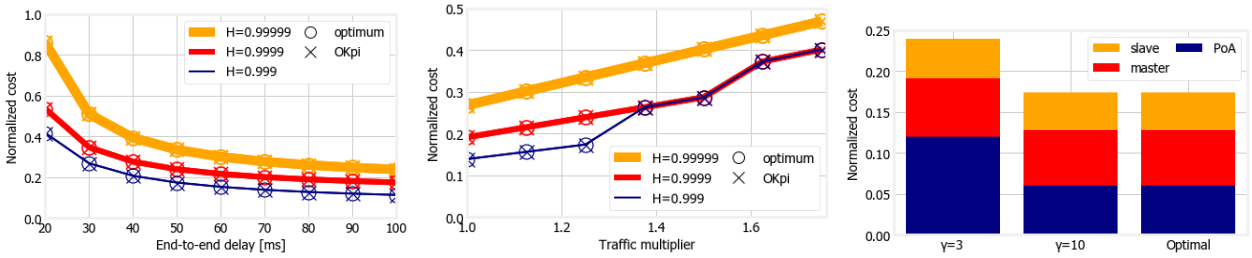


Fig. 6. Small-scale, smart factory scenario (inter-robot communication service): cost as a function of the maximum delay (left) and of the traffic load (center), for different values of target reliability; cost breakdown (right) when the target reliability is 0.999, the maximum delay is 50 ms, the traffic multiplier is 1, and γ varies.

TABLE III
SMALL-SCALE SMART FACTORY SCENARIO : POINTS OF ATTACHMENT AND ROBOT CHARACTERISTICS

Item	Reliability	Latency	Cost
Points of attachment			
micro-cell	0.999999	3 ms	40 USD/Mbit
pico-cell	0.99999	2 ms	30 USD/Mbit
femto-cell	0.9994	1.5 ms	15 USD/Mbit
Robots			
robot1	0.999999	NA	25.49 USD/Mbit
robot2	0.99999		17.04 USD/Mbit
robot3	0.9999		11.36 USD/Mbit

TABLE IV
ALGORITHM RUN TIMES FOR THE SMALL-SCALE SMART FACTORY SCENARIO, WHEN THE TARGET RELIABILITY IS 0.999, THE MAXIMUM DELAY IS 50 MS, AND THE TRAFFIC MULTIPLIER IS 1

γ	Run time [s]
2	2.1
4	2.4
6	2.7
8	3.1
10	3.8
optimal	284.8

The communication between the two selected robots can take place through three types of PoAs, with different levels of reliability (micro-cell: 0.999999, pico-cell: 0.99999, femto-cell: 0.9994), and costs as reported in [24] (see Tab. III). The offered traffic is 1 Mb/s per robot, as specified in [23].

Fig. 6 depicts the results when OKpi's resolution is set to $\gamma = 10$. A first aspect we are interested in is the relationship between the target KPIs and cost: as we can see from Fig. 6(left) and Fig. 6(center), a longer allowable delay results in a lower cost; conversely, a higher traffic load or a higher target reliability both result in higher costs. Intuitively, this is due to the fact that cheaper resources (e.g., robot 3) tend to have lower reliability and/or capacity, hence, it is impossible to use them when the KPI targets become very strict.

Interestingly, in both Fig. 6(left) and Fig. 6(center), OKpi matches the optimum in *all* cases. Indeed, as discussed in Sec. IV-B, OKpi always matches the optimum if the resolution γ is high enough; in the small-scale scenario we consider for Fig. 6, $\gamma = 10$ is sufficient to this end.

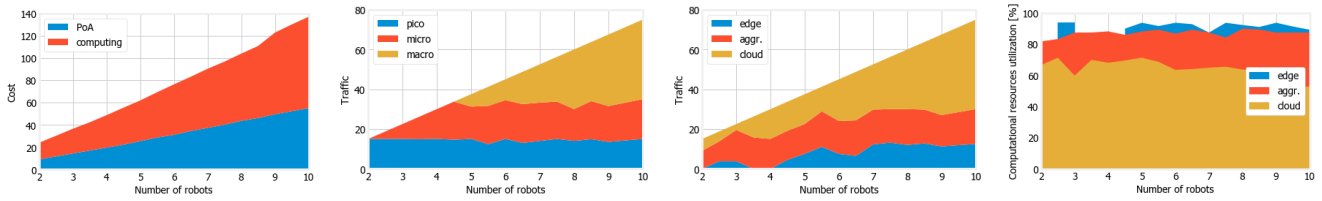
Fig. 6(right) shows the effect of setting a lower resolution, namely, $\gamma = 3$. As we can see by comparing the left and center bars, a lower value of γ results in suboptimal, higher-cost decisions. Specifically, the difference is due to the fact that, when $\gamma = 3$, a higher-cost PoA is selected, namely, the pico-cell *in lieu* of the femto-cell. This happens because, for $\gamma = 3$, the edges corresponding to the femto-cell in the expanded graph have steepness $\left\lceil \gamma \frac{\log 0.9994}{\log 0.999} \right\rceil = 2$. Considering that (i) all other edges have steepness 1 and (ii) OKpi seeks for paths composed of three edges (same as the number of VNFs to place) with a total steepness not exceeding $\gamma = 3$, the edges corresponding to the femto-cell will never be selected, hence, the corresponding decision is never considered. In summary, as discussed in the previous sections, using a too-low γ made us overlook a feasible – and, in this case, optimal – solution.

In the same settings as for Fig. 6(right), Tab. IV presents the time taken by the OKpi algorithm for different values of γ, as well as the time it takes to find the optimal solution. OKpi is implemented in Python, and all tests are run on a server with 40-core Intel Xeon E5-2690 v2 3.00GHz CPU and 64 GB of memory. We can observe that OKpi run times are very short, much shorter than the time it takes to find the optimal solution. Even more interestingly, larger values of γ do result in longer run times, but the increase is substantially slower than the worst-case complexity derived in Property 4.

B. Large-scale, smart factory scenario: impact of the number of robots

To demonstrate the scalability of OKpi, we now move to the scenario depicted in Fig. 1 and the mobile robot, smart factory service in Fig. 2(left), including a set of N robots. This scenario is an enriched version of the small-case, smart factory robotic service, as resource provisioning has to account for more robots, and the smart factory service graph now includes additional VNFs that can be placed on servers rather than on robots.

Different PoAs are considered with different reliability, latency, and cost values, as reported in Tab. V. The front- and back-haul network topology is based on [25] and ITU standard [26], and includes core nodes, aggregation nodes, and local (i.e., close to the PoAs) nodes, with features as summarized in Tab. V [27]. Our goal is to study how the number of users (robots, in this case) impacts the decisions made by OKpi and the resulting performance. Fig. 7(a) shows that, as one might expect, a larger number of robots always results in a higher cost; interestingly, computing nodes and PoAs account for comparable shares of the overall cost.



(a) (b) (c) (d)

Fig. 7. Large-scale, smart factory service with a varying number of robots: cost breakdown (a), choice of PoAs (b) and computing nodes (c), usage of computing resources at different location in the network infrastructure (d).

TABLE V
LARGE-SCALE SCENARIOS: POINTS OF ATTACHMENT AND COMPUTING NODES CHARACTERISTICS

Item	Reliability	Latency	Cost
Points of attachment			
macro-cell	0.99999999	6 ms	1.02 USD/Gbit
micro-cell	0.9999999	3 ms	2.31 USD/Gbit
pico-cell	0.999999	2 ms	3.80 USD/Gbit
Computing nodes			
cloud ring (Azure DataBox)	0.99999999	8 ms	2.23 USD/Gbit
aggregation ring (PowerEdge)	0.9999999	3 ms	5.23 USD/Gbit
local ring (small data center)	0.999999	1 ms	10.47 USD/Gbit

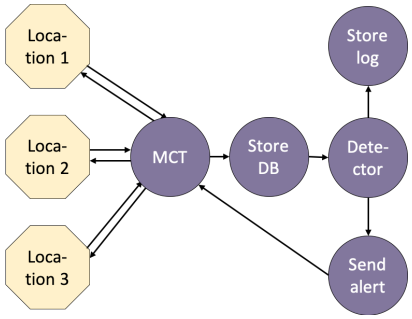


Fig. 8. Service graph of the safety automotive service (vehicle collision avoidance). Yellow and purple vertices denote endpoints and VNFs, respectively.

Fig. 7(b) and Fig. 7(c) summarize how much traffic is handled by different PoAs and computing nodes as the number of robots grows. Fig. 7(b) is fairly straightforward: the cheapest options are always preferred; only after their capacity is exhausted, more expensive PoAs are exploited. Fig. 7(c), concerning computing nodes, shows instead a different situation. The intermediate solution, i.e., aggregation rings, is preferred in most scenarios; edge servers are used for a limited amount of traffic, thanks to their low latency that allows using cheaper (albeit slower) PoAs. Cloud servers, thanks to their low cost and high capacity, are the preferred option when the number of robots grows, provided the target latency can be met.

Fig. 7(d) depicts the amount of computing resources consumed in the different sections of the network infrastructure. Interestingly, VMs closer to users (e.g., at the edge) are consistently used more than further-away ones (e.g., in the cloud). This is due to two reasons: first, more expensive computing nodes can be an optimal choice only if it is possible to fully utilize the VMs therein; second, faster processing times (hence, as specified in (2), more spare capacity) are required at farther-away nodes to make up for longer network delays.

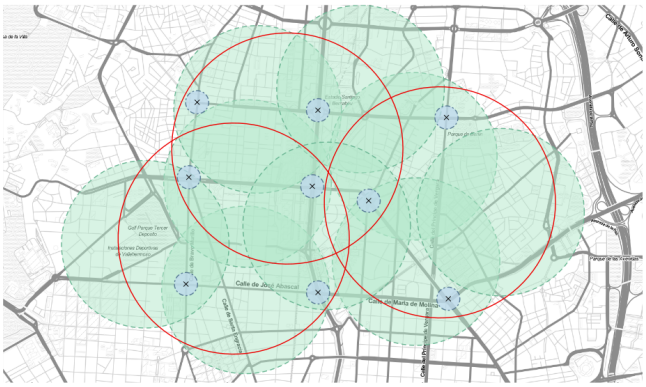


Fig. 9. Road topology used in the large-scale scenario. The nine crossings correspond to endpoints; red, green, and blue circles represent the coverage areas of macro-, micro- and pico-cells, respectively.

C. Large-scale, automotive scenario: impact of traffic and delay

We now test OKpi on a large-scale scenario to validate its performance in presence of more complex service graphs and under a larger, and more diverse, network infrastructure. Specifically, we consider a urban environment where a safety service, namely, vehicle collision avoidance [28], [29], has to be provided at specific intersections. The service graph is depicted in Fig. 8: messages sent by vehicles are collected through the Mobile Communication Transport (MCT), e.g., virtual eNBs plus vEPC, then stored in a database and used for detecting vehicles on a collision course. The latter are warned by sending them an alert. Based on a real-world road topology (see Fig. 9), a total of 9 intersections (hence, endpoints) are covered by a combination of PoAs, namely, macro-, micro- and pico-cells, whose coverage is shown in Fig. 9.

Fig. 10(left) shows that, as one might expect, a shorter target delay results in higher costs. It is also interesting to observe the behavior of the intermediate curve, corresponding to $H(s) = 0.9999$: when the target delay is very short, its associated cost is almost the same as for $H(s) = 0.999999$ case; as the target delay increases, its cost drops to the same level as the $H(s) = 0.999$ case. This bespeaks the complexity of the decisions OKpi has to make, and their sometimes counter-intuitive effects.

In Fig. 10(center), the traffic load is multiplied by a factor ranging between 0.5 and 3. We can again observe that to a higher traffic corresponds a higher cost, even though the

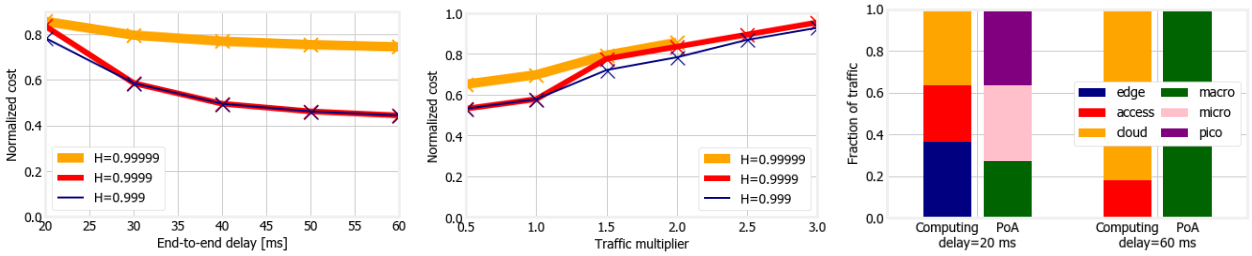


Fig. 10. Large-scale, automotive scenario (safety service): cost as a function of the maximum delay (left) and of the traffic load (center), for different values of target reliability; fraction of traffic (right) traversing different PoAs and computing nodes when the target reliability is 0.999, the traffic multiplier is 1, and the target delay varies.

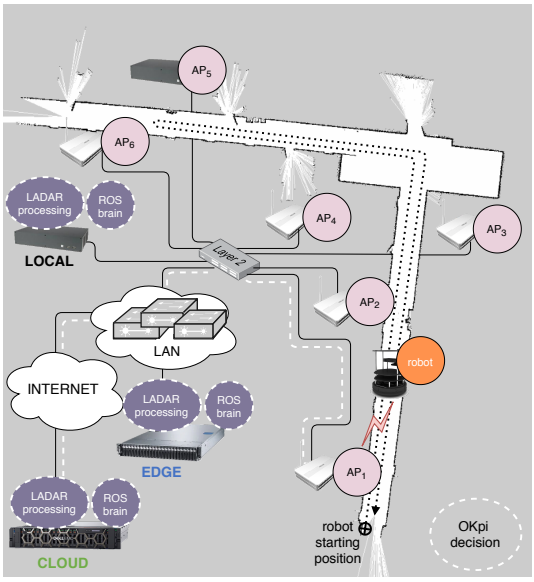


Fig. 11. Illustration of the mobile robot, smart factory testbed: the robot starts at the bottom end of the corridor and comes back once it has reached the other end at the top left. Dashed circles highlight possible VNF deployments.

growth is less than linear, owing to the fixed costs described in Sec. III-F. Also notice how the yellow curve in Fig. 10(center), corresponding to the highest reliability level, stops at a multiplier of 2: for higher traffic demands, the network capacity is insufficient to provide the service with the required reliability.

Fig. 10(right) shows which PoAs and computing nodes are selected for the minimum and maximum target delay values. Interestingly, in the presence of tight delay constraints, different PoAs and resources are all used (left bars). On the contrary, for the largest target delay, the cheapest options – cloud and macro-cells – are preferred.

VI. Testbed and Validation

We now present the implementation of OKpi in an experimental testbed where the mobile robot, smart factory service (depicted in Fig. 2(left)) is deployed in an indoor scenario.

The testbed consists of: (i) 5 ASUS WL500G Premium v1 APs running OpenWrt 18.06.2 [30]; (ii) 2 MiniPC, with 4 vCPUs and 8GB of RAM each, one used as an AP and the latter as a local server (i.e., located close to the APs); (iii) 1 PowerEdge C6220 server with 94GB of RAM and 16 vCPUS, acting as edge server; and (iv) 1 PowerEdge R840 Rack Server7 with 94GB of RAM and 16 vCPUs, acting as cloud server. The six APs and the local server are deployed along two corridors of the Universidad Carlos III de Madrid building (see Fig. 11), while the edge and cloud server are located in different buildings. To emulate different levels of link congestion, we leverage NetEm [31] to artificially introduce some latency on the connection between the APs and the servers, as reported in Tab. VI. Note instead that the latency on the AP-robot link never exceeds 6 ms. The cost associated with the APs and servers match those presented in Tab. V, considering the pico-cell value for the APs. Additionally, we used a ROS-compatible Kobuki Turtlebot S2 robot equipped with a laptop with 8-GB RAM and 2 vCPUs, and a RPLIDAR A2 lidar for 360-degree omnidirectional laser range scanning.

TABLE VI
Testbed scenario: AP-server latencies

	Cloud	Edge	Local
AP_1, AP_2	9 ms	4 ms	3 ms
AP_3, AP_4	18 ms	8 ms	9 ms
AP_5, AP_6	27 ms	12 ms	9 ms

The laptop hosts the robot VNF, which, as mentioned in Sec. II, (a) probes the robot sensors (e.g., odometry, LIDAR), (b) transmits the sensors data to the ROS brain, and (c) executes the navigation instructions received from the ROS brain. The LADAR and ROS brain VNFs can be hosted at any of the available servers

The target of the experiment is to ensure that the one-way, end-to-end (e2e) latency of the service remains within 15 ms [32] during the robot's trip. The experiment starts with the robot positioned at the bottom end of the corridor and connected with AP_1 (see Fig. 11). Also, the initial decision by OKpi is to deploy the ROS brain and LADAR VNFs in the cloud server. The ROS brain then navigates the robot along the trajectory shown in Fig. 11 and, as the robot moves, OKpi determines which AP the robot should connect to[7] and which server (cloud, edge, local) should host the ROS brain and LADAR VNFs. Both the AP and server selection change according to the robot position and latency of the AP, respectively. In particular, depending on which AP the robot is attached to, OKpi decides which server should host the ROS

[7]As the robot moves, it roams to the selected AP using 802.11r Fast Transitioning [33].

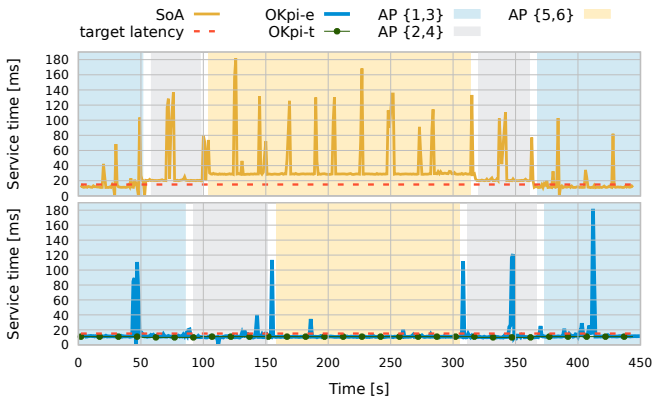


Fig. 12. Service latency experienced during the robot's trip. Different background colors refer to time intervals in which the robot is connected to different APs.

TABLE VII
OKPI AND SOA SERVICE LATENCY

Solution	Average	Std. deviation	E2e violations
$OKpi_e$	13.63 ms	15.21 ms	7%
SoA	29.61 ms	26.18 ms.	74%

brain and LADAR VNFs by accounting for the latency values reported in Table VI, in such a way that the overall service latency remains below 15 ms. During the experiments, OKpi recomputed the AP selection, ROS brain, and LADAR VNFs embedding, in less than 1 s. The robot position is reported to OKpi through MQTT messages transmitted by the robot itself, while the APs coverage, which may vary over time, is acquired through locally available channel measurements (e.g., signal level).

Fig. 12 compares the temporal behavior of the e2e latency obtained in the following cases:

- **SoA**, i.e., when the solution based on [34] and [35] is used. In this case, the LADAR and ROS brain VNFs are always placed in the cloud and the robot performs roaming checks every 20 s in order to ensure connectivity to the AP with strongest signal. The roaming checks consist of Wi-Fi active scan to acquire the signal related information, followed by automatic handover. For our testbed, we discovered that 20 s was the lowest check frequency that allows the robot to perform successful roaming along the navigated trajectory.
- **$OKpi_t$**, i.e., when OKpi makes decisions in a simulated environment that mimics the testbed, using the theoretical delays that each AP offers.
- **$OKpi_e$**, i.e., when OKpi operates in the developed testbed, it uses the robot's real time location and available channel measurements to trigger the 802.11r roaming. The roaming message is send via MQTT to the robot. In this case, the periodic roaming checks are prevented and the robot is configured to perform handovers only when roaming message is received.

Fig. 12 compares the service time measured during a single run execution of the SoA, $OKpi_e$, and $OKpi_t$ solutions.

At the beginning of the experiment, neither the SoA nor the $OKpi_e$ violate the target e2e latency of 15 ms, except for the peaks due to the robot Wi-Fi scans. While under SoA the robot performs a scan every 20 s, under $OKpi_e$ it does so only when it has to connect to a new AP, as per the OKpi decision. Under SoA, 50 s later the robot connects to AP_3, and the latency jumps above 20 ms because the LADAR and ROS brain VNFs are still running in the cloud server. In the case of $OKpi_e$, instead, when the robot connects to AP_3 at 89 s., the LADAR and ROS brain VNFs are moved to the edge server, so that the e2e latency remains below 15 ms. The same behavior is observed at time 102 s, when the robot connects to AP_5 in the case of SoA and the e2e latency increases up to 29 ms. On the contrary, $OKpi_e$ still meets the target e2e latency upon making the robot connecting with AP_5 (at time 156 s), since it now places the VNFs in the local server. In the rest of the time elapse, we can observe similar performance, as the robot returns to its initial position.

As a final remark, Fig. 12 highlights that the performance of $OKpi_e$ is always close to the e2e latency exhibited by $OKpi_t$. Furthermore, the target e2e service latency (15 ms) was only violated the 7% of the times during the experiment by $OKpi_e$ (see Table VII), while it was violated the 74% of the times under SoA.

VII. RELATED WORK

One of the pioneering works on VNF placement is [16], which casts placement as a generalized assignment problem (GAP) and proposes a near-optimal solution based on bi-criteria approximation. Very recent works [10], [12], [19] focus on the mutual influence of VNF placement and traffic routing. Others tackle the VNF placement problem through graph theory [9], [36] and set-covering [5], obtaining very good competitive ratios (constant in specific cases for [5]).

In the context of edge computing, some works tackle tasks different from sheer data processing; as an example, [11], [37] aim at jointly optimizing computation and caching offloading between cloud-based and edge-based infrastructure. Others focus on additional decisions that can be made in slicing scenarios, e.g., priority assignment in [4]. A body of works considers incremental deployment, i.e., service requests arriving at different times: in this case, it is possible to share existing VNF instances [4], [7], [38], augment routing paths instead of computing them from scratch [7], [38], and minimize the difference between current and future network configuration [38]. Among the few works tackling non-functional requirements, [3] performs resilient VNF placement, to achieve robustness to equipment failures. More recently, [13] considered the problem of jointly placing the VNFs and the data they need.

VNF placement, along with the closely-related problem of VNF chaining, has been studied in the software-defined networking and cloud-computing contexts as well. For instance, [39] focuses on updating the placement in order to react to traffic changes, and [40] deals with the parallelization opportunities offered by VNF graphs. Other works [10], [41] focus on the choice between edge- and cloud-based computation resources, while [8] studies which cache storage (i.e., edge- or cloud-based) to access, balancing miss probability and cost.

Several works aim at simplifying the problem of VNF placement by characterizing and/or predicting the traffic demand. In particular, [42] exploits the spatial and temporal variability of traffic demand to serve it with as little resources as possible; as for demand prediction, popular approaches include reinforcement learning [43]. In a similar spirit, [44] estimates the resources needed by an incoming service request before deciding whether or not it shall be accepted. A different body of work instead addresses the slicing of the radio access network; in particular, [45] proposes solutions that let different virtual operators use the radio resources without interfering, while [46] develops a stochastic model to investigate the throughput and delay of a slice as functions of the cell parameters. Although such specific aspects are out of the scope of our work, we do tackle the problem of selecting radio technologies and points of attachment that honor the required KPI targets and minimize the cost.

Finally, we mention that a preliminary version of this work has appeared in our conference paper in [1] where, however, the experimental implementation of our solutions as well as some proofs very missing, and performance was showed only through numerical results and under single-service scenarios.

VIII. Conclusions

One of the paramount issues in network slicing is to meet all target KPIs required by vertical services, in spite of the limited resources that are available in the mobile network. We addressed this problem by proposing OKpi, an efficient and effective solution strategy that can jointly make VNF placement and data routing decisions (including the selection of the radio points of attachment), while accounting for all resources that may be available from the fog to the cloud. OKpi draws on a novel methodology that leverages graph-based models blended with optimization. We analyzed the properties and the computational complexity of OKpi, which turns out to be polynomial, and we evaluated the performance of the proposed solution both numerically and in a real-world testbed. Numerical results demonstrate the performance of our solution in the presence of different relevant applications. In all performed experiments, OKpi showed to closely match the optimum. Finally, field tests demonstrated the functionality of OKpi and its ability to make effective decisions also when a real-world mobile robot, smart factory service is considered.

Future work will study in more detail how the γ parameter affects the optimality vs. execution time tradeoff, so that the γ parameter can be tuned so as not to not exceed a given timeout threshold. This is of special interest for use cases in which fast deployments are required, such as mission critical services.

References

[1] J. Martín-Pérez, F. Malandrino, C. F. Chiasserini, and C. J. Bernardos, "OKpi: All-KPI network slicing through efficient resource allocation," in *IEEE INFOCOM*, 2020.

[2] ETSI. (2018) MEC in 5G networks. https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf.

[3] P. Zhao and G. Dán, "A Benders decomposition approach for resilient placement of virtual process control functions in mobile edge clouds," *IEEE Transactions on Network and Service Management*, 2018.

[4] F. Malandrino and C.-F. Chiasserini, "Getting the Most Out of Your VNFs: Flexible Assignment of Service Priorities in 5G," in *IEEE WoWMoM*, 2019.

[5] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *IEEE INFOCOM*, 2017.

[6] "IMT Vision: Framework and overall objectives of the future development of IMT for 2020 and beyond," *ITU Recommendation M.2083-0*, 2015.

[7] T. Lukovszki, M. Rost, and S. Schmid, "Approximate and incremental network function placement," *Elsevier Journal of Parallel and Distributed Computing*, 2018.

[8] I. Cohen, G. Einziger, R. Friedman, and G. Scalosub, "Access Strategies for Network Caching," in *IEEE INFOCOM*, 2019.

[9] S. Dräxler, H. Karl, and Z. Á. Mann, "Jasper: Joint optimization of scaling, placement, and routing of virtual network services," *IEEE Transactions on Network and Service Management*, 2018.

[10] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint Service Placement and Request Routing in Multi-cell Mobile Edge Computing Networks," in *IEEE INFOCOM*, 2019.

[11] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM*, 2018.

[12] S. Agarwal, F. Malandrino, C. F. Chiasserini, and S. De, "VNF Placement and Resource Allocation for the Support of Vertical Services in 5G Networks," *IEEE/ACM Transactions on Networking*, 2019.

[13] K. Kamran, E. Yeh, and Q. Ma, "Deco: Joint computation, caching and forwarding in data-centric computing networks," in *ACM MobiHoc*, 2019.

[14] 5G PPP. Key Performance Indicators. https://5g-ppp.eu/kpis/.

[15] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying middlebox policy enforcement using SDN," *ACM SIGCOMM computer communication review*.

[16] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *IEEE INFOCOM*, 2015.

[17] F. B. Jemaa, G. Pujolle, and M. Pariente, "Qos-aware vnf placement optimization in edge-central carrier cloud architecture," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–7.

[18] D. B. Oljira, K.-J. Grinnemo, J. Taheri, and A. Brunstrom, "A model for qos-aware vnf placement and provisioning," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2017, pp. 1–7.

[19] H. Feng, J. Llorca, A. M. Tulino, D. Raz, and A. F. Molisch, "Approximation algorithms for the NFV service distribution problem," in *IEEE INFOCOM*, 2017.

[20] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1979.

[21] G. Xue, A. Sen, W. Zhang, J. Tang, and K. Thulasiraman, "Finding a path subject to many additive QoS constraints," *IEEE/ACM Transactions on Networking*, 2007.

[22] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[23] "5G-CORAL initial system design, use cases, and requirements," *5G-CORAL deliverable 1.1*, 2018, http://5g-coral.eu/wp-content/uploads/2018/04/D1.1_final7760.pdf.

[24] V. Nikolikja and T. Janevski, "A cost modeling of high-capacity LTE-Advanced and IEEE 802.11ac based heterogeneous networks, deployed in the 700 MHz, 2.6 GHz and 5 GHz bands," *Procedia Computer Science*, 2014.

[25] J. Martín-Pérez, L. Cominardi, C. J. Bernardos, A. De la Oliva, and A. Azcorra, "Modeling mobile edge computing deployments for low latency multimedia services," *IEEE Transactions on Broadcasting*, 2019.

[26] "Consideration on 5G transport network reference architecture and bandwidth requirements," *ITU Contribution 0462*, 2018.

[27] "Final system design and techno-economic analysis," *5G-TRANSFORMER deliverable D1.4*, 2019.

[28] G. Avino, P. Bande, P. A. Frangoudis, C. Vitale, C. Casetti, C. F. Chiasserini, K. Gebru, A. Ksentini, and G. Zennaro, "A MEC-based Extended Virtual Sensing for Automotive Services," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1450–1463, 2019.

[29] M. Malinverno, G. Avino, C. Casetti, C. F. Chiasserini, F. Malandrino, and S. Scarpina, "Edge-based collision avoidance for vehicles and vulnerable users: An architecture based on MEC," *IEEE Vehicular Technology Magazine*, vol. 15, no. 1, pp. 27–35, 2020.

[30] F. Fainelli, "The openwrt embedded development framework," in *Proceedings of the Free and Open Source Software Developers European Meeting*. sn, 2008, p. 106.
[31] S. Hemminger *et al.*, "Network emulation with netem," in *Linux conf au*, 2005, pp. 18–23.
[32] 3GPP, "Study on Communication for Automation in Vertical Domains," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 22.804, 12 2018, version 16.2.0.
[33] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: Fast Basic Service Set (BSS) Transition*, IEEE Std. 802.11r-2008, 07 2008.
[34] J. Malinen, *Developers' documentation for wpa_supplicant/hostapd*, 0th ed., November 2009. [Online]. Available: http://w1.fi/wpa_supplicant/wpa_supplicant-devel.pdf
[35] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.
[36] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent NFV middleboxes," in *IEEE INFOCOM*, 2017.
[37] M. Chen, Y. Hao, L. Hu, M. S. Hossain, and A. Ghoneim, "Edge-CoCaCo: Toward joint optimization of computation, caching, and communication on edge cloud," *IEEE Wireless Communications*, 2018.
[38] T. Lukovszki, M. Rost, and S. Schmid, "It's a match!: Near-optimal and incremental middlebox deployment," *ACM SIGCOMM Computer Communication Review*, 2016.
[39] S. G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K. Ramakrishnan, T. Wood, M. Arumaithurai, and X. Fu, "NFVnice: Dynamic backpressure and scheduling for NFV service chains," in *ACM SIGCOMM*, 2017.
[40] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "NFP: Enabling network function parallelism in NFV," in *ACM SIGCOMM*, 2017.
[41] Y. Guo, A. L. Stolyar, and A. Walid, "Shadow-routing based dynamic algorithms for virtual machine placement in a network cloud," *IEEE Transactions on Cloud Computing*, 2015.
[42] M. Bouet and V. Conan, "Mobile edge computing resources optimization: a geo-clustering approach," *IEEE Transactions on Network and Service Management*, 2018.
[43] V. Sciancalepore, F. Z. Yousaf, and X. Costa-Perez, "z-TORCH: An automated NFV orchestration and monitoring solution," *IEEE Transactions on Network and Service Management*, 2018.
[44] B. Han, V. Sciancalepore, D. Feng, X. Costa-Perez, and H. D. Schotten, "A Utility-Driven Multi-Queue Admission Control Solution for Network Slicing," in *IEEE INFOCOM*, 2019.
[45] S. D'Oro, F. Restuccia, A. Talamonti, and T. Melodia, "The slice is served: Enforcing radio access network slicing in virtualized 5G systems," in *IEEE INFOCOM*, 2019.
[46] V. Mancuso, P. Castagno, M. Sereno, and M. A. Marsan, "Slicing cell resources: The case of HTC and MTC coexistence," in *IEEE INFOCOM*, 2019.
[47] R. Seidel, "On the all-pairs-shortest-path problem in unweighted undirected graphs," *Journal of computer and system sciences*, 1995.

APPENDIX: PROOFS OF OKPI PROPERTIES

A. Property 1

Proof: By Lemma 1, all decisions honor the additive KPIs. Concerning delay, it is guaranteed that such a KPI target is met, thanks to the delay constraint imposed while performing the CPU assignment. As noted in Sec. IV-C, decisions resulting in an infeasible problem are discarded, hence, the selected decision honors the delay target. Finally, the availability constraints are satisfied through the initial selection of the vertices of the decision graph (see Sec. IV-B). ■

B. Property 2

Proof: To prove the property, we show that each of the steps described in Sec. IV has a polynomial runtime. Specifically,

(i) creating the decision/expanded graph (Sec. IV-B and Sec. IV-A) requires creating at most $\gamma^2(|\mathcal{V}||\mathcal{C}|+|\mathcal{E}|)$ nodes and at most $\gamma^2|\mathcal{V}||\mathcal{L}|$ edges, where $|\mathcal{V}|$ is the number of VNFs specifying the service and, given the service, is a constant.
(ii) Finding the possible decisions (Sec. IV-B) implies computing the shortest paths between any endpoint (i.e., vertex meeting the availability constraints) and any other node in the expanded graph, which, in the worst case, has complexity [47] $o(n^{2.3})$ with n being the number of nodes in the expanded graph.
(iii) Computing the optimal CPU assignments (Sec. IV-C) requires solving a convex optimization problem, which has cubic complexity [22] in the problem size; indeed, convex problems are routinely solved in embedded computing scenarios.

Thus, the overall time complexity of the OKpi approach is polynomial. ■

C. Property 3

Proof: There is only one point in the procedure we described where, in general scenarios, we may overlook the optimal solution. As remarked in Sec. IV-B, finite γ values may cause a quantization-like error: solutions with different KPI consumption and/or cost can be associated with the same path over the extended graph; therefore, the extended graph may not consider all possible ways to move from one node of the decision graph to another. In the special case of homogeneous links and nodes, however, no such different possibilities exist: taking a finite value of γ is enough to consider all possible choices the system offers and, hence, to make an optimal decision. Note that restricting our attention to shortest paths on the expanded graph does not harm optimality, as adding hops implies consuming a higher (or equal at best) fraction of KPI targets and cannot decrease the cost. ■

D. Property 4

Proof: As noted in Sec. IV-B, given that we have two additive KPIs, the expanded graph has $(\gamma+1)^2$ nodes for each node in the decision graph, (with the number of nodes in the decision graph being $|\widetilde{N}|$). Therefore, the total number of pairs in the expanded graph is $O\left(\left[(\gamma+1)^2|\widetilde{N}|\right]^2\right)$, and we need to evaluate whether or not an edge shall be created for each of these pairs. Doing so requires checking each KPI, for a total of $O\left((\gamma+1)^4|\widetilde{N}|^2K\right)$ checks, where K is the number of KPIs. It follows that the global, worst-case complexity of building the expanded graph is quadratic in the network topology and polynomial in γ. ■